

# Package ‘sparklyr’

March 25, 2024

**Type** Package

**Title** R Interface to Apache Spark

**Version** 1.8.5

**Maintainer** Edgar Ruiz <edgar@rstudio.com>

**Description** R interface to Apache Spark, a fast and general engine for big data processing, see <<https://spark.apache.org/>>. This package supports connecting to local and remote Apache Spark clusters, provides a 'dplyr' compatible back-end, and provides an interface to Spark's built-in machine learning algorithms.

**License** Apache License 2.0 | file LICENSE

**URL** <https://spark.posit.co/>

**BugReports** <https://github.com/sparklyr/sparklyr/issues>

**Depends** R (>= 3.2)

**Imports** config (>= 0.2), DBI (>= 1.0.0), dbplyr (>= 2.5.0), dplyr (>= 1.0.9), generics, globals, glue, httr (>= 1.2.1), jsonlite (>= 1.4), methods, openssl (>= 0.8), purrr, rlang (>= 0.1.4), rstudioapi (>= 0.10), tidyr (>= 1.2.0), tidyselect, uuid, vctrs, withr, xml2

**Suggests** arrow (>= 0.17.0), broom, diffobj, foreach, ggplot2, iterators, janeaustenr, Lahman, mlbench, nnet, nycflights13, R6, r2d3, RCurl, reshape2, shiny (>= 1.0.1), parsnip, testthat, rprojroot

**Encoding** UTF-8

**RoxygenNote** 7.3.0

**SystemRequirements** Spark: 1.6.x, 2.x, or 3.x

**Collate** 'spark\_data\_build\_types.R' 'arrow\_data.R' 'spark\_invoke.R' 'browse\_url.R' 'spark\_connection.R' 'avro\_utils.R' 'config\_settings.R' 'config\_spark.R' 'connection\_instances.R' 'connection\_progress.R' 'connection\_shinyapp.R' 'spark\_version.R' 'connection\_spark.R' 'core\_arrow.R' 'core\_config.R' 'core\_connection.R' 'core\_deserialize.R'

'core\_gateway.R' 'core\_invoke.R' 'core\_job.R'  
'core\_serialize.R' 'core\_utils.R' 'core\_worker\_config.R'  
'utils.R' 'sql\_utils.R' 'data\_copy.R' 'data\_csv.R'  
'spark\_schema\_from\_rdd.R' 'spark\_apply\_bundle.R'  
'spark\_apply.R' 'tables\_spark.R' 'tbl\_spark.R' 'spark\_sql.R'  
'spark\_dataframe.R' 'dplyr\_spark.R' 'sdf\_interface.R'  
'data\_interface.R' 'databricks\_connection.R'  
'dbi\_spark\_connection.R' 'dbi\_spark\_result.R'  
'dbi\_spark\_table.R' 'do\_spark.R' 'dplyr\_do.R' 'dplyr\_hof.R'  
'dplyr\_join.R' 'dplyr\_spark\_data.R' 'dplyr\_spark\_table.R'  
'stratified\_sample.R' 'sdf\_sql.R' 'dplyr\_sql.R'  
'dplyr\_sql\_translation.R' 'dplyr\_verbs.R' 'imports.R'  
'install\_spark.R' 'install\_spark\_versions.R'  
'install\_spark\_windows.R' 'install\_tools.R' 'java.R'  
'jobs\_api.R' 'kubernetes\_config.R' 'shell\_connection.R'  
'livy\_connection.R' 'livy\_install.R' 'livy\_invoke.R'  
'livy\_service.R' 'ml\_clustering.R'  
'ml\_classification\_decision\_tree\_classifier.R'  
'ml\_classification\_gbt\_classifier.R'  
'ml\_classification\_linear\_svc.R'  
'ml\_classification\_logistic\_regression.R'  
'ml\_classification\_multilayer\_perceptron\_classifier.R'  
'ml\_classification\_naive\_bayes.R'  
'ml\_classification\_one\_vs\_rest.R'  
'ml\_classification\_random\_forest\_classifier.R'  
'ml\_model\_helpers.R' 'ml\_clustering\_bisecting\_kmeans.R'  
'ml\_clustering\_gaussian\_mixture.R' 'ml\_clustering\_kmeans.R'  
'ml\_clustering\_lda.R' 'ml\_clustering\_power\_iteration.R'  
'ml\_constructor\_utils.R' 'ml\_evaluate.R'  
'ml\_evaluation\_clustering.R' 'ml\_evaluation\_prediction.R'  
'ml\_evaluator.R' 'ml\_feature\_binarizer.R'  
'ml\_feature\_bucketed\_random\_projection\_lsh.R'  
'ml\_feature\_bucketizer.R' 'ml\_feature\_chisq\_selector.R'  
'ml\_feature\_count\_vectorizer.R' 'ml\_feature\_dct.R'  
'ml\_feature\_sql\_transformer.R' 'ml\_feature\_dplyr\_transformer.R'  
'ml\_feature\_elementwise\_product.R'  
'ml\_feature\_feature\_hasher.R' 'ml\_feature\_hashing\_tf.R'  
'ml\_feature\_idf.R' 'ml\_feature\_imputer.R'  
'ml\_feature\_index\_to\_string.R' 'ml\_feature\_interaction.R'  
'ml\_feature\_lsh\_utils.R' 'ml\_feature\_max\_abs\_scaler.R'  
'ml\_feature\_min\_max\_scaler.R' 'ml\_feature\_minhash\_lsh.R'  
'ml\_feature\_ngram.R' 'ml\_feature\_normalizer.R'  
'ml\_feature\_one\_hot\_encoder.R'  
'ml\_feature\_one\_hot\_encoder\_estimator.R' 'ml\_feature\_pca.R'  
'ml\_feature\_polynomial\_expansion.R'  
'ml\_feature\_quantile\_discretizer.R' 'ml\_feature\_r\_formula.R'  
'ml\_feature\_regex\_tokenizer.R' 'ml\_feature\_robust\_scaler.R'  
'ml\_feature\_standard\_scaler.R'

'ml\_feature\_stop\_words\_remover.R' 'ml\_feature\_string\_indexer.R'  
 'ml\_feature\_string\_indexer\_model.R' 'ml\_feature\_tokenizer.R'  
 'ml\_feature\_vector\_assembler.R' 'ml\_feature\_vector\_indexer.R'  
 'ml\_feature\_vector\_slicer.R' 'ml\_feature\_word2vec.R'  
 'ml\_fpm\_fpgrowth.R' 'ml\_fpm\_prefixspan.R' 'ml\_helpers.R'  
 'ml\_mapping\_tables.R' 'ml\_metrics.R' 'ml\_model\_als.R'  
 'ml\_model\_bisecting\_kmeans.R' 'ml\_model\_constructors.R'  
 'ml\_model\_decision\_tree.R' 'ml\_model\_gaussian\_mixture.R'  
 'ml\_model\_generalized\_linear\_regression.R'  
 'ml\_model\_gradient\_boosted\_trees.R'  
 'ml\_model\_isotonic\_regression.R' 'ml\_model\_kmeans.R'  
 'ml\_model\_lda.R' 'ml\_model\_linear\_regression.R'  
 'ml\_model\_linear\_svc.R' 'ml\_model\_logistic\_regression.R'  
 'ml\_model\_naive\_bayes.R' 'ml\_model\_one\_vs\_rest.R'  
 'ml\_model\_random\_forest.R' 'ml\_model\_utils.R'  
 'ml\_param\_utils.R' 'ml\_persistence.R' 'ml\_pipeline.R'  
 'ml\_pipeline\_utils.R' 'ml\_print\_utils.R'  
 'ml\_recommendation\_als.R'  
 'ml\_regression\_aft\_survival\_regression.R'  
 'ml\_regression\_decision\_tree\_regressor.R'  
 'ml\_regression\_gbt\_regressor.R'  
 'ml\_regression\_generalized\_linear\_regression.R'  
 'ml\_regression\_isotonic\_regression.R'  
 'ml\_regression\_linear\_regression.R'  
 'ml\_regression\_random\_forest\_regressor.R' 'ml\_stat.R'  
 'ml\_summary.R' 'ml\_transformation\_methods.R'  
 'ml\_transformer\_and\_estimator.R' 'ml\_tuning.R'  
 'ml\_tuning\_cross\_validator.R'  
 'ml\_tuning\_train\_validation\_split.R' 'ml\_utils.R'  
 'ml\_validator\_utils.R' 'mutation.R' 'na\_actions.R'  
 'new\_model\_multilayer\_perceptron.R' 'params\_validator.R'  
 'precondition.R' 'project\_template.R' 'qubole\_connection.R'  
 'reexports.R' 'sdf\_dim.R' 'sdf\_distinct.R' 'sdf\_ml.R'  
 'sdf\_saveload.R' 'sdf\_sequence.R' 'sdf\_stat.R'  
 'sdf\_streaming.R' 'tidyr\_utils.R' 'sdf\_unnest\_longer.R'  
 'sdf\_wrapper.R' 'sdf\_unnest\_wider.R' 'sdf\_utils.R'  
 'spark\_compile.R' 'spark\_context\_config.R' 'spark\_extensions.R'  
 'spark\_gateway.R' 'spark\_gen\_embedded\_sources.R'  
 'spark\_globals.R' 'spark\_hive.R' 'spark\_home.R' 'spark\_ide.R'  
 'spark\_submit.R' 'spark\_update\_embedded\_sources.R'  
 'spark\_utils.R' 'spark\_verify\_embedded\_sources.R'  
 'stream\_data.R' 'stream\_job.R' 'stream\_operations.R'  
 'stream\_shiny.R' 'stream\_view.R' 'synapse\_connection.R'  
 'test\_connection.R' 'tidiers\_ml\_aft\_survival\_regression.R'  
 'tidiers\_ml\_als.R' 'tidiers\_ml\_isotonic\_regression.R'  
 'tidiers\_ml\_lda.R' 'tidiers\_ml\_linear\_models.R'  
 'tidiers\_ml\_logistic\_regression.R'  
 'tidiers\_ml\_multilayer\_perceptron.R' 'tidiers\_ml\_naive\_bayes.R'

'tidiers\_ml\_svc\_models.R' 'tidiers\_ml\_tree\_models.R'  
 'tidiers\_ml\_unsupervised\_models.R' 'tidiers\_pca.R'  
 'tidiers\_utils.R' 'tidyr\_fill.R' 'tidyr\_nest.R'  
 'tidyr\_pivot\_utils.R' 'tidyr\_pivot\_longer.R'  
 'tidyr\_pivot\_wider.R' 'tidyr\_separate.R' 'tidyr\_unite.R'  
 'tidyr\_unnest.R' 'worker\_apply.R' 'worker\_connect.R'  
 'worker\_connection.R' 'worker\_invoke.R' 'worker\_log.R'  
 'worker\_main.R' 'yarn\_cluster.R' 'yarn\_config.R' 'yarn\_ui.R'  
 'zzz.R'

**NeedsCompilation** no

**Author** Javier Luraschi [aut],

Kevin Kuo [aut] (<<https://orcid.org/0000-0001-7803-7901>>),

Kevin Ushey [aut],

JJ Allaire [aut],

Samuel Macedo [ctb],

Hossein Falaki [aut],

Lu Wang [aut],

Andy Zhang [aut],

Yitao Li [aut] (<<https://orcid.org/0000-0002-1261-905X>>),

Jozef Hajnala [ctb],

Maciej Szymkiewicz [ctb] (<<https://orcid.org/0000-0003-1469-9396>>),

Wil Davis [ctb],

Edgar Ruiz [aut, cre],

RStudio [cph],

The Apache Software Foundation [aut, cph]

**Repository** CRAN

**Date/Publication** 2024-03-25 21:40:02 UTC

## R topics documented:

|                                    |    |
|------------------------------------|----|
| checkpoint_directory . . . . .     | 10 |
| collect_from_rds . . . . .         | 11 |
| compile_package_jars . . . . .     | 11 |
| connection_config . . . . .        | 12 |
| copy_to.spark_connection . . . . . | 12 |
| distinct . . . . .                 | 13 |
| download_scalac . . . . .          | 14 |
| dplyr_hof . . . . .                | 14 |
| ensure . . . . .                   | 14 |
| fill . . . . .                     | 15 |
| filter . . . . .                   | 15 |
| find_scalac . . . . .              | 15 |
| ft_binarizer . . . . .             | 16 |
| ft_bucketizer . . . . .            | 17 |
| ft_chisq_selector . . . . .        | 19 |
| ft_count_vectorizer . . . . .      | 20 |

|  |    |
|--|----|
| ft_dct . . . . .                               | 22 |
| ft_elementwise_product . . . . .               | 23 |
| ft_feature_hasher . . . . .                    | 24 |
| ft_hashing_tf . . . . .                        | 26 |
| ft_idf . . . . .                               | 27 |
| ft_imputer . . . . .                           | 28 |
| ft_index_to_string . . . . .                   | 29 |
| ft_interaction . . . . .                       | 30 |
| ft_lsh . . . . .                               | 31 |
| ft_lsh_utils . . . . .                         | 33 |
| ft_max_abs_scaler . . . . .                    | 34 |
| ft_min_max_scaler . . . . .                    | 35 |
| ft_ngram . . . . .                             | 37 |
| ft_normalizer . . . . .                        | 38 |
| ft_one_hot_encoder . . . . .                   | 39 |
| ft_one_hot_encoder_estimator . . . . .         | 40 |
| ft_pca . . . . .                               | 41 |
| ft_polynomial_expansion . . . . .              | 43 |
| ft_quantile_discretizer . . . . .              | 44 |
| ft_regex_tokenizer . . . . .                   | 46 |
| ft_robust_scaler . . . . .                     | 47 |
| ft_r_formula . . . . .                         | 48 |
| ft_sql_transformer . . . . .                   | 50 |
| ft_standard_scaler . . . . .                   | 51 |
| ft_stop_words_remover . . . . .                | 53 |
| ft_string_indexer . . . . .                    | 54 |
| ft_tokenizer . . . . .                         | 56 |
| ft_vector_assembler . . . . .                  | 57 |
| ft_vector_indexer . . . . .                    | 58 |
| ft_vector_slicer . . . . .                     | 59 |
| ft_word2vec . . . . .                          | 60 |
| full_join . . . . .                            | 61 |
| generic_call_interface . . . . .               | 62 |
| get_spark_sql_catalog_implementation . . . . . | 62 |
| hive_context_config . . . . .                  | 63 |
| hof_aggregate . . . . .                        | 63 |
| hof_array_sort . . . . .                       | 64 |
| hof_exists . . . . .                           | 65 |
| hof_filter . . . . .                           | 65 |
| hof_forall . . . . .                           | 66 |
| hof_map_filter . . . . .                       | 67 |
| hof_map_zip_with . . . . .                     | 68 |
| hof_transform . . . . .                        | 69 |
| hof_transform_keys . . . . .                   | 70 |
| hof_transform_values . . . . .                 | 71 |
| hof_zip_with . . . . .                         | 71 |
| inner_join . . . . .                           | 72 |
| invoke . . . . .                               | 73 |

|                                     |     |
|-------------------------------------|-----|
| jarray                              | 73  |
| jfloat                              | 74  |
| jfloat_array                        | 75  |
| join.tbl_spark                      | 75  |
| j_invoke                            | 77  |
| left_join                           | 78  |
| list_sparklyr_jars                  | 78  |
| livy_config                         | 78  |
| livy_service_start                  | 80  |
| ml-params                           | 80  |
| ml-persistence                      | 81  |
| ml-transform-methods                | 82  |
| ml-tuning                           | 83  |
| ml_aft_survival_regression          | 85  |
| ml_als                              | 88  |
| ml_als_tidiers                      | 91  |
| ml_bisecting_kmeans                 | 91  |
| ml_chisquare_test                   | 93  |
| ml_clustering_evaluator             | 94  |
| ml_corr                             | 95  |
| ml_decision_tree_classifier         | 96  |
| ml_default_stop_words               | 100 |
| ml_evaluate                         | 101 |
| ml_evaluator                        | 102 |
| ml_feature_importances              | 104 |
| ml_fpgrowth                         | 105 |
| ml_gaussian_mixture                 | 106 |
| ml_gbt_classifier                   | 107 |
| ml_generalized_linear_regression    | 112 |
| ml_glm_tidiers                      | 114 |
| ml_isotonic_regression              | 116 |
| ml_isotonic_regression_tidiers      | 117 |
| ml_kmeans                           | 118 |
| ml_kmeans_cluster_eval              | 120 |
| ml_lda                              | 120 |
| ml_lda_tidiers                      | 124 |
| ml_linear_regression                | 125 |
| ml_linear_svc                       | 127 |
| ml_linear_svc_tidiers               | 129 |
| ml_logistic_regression              | 129 |
| ml_logistic_regression_tidiers      | 132 |
| ml_metrics_binary                   | 133 |
| ml_metrics_multiclass               | 134 |
| ml_metrics_regression               | 135 |
| ml_model_data                       | 136 |
| ml_multilayer_perceptron_classifier | 137 |
| ml_multilayer_perceptron_tidiers    | 140 |
| ml_naive_bayes                      | 140 |

|  |     |
|--|-----|
| ml_naive_bayes_tidiers . . . . .         | 142 |
| ml_one_vs_rest . . . . .                 | 143 |
| ml_pca_tidiers . . . . .                 | 144 |
| ml_pipeline . . . . .                    | 145 |
| ml_power_iteration . . . . .             | 145 |
| ml_prefixspan . . . . .                  | 148 |
| ml_random_forest_classifier . . . . .    | 149 |
| ml_stage . . . . .                       | 153 |
| ml_summary . . . . .                     | 154 |
| ml_survival_regression_tidiers . . . . . | 154 |
| ml_tree_tidiers . . . . .                | 155 |
| ml_uid . . . . .                         | 157 |
| ml_unsupervised_tidiers . . . . .        | 157 |
| mutate . . . . .                         | 158 |
| na.replace . . . . .                     | 158 |
| nest . . . . .                           | 158 |
| pivot_longer . . . . .                   | 159 |
| pivot_wider . . . . .                    | 159 |
| random_string . . . . .                  | 159 |
| reactiveSpark . . . . .                  | 160 |
| registerDoSpark . . . . .                | 160 |
| register_extension . . . . .             | 161 |
| replace_na . . . . .                     | 161 |
| right_join . . . . .                     | 161 |
| sdf-saveload . . . . .                   | 162 |
| sdf-transform-methods . . . . .          | 162 |
| sdf_along . . . . .                      | 163 |
| sdf_bind . . . . .                       | 163 |
| sdf_broadcast . . . . .                  | 164 |
| sdf_checkpoint . . . . .                 | 165 |
| sdf_coalesce . . . . .                   | 165 |
| sdf_collect . . . . .                    | 166 |
| sdf_copy_to . . . . .                    | 166 |
| sdf_crosstab . . . . .                   | 167 |
| sdf_debug_string . . . . .               | 168 |
| sdf_describe . . . . .                   | 168 |
| sdf_dim . . . . .                        | 169 |
| sdf_distinct . . . . .                   | 169 |
| sdf_drop_duplicates . . . . .            | 170 |
| sdf_expand_grid . . . . .                | 170 |
| sdf_from_avro . . . . .                  | 171 |
| sdf_is_streaming . . . . .               | 171 |
| sdf_last_index . . . . .                 | 172 |
| sdf_len . . . . .                        | 172 |
| sdf_num_partitions . . . . .             | 173 |
| sdf_partition_sizes . . . . .            | 173 |
| sdf_persist . . . . .                    | 174 |
| sdf_pivot . . . . .                      | 174 |

|  |     |
|--|-----|
| sdf_project  | 175 |
| sdf_quantile   | 176 |
| sdf_random_split                                     | 177 |
| sdf_rbeta  | 178 |
| sdf_rbinom   | 179 |
| sdf_rcauchy  | 180 |
| sdf_rchisq   | 181 |
| sdf_read_column                                      | 181 |
| sdf_register   | 182 |
| sdf_repartition                                      | 182 |
| sdf_residuals.ml_model_generalized_linear_regression | 183 |
| sdf_rexp   | 183 |
| sdf_rgamma   | 184 |
| sdf_rgeom  | 185 |
| sdf_rhyper   | 186 |
| sdf_rlnorm   | 187 |
| sdf_rnorm  | 188 |
| sdf_rpois  | 189 |
| sdf_rt   | 189 |
| sdf_runif  | 190 |
| sdf_rweibull   | 191 |
| sdf_sample   | 192 |
| sdf_schema   | 192 |
| sdf_separate_column                                  | 193 |
| sdf_seq  | 193 |
| sdf_sort   | 194 |
| sdf_sql  | 194 |
| sdf_to_avro  | 195 |
| sdf_unnest_longer                                    | 195 |
| sdf_unnest_wider                                     | 196 |
| sdf_weighted_sample                                  | 198 |
| sdf_with_sequential_id                               | 198 |
| sdf_with_unique_id                                   | 199 |
| select   | 199 |
| separate   | 199 |
| spark-api  | 200 |
| spark-connections                                    | 201 |
| sparklyr_get_backend_port                            | 203 |
| spark_adaptive_query_execution                       | 203 |
| spark_advisory_shuffle_partition_size                | 204 |
| spark_apply  | 204 |
| spark_apply_bundle                                   | 207 |
| spark_apply_log                                      | 207 |
| spark_auto_broadcast_join_threshold                  | 208 |
| spark_coalesce_initial_num_partitions                | 208 |
| spark_coalesce_min_num_partitions                    | 209 |
| spark_coalesce_shuffle_partitions                    | 209 |
| spark_compilation_spec                               | 210 |



|  |     |
|--|-----|
| spark_config . . . . .                   | 211 |
| spark_config_kubernetes . . . . .        | 212 |
| spark_config_settings . . . . .          | 213 |
| spark_connection . . . . .               | 213 |
| spark_connection-class . . . . .         | 213 |
| spark_connection_find . . . . .          | 214 |
| spark_connect_method . . . . .           | 214 |
| spark_context_config . . . . .           | 215 |
| spark_dataframe . . . . .                | 216 |
| spark_default_compilation_spec . . . . . | 216 |
| spark_dependency . . . . .               | 217 |
| spark_dependency_fallback . . . . .      | 218 |
| spark_extension . . . . .                | 218 |
| spark_home_set . . . . .                 | 219 |
| spark_ide_connection_open . . . . .      | 219 |
| spark_insert_table . . . . .             | 221 |
| spark_install . . . . .                  | 222 |
| spark_integ_test_skip . . . . .          | 223 |
| spark_jobj . . . . .                     | 223 |
| spark_jobj-class . . . . .               | 224 |
| spark_last_error . . . . .               | 224 |
| spark_load_table . . . . .               | 224 |
| spark_log . . . . .                      | 225 |
| spark_read . . . . .                     | 226 |
| spark_read_avro . . . . .                | 227 |
| spark_read_binary . . . . .              | 228 |
| spark_read_csv . . . . .                 | 229 |
| spark_read_delta . . . . .               | 231 |
| spark_read_image . . . . .               | 232 |
| spark_read_jdbc . . . . .                | 233 |
| spark_read_json . . . . .                | 234 |
| spark_read_libsvm . . . . .              | 235 |
| spark_read_orc . . . . .                 | 236 |
| spark_read_parquet . . . . .             | 238 |
| spark_read_source . . . . .              | 239 |
| spark_read_table . . . . .               | 240 |
| spark_read_text . . . . .                | 241 |
| spark_save_table . . . . .               | 242 |
| spark_session_config . . . . .           | 243 |
| spark_statistical_routines . . . . .     | 243 |
| spark_table_name . . . . .               | 244 |
| spark_version . . . . .                  | 244 |
| spark_version_from_home . . . . .        | 245 |
| spark_web . . . . .                      | 245 |
| spark_write . . . . .                    | 246 |
| spark_write_avro . . . . .               | 247 |
| spark_write_csv . . . . .                | 248 |
| spark_write_delta . . . . .              | 249 |

|                           |     |
|---------------------------|-----|
| spark_write_jdbc          | 250 |
| spark_write_json          | 251 |
| spark_write_orc           | 252 |
| spark_write_parquet       | 253 |
| spark_write_rds           | 254 |
| spark_write_source        | 255 |
| spark_write_table         | 256 |
| spark_write_text          | 257 |
| src_databases             | 258 |
| stream_find               | 258 |
| stream_generate_test      | 259 |
| stream_id                 | 259 |
| stream_lag                | 260 |
| stream_name               | 261 |
| stream_read_csv           | 261 |
| stream_render             | 263 |
| stream_stats              | 264 |
| stream_stop               | 265 |
| stream_trigger_continuous | 265 |
| stream_trigger_interval   | 266 |
| stream_view               | 266 |
| stream_watermark          | 267 |
| stream_write_csv          | 267 |
| stream_write_memory       | 270 |
| tbl_cache                 | 271 |
| tbl_change_db             | 272 |
| tbl_uncache               | 272 |
| transform_sdf             | 273 |
| unite                     | 273 |
| unnest                    | 273 |
| [.tbl_spark               | 274 |
| %->%                      | 274 |

**Index****276**


---

checkpoint\_directory *Set/Get Spark checkpoint directory*

---

**Description**

Set/Get Spark checkpoint directory

**Usage**

spark\_set\_checkpoint\_dir(sc, dir)

spark\_get\_checkpoint\_dir(sc)

**Arguments**

sc                    A spark\_connection.  
 dir                   checkpoint directory, must be HDFS path of running on cluster

---

collect\_from\_rds        *Collect Spark data serialized in RDS format into R*

---

**Description**

Deserialize Spark data that is serialized using ‘spark\_write\_rds()’ into a R dataframe.

**Usage**

```
collect_from_rds(path)
```

**Arguments**

path                   Path to a local RDS file that is produced by ‘spark\_write\_rds()’ (RDS files stored in HDFS will need to be downloaded to local filesystem first (e.g., by running ‘hadoop fs -copyToLocal ...’ or similar)

**See Also**

Other Spark serialization routines: [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

compile\_package\_jars    *Compile Scala sources into a Java Archive (jar)*

---

**Description**

Compile the scala source files contained within an R package into a Java Archive (jar) file that can be loaded and used within a Spark environment.

**Usage**

```
compile_package_jars(..., spec = NULL)
```

**Arguments**

- ... Optional compilation specifications, as generated by `spark_compilation_spec`. When no arguments are passed, `spark_default_compilation_spec` is used instead.
- spec An optional list of compilation specifications. When set, this option takes precedence over arguments passed to ...

---

`connection_config`      *Read configuration values for a connection*

---

**Description**

Read configuration values for a connection

**Usage**

```
connection_config(sc, prefix, not_prefix = list())
```

**Arguments**

- sc                      spark\_connection
- prefix                 Prefix to read parameters for (e.g. `spark.context.`, `spark.sql.`, etc.)
- not\_prefix             Prefix to not include.

**Value**

Named list of config parameters (note that if a prefix was specified then the names will not include the prefix)

---

`copy_to.spark_connection`  
*Copy an R Data Frame to Spark*

---

**Description**

Copy an R `data.frame` to Spark, and return a reference to the generated Spark DataFrame as a `tbl_spark`. The returned object will act as a dplyr-compatible interface to the underlying Spark table.

**Usage**

```
## S3 method for class 'spark_connection'
copy_to(
  dest,
  df,
  name = spark_table_name(substitute(df)),
  overwrite = FALSE,
  memory = TRUE,
  repartition = 0L,
  ...
)
```

**Arguments**

|                          |  |
|--------------------------|--|
| <code>dest</code>        | A <code>spark_connection</code> .  |
| <code>df</code>          | An R data.frame.   |
| <code>name</code>        | The name to assign to the copied table in Spark.   |
| <code>overwrite</code>   | Boolean; overwrite a pre-existing table with the name <code>name</code> if one already exists?   |
| <code>memory</code>      | Boolean; should the table be cached into memory?   |
| <code>repartition</code> | The number of partitions to use when distributing the table across the Spark cluster. The default (0) can be used to avoid partitioning. |
| <code>...</code>         | Optional arguments; currently unused.  |

**Value**

A `tbl_spark`, representing a dplyr-compatible interface to a Spark DataFrame.

---

distinct

*Distinct*

---

**Description**

See [distinct](#) for more details.

---

|                 |  |
|-----------------|--|
| download_scalac | <i>Downloads default Scala Compilers</i> |
|-----------------|--|

---

### Description

compile\_package\_jars requires several versions of the scala compiler to work, this is to match Spark scala versions. To help setup your environment, this function will download the required compilers under the default search path.

### Usage

```
download_scalac(dest_path = NULL)
```

### Arguments

|           |  |
|-----------|--|
| dest_path | The destination path where scalac will be downloaded to. |
|-----------|--|

### Details

See find\_scalac for a list of paths searched and used by this function to install the required compilers.

---

|           |   |
|-----------|---|
| dplyr_hof | <i>dplyr wrappers for Apache Spark higher order functions</i> |
|-----------|---|

---

### Description

These methods implement dplyr grammars for Apache Spark higher order functions

---

|        |   |
|--------|---|
| ensure | <i>Enforce Specific Structure for R Objects</i> |
|--------|---|

---

### Description

These routines are useful when preparing to pass objects to a Spark routine, as it is often necessary to ensure certain parameters are scalar integers, or scalar doubles, and so on.

### Arguments

|            |   |
|------------|---|
| object     | An R object.  |
| allow.na   | Are NA values permitted for this object?  |
| allow.null | Are NULL values permitted for this object?  |
| default    | If object is NULL, what value should be used in its place? If default is specified, allow.null is ignored (and assumed to be TRUE). |

---

|      |             |
|------|-------------|
| fill | <i>Fill</i> |
|------|-------------|

---

**Description**

See [fill](#) for more details.

---

|        |               |
|--------|---------------|
| filter | <i>Filter</i> |
|--------|---------------|

---

**Description**

See [filter](#) for more details.

---

|             |                                    |
|-------------|------------------------------------|
| find_scalac | <i>Discover the Scala Compiler</i> |
|-------------|------------------------------------|

---

**Description**

Find the scalac compiler for a particular version of scala, by scanning some common directories containing scala installations.

**Usage**

```
find_scalac(version, locations = NULL)
```

**Arguments**

|           |  |
|-----------|--|
| version   | The scala version to search for. Versions of the form major.minor will be matched against the scalac installation with version major.minor.patch; if multiple compilers are discovered the most recent one will be used. |
| locations | Additional locations to scan. By default, the directories /opt/scala and /usr/local/scala will be scanned.   |

ft\_binarizer

*Feature Transformation – Binarizer (Transformer)***Description**

Apply thresholding to a column, such that values less than or equal to the threshold are assigned the value 0.0, and values greater than the threshold are assigned the value 1.0. Column output is numeric for compatibility with other modeling functions.

**Usage**

```
ft_binarizer(
  x,
  input_col,
  output_col,
  threshold = 0,
  uid = random_string("binarizer_"),
  ...
)
```

**Arguments**

|            |   |
|------------|---|
| x          | A spark_connection, ml_pipeline, or a tbl_spark.                      |
| input_col  | The name of the input column.   |
| output_col | The name of the output column.  |
| threshold  | Threshold used to binarize continuous features.                       |
| uid        | A character string used to uniquely identify the feature transformer. |
| ...        | Optional arguments; currently unused.                                 |

**Value**

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

**See Also**

Other feature transformers: [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remover\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)



**Examples**

```
## Not run:
library(dplyr)

sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

iris_tbl %>%
  ft_binarizer(
    input_col = "Sepal_Length",
    output_col = "Sepal_Length_bin",
    threshold = 5
  ) %>%
  select(Sepal_Length, Sepal_Length_bin, Species)

## End(Not run)
```

ft\_bucketizer

*Feature Transformation – Bucketizer (Transformer)***Description**

Similar to R's `cut` function, this transforms a numeric column into a discretized column, with breaks specified through the `splits` parameter.

**Usage**

```
ft_bucketizer(
  x,
  input_col = NULL,
  output_col = NULL,
  splits = NULL,
  input_cols = NULL,
  output_cols = NULL,
  splits_array = NULL,
  handle_invalid = "error",
  uid = random_string("bucketizer_"),
  ...
)
```

**Arguments**

|                         |  |
|-------------------------|--|
| <code>x</code>          | A <code>spark_connection</code> , <code>ml_pipeline</code> , or a <code>tbl_spark</code> . |
| <code>input_col</code>  | The name of the input column.  |
| <code>output_col</code> | The name of the output column.   |
| <code>splits</code>     | A numeric vector of cutpoints, indicating the bucket boundaries.                           |

|                |  |
|----------------|--|
| input_cols     | Names of input columns.  |
| output_cols    | Names of output columns.   |
| splits_array   | Parameter for specifying multiple splits parameters. Each element in this array can be used to map continuous features into buckets.   |
| handle_invalid | (Spark 2.1.0+) Param for how to handle invalid entries. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (keep invalid values in a special additional bucket). Default: "error" |
| uid            | A character string used to uniquely identify the feature transformer.  |
| ...            | Optional arguments; currently unused.  |

### Value

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

### See Also

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remover\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

### Examples

```
## Not run:
library(dplyr)

sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

iris_tbl %>%
  ft_bucketizer(
    input_col = "Sepal_Length",
    output_col = "Sepal_Length_bucket",
    splits = c(0, 4.5, 5, 8)
  ) %>%
  select(Sepal_Length, Sepal_Length_bucket, Species)

## End(Not run)
```

---

 ft\_chisq\_selector      *Feature Transformation – ChiSqSelector (Estimator)*


---

## Description

Chi-Squared feature selection, which selects categorical features to use for predicting a categorical label

## Usage

```
ft_chisq_selector(
  x,
  features_col = "features",
  output_col = NULL,
  label_col = "label",
  selector_type = "numTopFeatures",
  fdr = 0.05,
  fpr = 0.05,
  fwe = 0.05,
  num_top_features = 50,
  percentile = 0.1,
  uid = random_string("chisq_selector_"),
  ...
)
```

## Arguments

|               |  |
|---------------|--|
| x             | A spark_connection, ml_pipeline, or a tbl_spark.   |
| features_col  | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> . |
| output_col    | The name of the output column.   |
| label_col     | Label column name. The column should be a numeric column. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| selector_type | (Spark 2.1.0+) The selector type of the ChiSqSelector. Supported options: "num-TopFeatures" (default), "percentile", "fpr", "fdr", "fwe".  |
| fdr           | (Spark 2.2.0+) The upper bound of the expected false discovery rate. Only applicable when selector_type = "fdr". Default value is 0.05.  |
| fpr           | (Spark 2.1.0+) The highest p-value for features to be kept. Only applicable when selector_type = "fpr". Default value is 0.05.   |
| fwe           | (Spark 2.2.0+) The upper bound of the expected family-wise error rate. Only applicable when selector_type = "fwe". Default value is 0.05.  |

|                  |  |
|------------------|--|
| num_top_features | Number of features that selector will select, ordered by ascending p-value. If the number of features is less than num_top_features, then this will select all features. Only applicable when selector_type = "numTopFeatures". The default value of num_top_features is 50. |
| percentile       | (Spark 2.1.0+) Percentile of features that selector will select, ordered by statistics value descending. Only applicable when selector_type = "percentile". Default value is 0.1.  |
| uid              | A character string used to uniquely identify the feature transformer.  |
| ...              | Optional arguments; currently unused.  |

### Details

In the case where x is a tbl\_spark, the estimator fits against x to obtain a transformer, returning a tbl\_spark.

### Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

### See Also

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remover\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

---

ft\_count\_vectorizer      *Feature Transformation – CountVectorizer (Estimator)*

---

### Description

Extracts a vocabulary from document collections.

**Usage**

```
ft_count_vectorizer(
  x,
  input_col = NULL,
  output_col = NULL,
  binary = FALSE,
  min_df = 1,
  min_tf = 1,
  vocab_size = 2^18,
  uid = random_string("count_vectorizer_"),
  ...
)

ml_vocabulary(model)
```

**Arguments**

|            |  |
|------------|--|
| x          | A spark_connection, ml_pipeline, or a tbl_spark.   |
| input_col  | The name of the input column.  |
| output_col | The name of the output column.   |
| binary     | Binary toggle to control the output vector values. If TRUE, all nonzero counts (after min_tf filter applied) are set to 1. This is useful for discrete probabilistic models that model binary events rather than integer counts. Default: FALSE  |
| min_df     | Specifies the minimum number of different documents a term must appear in to be included in the vocabulary. If this is an integer greater than or equal to 1, this specifies the number of documents the term must appear in; if this is a double in [0,1), then this specifies the fraction of documents. Default: 1.   |
| min_tf     | Filter to ignore rare words in a document. For each document, terms with frequency/count less than the given threshold are ignored. If this is an integer greater than or equal to 1, then this specifies a count (of times the term must appear in the document); if this is a double in [0,1), then this specifies a fraction (out of the document's token count). Default: 1. |
| vocab_size | Build a vocabulary that only considers the top vocab_size terms ordered by term frequency across the corpus. Default: 2^18.  |
| uid        | A character string used to uniquely identify the feature transformer.  |
| ...        | Optional arguments; currently unused.  |
| model      | A ml_count_vectorizer_model.   |

**Details**

In the case where x is a tbl\_spark, the estimator fits against x to obtain a transformer, returning a tbl\_spark.

**Value**

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

`ml_vocabulary()` returns a vector of vocabulary built.

**See Also**

Other feature transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_robust_scaler()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

---

|        |   |
|--------|---|
| ft_dct | <i>Feature Transformation – Discrete Cosine Transform (DCT) (Transformer)</i> |
|--------|---|

---

**Description**

A feature transformer that takes the 1D discrete cosine transform of a real vector. No zero padding is performed on the input vector. It returns a real vector of the same length representing the DCT. The return vector is scaled such that the transform matrix is unitary (aka scaled DCT-II).

**Usage**

```
ft_dct(
  x,
  input_col = NULL,
  output_col = NULL,
  inverse = FALSE,
  uid = random_string("dct_"),
  ...
)

ft_discrete_cosine_transform(
  x,
  input_col,
  output_col,
  inverse = FALSE,
  uid = random_string("dct_"),
  ...
)
```

### Arguments

|            |   |
|------------|---|
| x          | A spark_connection, ml_pipeline, or a tbl_spark.                            |
| input_col  | The name of the input column.   |
| output_col | The name of the output column.  |
| inverse    | Indicates whether to perform the inverse DCT (TRUE) or forward DCT (FALSE). |
| uid        | A character string used to uniquely identify the feature transformer.       |
| ...        | Optional arguments; currently unused.                                       |

### Details

ft\_discrete\_cosine\_transform() is an alias for ft\_dct for backwards compatibility.

### Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

### See Also

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remover\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

---

ft\_elementwise\_product

*Feature Transformation – ElementwiseProduct (Transformer)*

---

### Description

Outputs the Hadamard product (i.e., the element-wise product) of each input vector with a provided "weight" vector. In other words, it scales each column of the dataset by a scalar multiplier.

**Usage**

```
ft_elementwise_product(
  x,
  input_col = NULL,
  output_col = NULL,
  scaling_vec = NULL,
  uid = random_string("elementwise_product_"),
  ...
)
```

**Arguments**

|             |   |
|-------------|---|
| x           | A spark_connection, ml_pipeline, or a tbl_spark.                      |
| input_col   | The name of the input column.   |
| output_col  | The name of the output column.  |
| scaling_vec | the vector to multiply with input vectors                             |
| uid         | A character string used to uniquely identify the feature transformer. |
| ...         | Optional arguments; currently unused.                                 |

**Value**

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

**See Also**

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remover\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

---

ft\_feature\_hasher      *Feature Transformation – FeatureHasher (Transformer)*

---

**Description**

Feature Transformation – FeatureHasher (Transformer)



**Usage**

```
ft_feature_hasher(
  x,
  input_cols = NULL,
  output_col = NULL,
  num_features = 2^18,
  categorical_cols = NULL,
  uid = random_string("feature_hasher_"),
  ...
)
```

**Arguments**

|                               |   |
|-------------------------------|---|
| <code>x</code>                | A <code>spark_connection</code> , <code>ml_pipeline</code> , or a <code>tbl_spark</code> .  |
| <code>input_cols</code>       | Names of input columns.   |
| <code>output_col</code>       | Name of output column.  |
| <code>num_features</code>     | Number of features. Defaults to $2^{18}$ .  |
| <code>categorical_cols</code> | Numeric columns to treat as categorical features. By default only string and boolean columns are treated as categorical, so this param can be used to explicitly specify the numerical columns to treat as categorical. |
| <code>uid</code>              | A character string used to uniquely identify the feature transformer.   |
| <code>...</code>              | Optional arguments; currently unused.   |

**Details**

Feature hashing projects a set of categorical or numerical features into a feature vector of specified dimension (typically substantially smaller than that of the original feature space). This is done using the hashing trick [https://en.wikipedia.org/wiki/Feature\\_hashing](https://en.wikipedia.org/wiki/Feature_hashing) to map features to indices in the feature vector.

The FeatureHasher transformer operates on multiple columns. Each column may contain either numeric or categorical features. Behavior and handling of column data types is as follows: -Numeric columns: For numeric features, the hash value of the column name is used to map the feature value to its index in the feature vector. By default, numeric features are not treated as categorical (even when they are integers). To treat them as categorical, specify the relevant columns in `categorical_cols`. -String columns: For categorical features, the hash value of the string "`column_name=value`" is used to map to the vector index, with an indicator value of 1.0. Thus, categorical features are "one-hot" encoded (similarly to using `OneHotEncoder` with `drop_last=FALSE`). -Boolean columns: Boolean values are treated in the same way as string columns. That is, boolean features are represented as "`column_name=true`" or "`column_name=false`", with an indicator value of 1.0.

Null (missing) values are ignored (implicitly zero in the resulting feature vector).

The hash function used here is also the MurmurHash 3 used in HashingTF. Since a simple modulo on the hashed value is used to determine the vector index, it is advisable to use a power of two as the `num_features` parameter; otherwise the features will not be mapped evenly to the vector indices.

**Value**

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

**See Also**

Other feature transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_robust_scaler()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

ft\_hashing\_tf

*Feature Transformation – HashingTF (Transformer)***Description**

Maps a sequence of terms to their term frequencies using the hashing trick.

**Usage**

```
ft_hashing_tf(
  x,
  input_col = NULL,
  output_col = NULL,
  binary = FALSE,
  num_features = 2^18,
  uid = random_string("hashing_tf_"),
  ...
)
```

**Arguments**

|                           |  |
|---------------------------|--|
| <code>x</code>            | A <code>spark_connection</code> , <code>ml_pipeline</code> , or a <code>tbl_spark</code> .   |
| <code>input_col</code>    | The name of the input column.  |
| <code>output_col</code>   | The name of the output column.   |
| <code>binary</code>       | Binary toggle to control term frequency counts. If true, all non-zero counts are set to 1. This is useful for discrete probabilistic models that model binary events rather than integer counts. (default = FALSE) |
| <code>num_features</code> | Number of features. Should be greater than 0. (default = 2 <sup>18</sup> )   |
| <code>uid</code>          | A character string used to uniquely identify the feature transformer.  |
| <code>...</code>          | Optional arguments; currently unused.  |

**Value**

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

**See Also**

Other feature transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_robust_scaler()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

ft\_idf

*Feature Transformation – IDF (Estimator)***Description**

Compute the Inverse Document Frequency (IDF) given a collection of documents.

**Usage**

```
ft_idf(
  x,
  input_col = NULL,
  output_col = NULL,
  min_doc_freq = 0,
  uid = random_string("idf_"),
  ...
)
```

**Arguments**

|                           |  |
|---------------------------|--|
| <code>x</code>            | A <code>spark_connection</code> , <code>ml_pipeline</code> , or a <code>tbl_spark</code> . |
| <code>input_col</code>    | The name of the input column.  |
| <code>output_col</code>   | The name of the output column.   |
| <code>min_doc_freq</code> | The minimum number of documents in which a term should appear. Default: 0                  |
| <code>uid</code>          | A character string used to uniquely identify the feature transformer.                      |
| <code>...</code>          | Optional arguments; currently unused.  |

**Details**

In the case where `x` is a `tbl_spark`, the estimator fits against `x` to obtain a transformer, returning a `tbl_spark`.

**Value**

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

**See Also**

Other feature transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_robust_scaler()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

---

 ft\_imputer

*Feature Transformation – Imputer (Estimator)*


---

**Description**

Imputation estimator for completing missing values, either using the mean or the median of the columns in which the missing values are located. The input columns should be of numeric type. This function requires Spark 2.2.0+.

**Usage**

```
ft_imputer(
  x,
  input_cols = NULL,
  output_cols = NULL,
  missing_value = NULL,
  strategy = "mean",
  uid = random_string("imputer_"),
  ...
)
```

**Arguments**

|                            |   |
|----------------------------|---|
| <code>x</code>             | A <code>spark_connection</code> , <code>ml_pipeline</code> , or a <code>tbl_spark</code> .  |
| <code>input_cols</code>    | The names of the input columns  |
| <code>output_cols</code>   | The names of the output columns.  |
| <code>missing_value</code> | The placeholder for the missing values. All occurrences of <code>missing_value</code> will be imputed. Note that null values are always treated as missing. |

|          |  |
|----------|--|
| strategy | The imputation strategy. Currently only "mean" and "median" are supported. If "mean", then replace missing values using the mean value of the feature. If "median", then replace missing values using the approximate median value of the feature. Default: mean |
| uid      | A character string used to uniquely identify the feature transformer.  |
| ...      | Optional arguments; currently unused.  |

### Details

In the case where `x` is a `tbl_spark`, the estimator fits against `x` to obtain a transformer, returning a `tbl_spark`.

### Value

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

### See Also

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_removal\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

---

ft\_index\_to\_string      *Feature Transformation – IndexToString (Transformer)*

---

### Description

A Transformer that maps a column of indices back to a new column of corresponding string values. The index-string mapping is either from the ML attributes of the input column, or from user-supplied labels (which take precedence over ML attributes). This function is the inverse of [ft\\_string\\_indexer](#).

### Usage

```
ft_index_to_string(
  x,
  input_col = NULL,
  output_col = NULL,
  labels = NULL,
```

```

    uid = random_string("index_to_string"),
    ...
)

```

### Arguments

|            |   |
|------------|---|
| x          | A spark_connection, ml_pipeline, or a tbl_spark.                      |
| input_col  | The name of the input column.   |
| output_col | The name of the output column.  |
| labels     | Optional param for array of labels specifying index-string mapping.   |
| uid        | A character string used to uniquely identify the feature transformer. |
| ...        | Optional arguments; currently unused.                                 |

### Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

### See Also

[ft\\_string\\_indexer](#)

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remover\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

---

ft\_interaction

*Feature Transformation – Interaction (Transformer)*

---

### Description

Implements the feature interaction transform. This transformer takes in Double and Vector type columns and outputs a flattened vector of their feature interactions. To handle interaction, we first one-hot encode any nominal features. Then, a vector of the feature cross-products is produced.

**Usage**

```
ft_interaction(
  x,
  input_cols = NULL,
  output_col = NULL,
  uid = random_string("interaction_"),
  ...
)
```

**Arguments**

|            |   |
|------------|---|
| x          | A spark_connection, ml_pipeline, or a tbl_spark.                      |
| input_cols | The names of the input columns  |
| output_col | The name of the output column.  |
| uid        | A character string used to uniquely identify the feature transformer. |
| ...        | Optional arguments; currently unused.                                 |

**Value**

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

**See Also**

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remover\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

---

ft\_lsh

*Feature Transformation – LSH (Estimator)*


---

**Description**

Locality Sensitive Hashing functions for Euclidean distance (Bucketed Random Projection) and Jaccard distance (MinHash).

**Usage**

```

ft_bucketed_random_projection_lsh(
  x,
  input_col = NULL,
  output_col = NULL,
  bucket_length = NULL,
  num_hash_tables = 1,
  seed = NULL,
  uid = random_string("bucketed_random_projection_lsh_"),
  ...
)

ft_minhash_lsh(
  x,
  input_col = NULL,
  output_col = NULL,
  num_hash_tables = 1L,
  seed = NULL,
  uid = random_string("minhash_lsh_"),
  ...
)

```

**Arguments**

|                 |   |
|-----------------|---|
| x               | A spark_connection, ml_pipeline, or a tbl_spark.  |
| input_col       | The name of the input column.   |
| output_col      | The name of the output column.  |
| bucket_length   | The length of each hash bucket, a larger bucket lowers the false negative rate. The number of buckets will be (max L2 norm of input vectors) / bucketLength.  |
| num_hash_tables | Number of hash tables used in LSH OR-amplification. LSH OR-amplification can be used to reduce the false negative rate. Higher values for this param lead to a reduced false negative rate, at the expense of added computational complexity. |
| seed            | A random seed. Set this value if you need your results to be reproducible across repeated calls.  |
| uid             | A character string used to uniquely identify the feature transformer.   |
| ...             | Optional arguments; currently unused.   |

**Details**

In the case where x is a tbl\_spark, the estimator fits against x to obtain a transformer, returning a tbl\_spark.

**Value**

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with



the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

### See Also

`ft_lsh_utils`

Other feature transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_robust_scaler()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

---

ft\_lsh\_utils

*Utility functions for LSH models*

---

### Description

Utility functions for LSH models

### Usage

```
ml_approx_nearest_neighbors(
  model,
  dataset,
  key,
  num_nearest_neighbors,
  dist_col = "distCol"
)
```

```
ml_approx_similarity_join(
  model,
  dataset_a,
  dataset_b,
  threshold,
  dist_col = "distCol"
)
```

### Arguments

|                                    |  |
|------------------------------------|--|
| <code>model</code>                 | A fitted LSH model, returned by either <code>ft_minhash_lsh()</code> or <code>ft_bucketed_random_projection_lsh()</code> |
| <code>dataset</code>               | The dataset to search for nearest neighbors of the key.  |
| <code>key</code>                   | Feature vector representing the item to search for.  |
| <code>num_nearest_neighbors</code> | The maximum number of nearest neighbors.   |

|           |   |
|-----------|---|
| dist_col  | Output column for storing the distance between each result row and the key. |
| dataset_a | One of the datasets to join.  |
| dataset_b | Another dataset to join.  |
| threshold | The threshold for the distance of row pairs.                                |

---

ft\_max\_abs\_scaler      *Feature Transformation – MaxAbsScaler (Estimator)*

---

### Description

Rescale each feature individually to range [-1, 1] by dividing through the largest maximum absolute value in each feature. It does not shift/center the data, and thus does not destroy any sparsity.

### Usage

```
ft_max_abs_scaler(
  x,
  input_col = NULL,
  output_col = NULL,
  uid = random_string("max_abs_scaler_"),
  ...
)
```

### Arguments

|            |   |
|------------|---|
| x          | A spark_connection, ml_pipeline, or a tbl_spark.                      |
| input_col  | The name of the input column.   |
| output_col | The name of the output column.  |
| uid        | A character string used to uniquely identify the feature transformer. |
| ...        | Optional arguments; currently unused.                                 |

### Details

In the case where x is a tbl\_spark, the estimator fits against x to obtain a transformer, returning a tbl\_spark.

### Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

**See Also**

Other feature transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_robust_scaler()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_removal()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

**Examples**

```
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

features <- c("Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width")

iris_tbl %>%
  ft_vector_assembler(
    input_col = features,
    output_col = "features_temp"
  ) %>%
  ft_max_abs_scaler(
    input_col = "features_temp",
    output_col = "features"
  )

## End(Not run)
```

---

ft\_min\_max\_scaler      *Feature Transformation – MinMaxScaler (Estimator)*

---

**Description**

Rescale each feature individually to a common range [min, max] linearly using column summary statistics, which is also known as min-max normalization or Rescaling

**Usage**

```
ft_min_max_scaler(
  x,
  input_col = NULL,
  output_col = NULL,
  min = 0,
  max = 1,
  uid = random_string("min_max_scaler_"),
```

```
    ...
  )
```

### Arguments

|            |   |
|------------|---|
| x          | A spark_connection, ml_pipeline, or a tbl_spark.                      |
| input_col  | The name of the input column.   |
| output_col | The name of the output column.  |
| min        | Lower bound after transformation, shared by all features Default: 0.0 |
| max        | Upper bound after transformation, shared by all features Default: 1.0 |
| uid        | A character string used to uniquely identify the feature transformer. |
| ...        | Optional arguments; currently unused.                                 |

### Details

In the case where x is a tbl\_spark, the estimator fits against x to obtain a transformer, returning a tbl\_spark.

### Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

### See Also

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remove\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

### Examples

```
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

features <- c("Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width")

iris_tbl %>%
  ft_vector_assembler(
    input_col = features,
    output_col = "features_temp"
  ) %>%
```

```

ft_min_max_scaler(
  input_col = "features_temp",
  output_col = "features"
)

## End(Not run)

```

---

ft\_ngram

*Feature Transformation – NGram (Transformer)*


---

### Description

A feature transformer that converts the input array of strings into an array of n-grams. Null values in the input array are ignored. It returns an array of n-grams where each n-gram is represented by a space-separated string of words.

### Usage

```

ft_ngram(
  x,
  input_col = NULL,
  output_col = NULL,
  n = 2,
  uid = random_string("ngram_"),
  ...
)

```

### Arguments

|            |  |
|------------|--|
| x          | A spark_connection, ml_pipeline, or a tbl_spark.                               |
| input_col  | The name of the input column.  |
| output_col | The name of the output column.   |
| n          | Minimum n-gram length, greater than or equal to 1. Default: 2, bigram features |
| uid        | A character string used to uniquely identify the feature transformer.          |
| ...        | Optional arguments; currently unused.  |

### Details

When the input is empty, an empty array is returned. When the input array length is less than n (number of elements per n-gram), no n-grams are returned.

### Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

**See Also**

Other feature transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_robust_scaler()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

ft\_normalizer

*Feature Transformation – Normalizer (Transformer)***Description**

Normalize a vector to have unit norm using the given p-norm.

**Usage**

```
ft_normalizer(
  x,
  input_col = NULL,
  output_col = NULL,
  p = 2,
  uid = random_string("normalizer_"),
  ...
)
```

**Arguments**

|                         |  |
|-------------------------|--|
| <code>x</code>          | A <code>spark_connection</code> , <code>ml_pipeline</code> , or a <code>tbl_spark</code> . |
| <code>input_col</code>  | The name of the input column.  |
| <code>output_col</code> | The name of the output column.   |
| <code>p</code>          | Normalization in $L^p$ space. Must be $\geq 1$ . Defaults to 2.                            |
| <code>uid</code>        | A character string used to uniquely identify the feature transformer.                      |
| <code>...</code>        | Optional arguments; currently unused.  |

**Value**

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

**See Also**

Other feature transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_robust_scaler()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

---

ft\_one\_hot\_encoder      *Feature Transformation – OneHotEncoder (Transformer)*

---

**Description**

One-hot encoding maps a column of label indices to a column of binary vectors, with at most a single one-value. This encoding allows algorithms which expect continuous features, such as Logistic Regression, to use categorical features. Typically, used with `ft_string_indexer()` to index a column first.

**Usage**

```
ft_one_hot_encoder(
  x,
  input_cols = NULL,
  output_cols = NULL,
  handle_invalid = NULL,
  drop_last = TRUE,
  uid = random_string("one_hot_encoder_"),
  ...
)
```

**Arguments**

|                             |  |
|-----------------------------|--|
| <code>x</code>              | A <code>spark_connection</code> , <code>ml_pipeline</code> , or a <code>tbl_spark</code> .   |
| <code>input_cols</code>     | The name of the input columns.   |
| <code>output_cols</code>    | The name of the output columns.  |
| <code>handle_invalid</code> | (Spark 2.1.0+) Param for how to handle invalid entries. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (keep invalid values in a special additional bucket). Default: "error" |
| <code>drop_last</code>      | Whether to drop the last category. Defaults to TRUE.   |
| <code>uid</code>            | A character string used to uniquely identify the feature transformer.  |
| <code>...</code>            | Optional arguments; currently unused.  |

**Value**

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

**See Also**

Other feature transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_robust_scaler()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

---

`ft_one_hot_encoder_estimator`

*Feature Transformation – OneHotEncoderEstimator (Estimator)*

---

**Description**

A one-hot encoder that maps a column of category indices to a column of binary vectors, with at most a single one-value per row that indicates the input category index. For example with 5 categories, an input value of 2.0 would map to an output vector of [0.0, 0.0, 1.0, 0.0]. The last category is not included by default (configurable via `dropLast`), because it makes the vector entries sum up to one, and hence linearly dependent. So an input value of 4.0 maps to [0.0, 0.0, 0.0, 0.0].

**Usage**

```
ft_one_hot_encoder_estimator(
  x,
  input_cols = NULL,
  output_cols = NULL,
  handle_invalid = "error",
  drop_last = TRUE,
  uid = random_string("one_hot_encoder_estimator_"),
  ...
)
```

**Arguments**

|                          |  |
|--------------------------|--|
| <code>x</code>           | A <code>spark_connection</code> , <code>ml_pipeline</code> , or a <code>tbl_spark</code> . |
| <code>input_cols</code>  | Names of input columns.  |
| <code>output_cols</code> | Names of output columns.   |



|                |  |
|----------------|--|
| handle_invalid | (Spark 2.1.0+) Param for how to handle invalid entries. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (keep invalid values in a special additional bucket). Default: "error" |
| drop_last      | Whether to drop the last category. Defaults to TRUE.   |
| uid            | A character string used to uniquely identify the feature transformer.  |
| ...            | Optional arguments; currently unused.  |

### Details

In the case where `x` is a `tbl_spark`, the estimator fits against `x` to obtain a transformer, returning a `tbl_spark`.

### Value

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

### See Also

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_removal\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

---

ft\_pca

*Feature Transformation – PCA (Estimator)*

---

### Description

PCA trains a model to project vectors to a lower dimensional space of the top `k` principal components.

### Usage

```
ft_pca(
  x,
  input_col = NULL,
  output_col = NULL,
  k = NULL,
  uid = random_string("pca_"),
  ...
)
```

```
ml_pca(x, features = tbl_vars(x), k = length(features), pc_prefix = "PC", ...)
```

**Arguments**

|            |  |
|------------|--|
| x          | A spark_connection, ml_pipeline, or a tbl_spark.                                       |
| input_col  | The name of the input column.  |
| output_col | The name of the output column.   |
| k          | The number of principal components   |
| uid        | A character string used to uniquely identify the feature transformer.                  |
| ...        | Optional arguments; currently unused.  |
| features   | The columns to use in the principal components analysis. Defaults to all columns in x. |
| pc_prefix  | Length-one character vector used to prepend names of components.                       |

**Details**

In the case where x is a tbl\_spark, the estimator fits against x to obtain a transformer, returning a tbl\_spark.

ml\_pca() is a wrapper around ft\_pca() that returns a ml\_model.

**Value**

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

**See Also**

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remover\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

**Examples**

```
## Not run:
library(dplyr)

sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

iris_tbl %>%
  select(-Species) %>%
  ml_pca(k = 2)
```

```
## End(Not run)
```

---

```
ft_polynomial_expansion
```

*Feature Transformation – PolynomialExpansion (Transformer)*

---

## Description

Perform feature expansion in a polynomial space. E.g. take a 2-variable feature vector as an example: (x, y), if we want to expand it with degree 2, then we get (x, x \* x, y, x \* y, y \* y).

## Usage

```
ft_polynomial_expansion(
  x,
  input_col = NULL,
  output_col = NULL,
  degree = 2,
  uid = random_string("polynomial_expansion_"),
  ...
)
```

## Arguments

|            |   |
|------------|---|
| x          | A spark_connection, ml_pipeline, or a tbl_spark.  |
| input_col  | The name of the input column.   |
| output_col | The name of the output column.  |
| degree     | The polynomial degree to expand, which should be greater than equal to 1. A value of 1 means no expansion. Default: 2 |
| uid        | A character string used to uniquely identify the feature transformer.   |
| ...        | Optional arguments; currently unused.   |

## Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

**See Also**

Other feature transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_robust_scaler()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

---

ft\_quantile\_discretizer

*Feature Transformation – QuantileDiscretizer (Estimator)*

---

**Description**

`ft_quantile_discretizer` takes a column with continuous features and outputs a column with binned categorical features. The number of bins can be set using the `num_buckets` parameter. It is possible that the number of buckets used will be smaller than this value, for example, if there are too few distinct values of the input to create enough distinct quantiles.

**Usage**

```
ft_quantile_discretizer(
  x,
  input_col = NULL,
  output_col = NULL,
  num_buckets = 2,
  input_cols = NULL,
  output_cols = NULL,
  num_buckets_array = NULL,
  handle_invalid = "error",
  relative_error = 0.001,
  uid = random_string("quantile_discretizer_"),
  weight_column = NULL,
  ...
)
```

**Arguments**

|                          |  |
|--------------------------|--|
| <code>x</code>           | A <code>spark_connection</code> , <code>ml_pipeline</code> , or a <code>tbl_spark</code> .                           |
| <code>input_col</code>   | The name of the input column.  |
| <code>output_col</code>  | The name of the output column.   |
| <code>num_buckets</code> | Number of buckets (quantiles, or categories) into which data points are grouped. Must be greater than or equal to 2. |
| <code>input_cols</code>  | Names of input columns.  |

|                   |  |
|-------------------|--|
| output_cols       | Names of output columns.   |
| num_buckets_array | Array of number of buckets (quantiles, or categories) into which data points are grouped. Each value must be greater than or equal to 2.   |
| handle_invalid    | (Spark 2.1.0+) Param for how to handle invalid entries. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (keep invalid values in a special additional bucket). Default: "error"   |
| relative_error    | (Spark 2.0.0+) Relative error (see documentation for <code>org.apache.spark.sql.DataFrameStatFunctions.approxQuantile</code> <a href="#">here</a> for description). Must be in the range [0, 1]. default: 0.001  |
| uid               | A character string used to uniquely identify the feature transformer.  |
| weight_column     | If not NULL, then a generalized version of the Greenwald-Khanna algorithm will be run to compute weighted percentiles, with each input having a relative weight specified by the corresponding value in 'weight_column'. The weights can be considered as relative frequencies of sample inputs. |
| ...               | Optional arguments; currently unused.  |

### Details

NaN handling: null and NaN values will be ignored from the column during `QuantileDiscretizer` fitting. This will produce a `Bucketizer` model for making predictions. During the transformation, `Bucketizer` will raise an error when it finds NaN values in the dataset, but the user can also choose to either keep or remove NaN values within the dataset by setting `handle_invalid`. If the user chooses to keep NaN values, they will be handled specially and placed into their own bucket, for example, if 4 buckets are used, then non-NaN data will be put into `buckets[0-3]`, but NaNs will be counted in a special bucket[4].

Algorithm: The bin ranges are chosen using an approximate algorithm (see the documentation for `org.apache.spark.sql.DataFrameStatFunctions.approxQuantile` [here](#) for a detailed description). The precision of the approximation can be controlled with the `relative_error` parameter. The lower and upper bin bounds will be `-Infinity` and `+Infinity`, covering all real values.

Note that the result may be different every time you run it, since the sample strategy behind it is non-deterministic.

In the case where `x` is a `tbl_spark`, the estimator fits against `x` to obtain a transformer, returning a `tbl_spark`.

### Value

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

### See Also

[ft\\_bucketizer](#)

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#),

```
ft_imputer(), ft_index_to_string(), ft_interaction(), ft_lsh, ft_max_abs_scaler(),
ft_min_max_scaler(), ft_ngram(), ft_normalizer(), ft_one_hot_encoder_estimator(), ft_one_hot_encoder(),
ft_pca(), ft_polynomial_expansion(), ft_r_formula(), ft_regex_tokenizer(), ft_robust_scaler(),
ft_sql_transformer(), ft_standard_scaler(), ft_stop_words_remover(), ft_string_indexer(),
ft_tokenizer(), ft_vector_assembler(), ft_vector_indexer(), ft_vector_slicer(), ft_word2vec()
```

---

ft\_regex\_tokenizer      *Feature Transformation – RegexTokenizer (Transformer)*

---

### Description

A regex based tokenizer that extracts tokens either by using the provided regex pattern to split the text (default) or repeatedly matching the regex (if gaps is false). Optional parameters also allow filtering tokens using a minimal length. It returns an array of strings that can be empty.

### Usage

```
ft_regex_tokenizer(
  x,
  input_col = NULL,
  output_col = NULL,
  gaps = TRUE,
  min_token_length = 1,
  pattern = "\\s+",
  to_lower_case = TRUE,
  uid = random_string("regex_tokenizer_"),
  ...
)
```

### Arguments

|                  |   |
|------------------|---|
| x                | A spark_connection, ml_pipeline, or a tbl_spark.                            |
| input_col        | The name of the input column.   |
| output_col       | The name of the output column.  |
| gaps             | Indicates whether regex splits on gaps (TRUE) or matches tokens (FALSE).    |
| min_token_length | Minimum token length, greater than or equal to 0.                           |
| pattern          | The regular expression pattern to be used.                                  |
| to_lower_case    | Indicates whether to convert all characters to lowercase before tokenizing. |
| uid              | A character string used to uniquely identify the feature transformer.       |
| ...              | Optional arguments; currently unused.                                       |

**Value**

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

**See Also**

Other feature transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_robust_scaler()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_vector_slicer()`, `ft_word2vec()`

---

 ft\_robust\_scaler

*Feature Transformation – RobustScaler (Estimator)*


---

**Description**

`RobustScaler` removes the median and scales the data according to the quantile range. The quantile range is by default IQR (Interquartile Range, quantile range between the 1st quartile = 25th quantile and the 3rd quartile = 75th quantile) but can be configured. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Median and quantile range are then stored to be used on later data using the transform method. Note that missing values are ignored in the computation of medians and ranges.

**Usage**

```
ft_robust_scaler(
  x,
  input_col = NULL,
  output_col = NULL,
  lower = 0.25,
  upper = 0.75,
  with_centering = TRUE,
  with_scaling = TRUE,
  relative_error = 0.001,
  uid = random_string("ft_robust_scaler_"),
  ...
)
```

**Arguments**

|                |   |
|----------------|---|
| x              | A spark_connection, ml_pipeline, or a tbl_spark.                      |
| input_col      | The name of the input column.   |
| output_col     | The name of the output column.  |
| lower          | Lower quantile to calculate quantile range.                           |
| upper          | Upper quantile to calculate quantile range.                           |
| with_centering | Whether to center data with median.                                   |
| with_scaling   | Whether to scale the data to quantile range.                          |
| relative_error | The target relative error for quantile computation.                   |
| uid            | A character string used to uniquely identify the feature transformer. |
| ...            | Optional arguments; currently unused.                                 |

**Details**

In the case where x is a tbl\_spark, the estimator fits against x to obtain a transformer, returning a tbl\_spark.

**Value**

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

**See Also**

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remover\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

---

ft\_r\_formula

*Feature Transformation – RFormula (Estimator)*


---

**Description**

Implements the transforms required for fitting a dataset against an R model formula. Currently we support a limited subset of the R operators, including ~, ., :, +, and -.



**Usage**

```
ft_r_formula(
  x,
  formula = NULL,
  features_col = "features",
  label_col = "label",
  force_index_label = FALSE,
  uid = random_string("r_formula_"),
  ...
)
```

**Arguments**

|                   |  |
|-------------------|--|
| x                 | A spark_connection, ml_pipeline, or a tbl_spark.   |
| formula           | R formula as a character string or a formula. Formula objects are converted to character strings directly and the environment is not captured.   |
| features_col      | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> .   |
| label_col         | Label column name. The column should be a numeric column. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| force_index_label | (Spark 2.1.0+) Force to index label whether it is numeric or string type. Usually we index label only when it is string type. If the formula was used by classification algorithms, we can force to index label even it is numeric type by setting this param with true. Default: FALSE. |
| uid               | A character string used to uniquely identify the feature transformer.  |
| ...               | Optional arguments; currently unused.  |

**Details**

The basic operators in the formula are:

- ~ separate target and terms
- + concat terms, "+ 0" means removing intercept
- - remove a term, "- 1" means removing intercept
- : interaction (multiplication for numeric values, or binarized categorical values)
- . all columns except target

Suppose a and b are double columns, we use the following simple examples to illustrate the effect of RFormula:

- $y \sim a + b$  means model  $y \sim w_0 + w_1 * a + w_2 * b$  where  $w_0$  is the intercept and  $w_1, w_2$  are coefficients.
- $y \sim a + b + a:b - 1$  means model  $y \sim w_1 * a + w_2 * b + w_3 * a * b$  where  $w_1, w_2, w_3$  are coefficients.

RFormula produces a vector column of features and a double or string column of label. Like when formulas are used in R for linear regression, string input columns will be one-hot encoded, and numeric columns will be cast to doubles. If the label column is of type string, it will be first transformed to double with StringIndexer. If the label column does not exist in the DataFrame, the output label column will be created from the specified response variable in the formula.

In the case where x is a tbl\_spark, the estimator fits against x to obtain a transformer, returning a tbl\_spark.

### Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

### See Also

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remover\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

---

ft\_sql\_transformer      *Feature Transformation – SQLTransformer*

---

### Description

Implements the transformations which are defined by SQL statement. Currently we only support SQL syntax like 'SELECT ... FROM \_\_THIS\_\_ ...' where '\_\_THIS\_\_' represents the underlying table of the input dataset. The select clause specifies the fields, constants, and expressions to display in the output, it can be any select clause that Spark SQL supports. Users can also use Spark SQL built-in function and UDFs to operate on these selected columns.

### Usage

```
ft_sql_transformer(
  x,
  statement = NULL,
  uid = random_string("sql_transformer_"),
  ...
)

ft_dplyr_transformer(x, tbl, uid = random_string("dplyr_transformer_"), ...)
```

**Arguments**

|           |   |
|-----------|---|
| x         | A spark_connection, ml_pipeline, or a tbl_spark.                      |
| statement | A SQL statement.  |
| uid       | A character string used to uniquely identify the feature transformer. |
| ...       | Optional arguments; currently unused.                                 |
| tbl       | A tbl_spark generated using dplyr transformations.                    |

**Details**

ft\_dplyr\_transformer() is mostly a wrapper around ft\_sql\_transformer() that takes a tbl\_spark instead of a SQL statement. Internally, the ft\_dplyr\_transformer() extracts the dplyr transformations used to generate tbl as a SQL statement or a sampling operation. Note that only single-table dplyr verbs are supported and that the sdf\_ family of functions are not.

**Value**

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

**See Also**

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remover\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

---

ft\_standard\_scaler      *Feature Transformation – StandardScaler (Estimator)*

---

**Description**

Standardizes features by removing the mean and scaling to unit variance using column summary statistics on the samples in the training set. The "unit std" is computed using the corrected sample standard deviation, which is computed as the square root of the unbiased sample variance.

**Usage**

```
ft_standard_scaler(
  x,
  input_col = NULL,
  output_col = NULL,
  with_mean = FALSE,
  with_std = TRUE,
  uid = random_string("standard_scaler_"),
  ...
)
```

**Arguments**

|            |   |
|------------|---|
| x          | A spark_connection, ml_pipeline, or a tbl_spark.  |
| input_col  | The name of the input column.   |
| output_col | The name of the output column.  |
| with_mean  | Whether to center the data with mean before scaling. It will build a dense output, so take care when applying to sparse input. Default: FALSE |
| with_std   | Whether to scale the data to unit standard deviation. Default: TRUE   |
| uid        | A character string used to uniquely identify the feature transformer.   |
| ...        | Optional arguments; currently unused.   |

**Details**

In the case where x is a tbl\_spark, the estimator fits against x to obtain a transformer, returning a tbl\_spark.

**Value**

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

**See Also**

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_stop\\_words\\_remover\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

**Examples**

```
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

features <- c("Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width")

iris_tbl %>%
  ft_vector_assembler(
    input_col = features,
    output_col = "features_temp"
  ) %>%
  ft_standard_scaler(
    input_col = "features_temp",
    output_col = "features",
    with_mean = TRUE
  )

## End(Not run)
```

---

ft\_stop\_words\_remover *Feature Transformation – StopWordsRemover (Transformer)*

---

**Description**

A feature transformer that filters out stop words from input.

**Usage**

```
ft_stop_words_remover(
  x,
  input_col = NULL,
  output_col = NULL,
  case_sensitive = FALSE,
  stop_words = ml_default_stop_words(spark_connection(x), "english"),
  uid = random_string("stop_words_remover_"),
  ...
)
```

**Arguments**

|                |  |
|----------------|--|
| x              | A spark_connection, ml_pipeline, or a tbl_spark.               |
| input_col      | The name of the input column.                                  |
| output_col     | The name of the output column.                                 |
| case_sensitive | Whether to do a case sensitive comparison over the stop words. |
| stop_words     | The words to be filtered out.                                  |

uid                    A character string used to uniquely identify the feature transformer.  
 ...                    Optional arguments; currently unused.

### Value

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

### See Also

[ml\\_default\\_stop\\_words](#)

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

---

ft\_string\_indexer            *Feature Transformation – StringIndexer (Estimator)*

---

### Description

A label indexer that maps a string column of labels to an ML column of label indices. If the input column is numeric, we cast it to string and index the string values. The indices are in  $[\emptyset, \text{numLabels})$ , ordered by label frequencies. So the most frequent label gets index 0. This function is the inverse of [ft\\_index\\_to\\_string](#).

### Usage

```
ft_string_indexer(
  x,
  input_col = NULL,
  output_col = NULL,
  handle_invalid = "error",
  string_order_type = "frequencyDesc",
  uid = random_string("string_indexer_"),
  ...
)

ml_labels(model)

ft_string_indexer_model(
  x,
```

```

    input_col = NULL,
    output_col = NULL,
    labels,
    handle_invalid = "error",
    uid = random_string("string_indexer_model_"),
    ...
)

```

### Arguments

|                   |  |
|-------------------|--|
| x                 | A spark_connection, ml_pipeline, or a tbl_spark.   |
| input_col         | The name of the input column.  |
| output_col        | The name of the output column.   |
| handle_invalid    | (Spark 2.1.0+) Param for how to handle invalid entries. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (keep invalid values in a special additional bucket). Default: "error" |
| string_order_type | (Spark 2.3+)How to order labels of string column. The first label after ordering is assigned an index of 0. Options are "frequencyDesc", "frequencyAsc", "alphabetDesc", and "alphabetAsc". Defaults to "frequencyDesc".     |
| uid               | A character string used to uniquely identify the feature transformer.  |
| ...               | Optional arguments; currently unused.  |
| model             | A fitted StringIndexer model returned by ft_string_indexer()   |
| labels            | Vector of labels, corresponding to indices to be assigned.   |

### Details

In the case where x is a tbl\_spark, the estimator fits against x to obtain a transformer, returning a tbl\_spark.

### Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

ml\_labels() returns a vector of labels, corresponding to indices to be assigned.

### See Also

[ft\\_index\\_to\\_string](#)

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#),

[ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remove\(\)](#),  
[ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

---

ft\_tokenizer

*Feature Transformation – Tokenizer (Transformer)*

---

## Description

A tokenizer that converts the input string to lowercase and then splits it by white spaces.

## Usage

```
ft_tokenizer(
  x,
  input_col = NULL,
  output_col = NULL,
  uid = random_string("tokenizer_"),
  ...
)
```

## Arguments

|            |   |
|------------|---|
| x          | A spark_connection, ml_pipeline, or a tbl_spark.                      |
| input_col  | The name of the input column.   |
| output_col | The name of the output column.  |
| uid        | A character string used to uniquely identify the feature transformer. |
| ...        | Optional arguments; currently unused.                                 |

## Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

## See Also

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remove\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)



---

ft\_vector\_assembler     *Feature Transformation – VectorAssembler (Transformer)*

---

### Description

Combine multiple vectors into a single row-vector; that is, where each row element of the newly generated column is a vector formed by concatenating each row element from the specified input columns.

### Usage

```
ft_vector_assembler(  
  x,  
  input_cols = NULL,  
  output_col = NULL,  
  uid = random_string("vector_assembler_"),  
  ...  
)
```

### Arguments

|            |   |
|------------|---|
| x          | A spark_connection, ml_pipeline, or a tbl_spark.                      |
| input_cols | The names of the input columns  |
| output_col | The name of the output column.  |
| uid        | A character string used to uniquely identify the feature transformer. |
| ...        | Optional arguments; currently unused.                                 |

### Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

### See Also

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_remover\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#), [ft\\_word2vec\(\)](#)

---

|                   |   |
|-------------------|---|
| ft_vector_indexer | <i>Feature Transformation – VectorIndexer (Estimator)</i> |
|-------------------|---|

---

### Description

Indexing categorical feature columns in a dataset of Vector.

### Usage

```
ft_vector_indexer(
  x,
  input_col = NULL,
  output_col = NULL,
  handle_invalid = "error",
  max_categories = 20,
  uid = random_string("vector_indexer_"),
  ...
)
```

### Arguments

|                |  |
|----------------|--|
| x              | A spark_connection, ml_pipeline, or a tbl_spark.   |
| input_col      | The name of the input column.  |
| output_col     | The name of the output column.   |
| handle_invalid | (Spark 2.1.0+) Param for how to handle invalid entries. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (keep invalid values in a special additional bucket). Default: "error" |
| max_categories | Threshold for the number of values a categorical feature can take. If a feature is found to have > max_categories values, then it is declared continuous. Must be greater than or equal to 2. Defaults to 20.                |
| uid            | A character string used to uniquely identify the feature transformer.  |
| ...            | Optional arguments; currently unused.  |

### Details

In the case where x is a tbl\_spark, the estimator fits against x to obtain a transformer, returning a tbl\_spark.

### Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator or a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the transformer or estimator appended to it. If a tbl\_spark, it will return a tbl\_spark with the transformation applied to it.

**See Also**

Other feature transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_robust_scaler()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_slicer()`, `ft_word2vec()`

---

|                  |  |
|------------------|--|
| ft_vector_slicer | <i>Feature Transformation – VectorSlicer (Transformer)</i> |
|------------------|--|

---

**Description**

Takes a feature vector and outputs a new feature vector with a subarray of the original features.

**Usage**

```
ft_vector_slicer(
  x,
  input_col = NULL,
  output_col = NULL,
  indices = NULL,
  uid = random_string("vector_slicer_"),
  ...
)
```

**Arguments**

|                         |  |
|-------------------------|--|
| <code>x</code>          | A <code>spark_connection</code> , <code>ml_pipeline</code> , or a <code>tbl_spark</code> .       |
| <code>input_col</code>  | The name of the input column.  |
| <code>output_col</code> | The name of the output column.   |
| <code>indices</code>    | An vector of indices to select features from a vector column. Note that the indices are 0-based. |
| <code>uid</code>        | A character string used to uniquely identify the feature transformer.                            |
| <code>...</code>        | Optional arguments; currently unused.  |

**Value**

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

**See Also**

Other feature transformers: `ft_binarizer()`, `ft_bucketizer()`, `ft_chisq_selector()`, `ft_count_vectorizer()`, `ft_dct()`, `ft_elementwise_product()`, `ft_feature_hasher()`, `ft_hashing_tf()`, `ft_idf()`, `ft_imputer()`, `ft_index_to_string()`, `ft_interaction()`, `ft_lsh`, `ft_max_abs_scaler()`, `ft_min_max_scaler()`, `ft_ngram()`, `ft_normalizer()`, `ft_one_hot_encoder_estimator()`, `ft_one_hot_encoder()`, `ft_pca()`, `ft_polynomial_expansion()`, `ft_quantile_discretizer()`, `ft_r_formula()`, `ft_regex_tokenizer()`, `ft_robust_scaler()`, `ft_sql_transformer()`, `ft_standard_scaler()`, `ft_stop_words_remover()`, `ft_string_indexer()`, `ft_tokenizer()`, `ft_vector_assembler()`, `ft_vector_indexer()`, `ft_word2vec()`

ft\_word2vec

*Feature Transformation – Word2Vec (Estimator)***Description**

Word2Vec transforms a word into a code for further natural language processing or machine learning process.

**Usage**

```
ft_word2vec(
  x,
  input_col = NULL,
  output_col = NULL,
  vector_size = 100,
  min_count = 5,
  max_sentence_length = 1000,
  num_partitions = 1,
  step_size = 0.025,
  max_iter = 1,
  seed = NULL,
  uid = random_string("word2vec_"),
  ...
)
```

```
ml_find_synonyms(model, word, num)
```

**Arguments**

|             |   |
|-------------|---|
| x           | A spark_connection, ml_pipeline, or a tbl_spark.  |
| input_col   | The name of the input column.   |
| output_col  | The name of the output column.  |
| vector_size | The dimension of the code that you want to transform from words. Default: 100                                 |
| min_count   | The minimum number of times a token must appear to be included in the word2vec model's vocabulary. Default: 5 |

|                     |  |
|---------------------|--|
| max_sentence_length | (Spark 2.0.0+) Sets the maximum length (in words) of each sentence in the input data. Any sentence longer than this threshold will be divided into chunks of up to max_sentence_length size. Default: 1000 |
| num_partitions      | Number of partitions for sentences of words. Default: 1  |
| step_size           | Param for Step size to be used for each iteration of optimization (> 0).   |
| max_iter            | The maximum number of iterations to use.   |
| seed                | A random seed. Set this value if you need your results to be reproducible across repeated calls.   |
| uid                 | A character string used to uniquely identify the feature transformer.  |
| ...                 | Optional arguments; currently unused.  |
| model               | A fitted Word2Vec model, returned by ft_word2vec().  |
| word                | A word, as a length-one character vector.  |
| num                 | Number of words closest in similarity to the given word to find.   |

### Details

In the case where `x` is a `tbl_spark`, the estimator fits against `x` to obtain a transformer, returning a `tbl_spark`.

### Value

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` or a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the transformer or estimator appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the transformation applied to it.

`ml_find_synonyms()` returns a `DataFrame` of synonyms and cosine similarities

### See Also

Other feature transformers: [ft\\_binarizer\(\)](#), [ft\\_bucketizer\(\)](#), [ft\\_chisq\\_selector\(\)](#), [ft\\_count\\_vectorizer\(\)](#), [ft\\_dct\(\)](#), [ft\\_elementwise\\_product\(\)](#), [ft\\_feature\\_hasher\(\)](#), [ft\\_hashing\\_tf\(\)](#), [ft\\_idf\(\)](#), [ft\\_imputer\(\)](#), [ft\\_index\\_to\\_string\(\)](#), [ft\\_interaction\(\)](#), [ft\\_lsh](#), [ft\\_max\\_abs\\_scaler\(\)](#), [ft\\_min\\_max\\_scaler\(\)](#), [ft\\_ngram\(\)](#), [ft\\_normalizer\(\)](#), [ft\\_one\\_hot\\_encoder\\_estimator\(\)](#), [ft\\_one\\_hot\\_encoder\(\)](#), [ft\\_pca\(\)](#), [ft\\_polynomial\\_expansion\(\)](#), [ft\\_quantile\\_discretizer\(\)](#), [ft\\_r\\_formula\(\)](#), [ft\\_regex\\_tokenizer\(\)](#), [ft\\_robust\\_scaler\(\)](#), [ft\\_sql\\_transformer\(\)](#), [ft\\_standard\\_scaler\(\)](#), [ft\\_stop\\_words\\_removal\(\)](#), [ft\\_string\\_indexer\(\)](#), [ft\\_tokenizer\(\)](#), [ft\\_vector\\_assembler\(\)](#), [ft\\_vector\\_indexer\(\)](#), [ft\\_vector\\_slicer\(\)](#)

---

full\_join

*Full join*

---

### Description

See [full\\_join](#) for more details.

---

 generic\_call\_interface

*Generic Call Interface*


---

**Description**

Generic Call Interface

**Arguments**

|        |  |
|--------|--|
| sc     | spark_connection   |
| static | Is this a static method call (including a constructor). If so then the object parameter should be the name of a class (otherwise it should be a spark_job instance). |
| object | Object instance or name of class (for static)  |
| method | Name of method   |
| ...    | Call parameters  |

---

get\_spark\_sql\_catalog\_implementation

*Retrieve the Spark connection's SQL catalog implementation property*


---

**Description**

Retrieve the Spark connection's SQL catalog implementation property

**Usage**

```
get_spark_sql_catalog_implementation(sc)
```

**Arguments**

|    |                  |
|----|------------------|
| sc | spark_connection |
|----|------------------|

**Value**

spark.sql.catalogImplementation property from the connection's runtime configuration

---

|                     |   |
|---------------------|---|
| hive_context_config | <i>Runtime configuration interface for Hive</i> |
|---------------------|---|

---

**Description**

Retrieves the runtime configuration interface for Hive.

**Usage**

```
hive_context_config(sc)
```

**Arguments**

|    |                     |
|----|---------------------|
| sc | A spark_connection. |
|----|---------------------|

---

|               |   |
|---------------|---|
| hof_aggregate | <i>Apply Aggregate Function to Array Column</i> |
|---------------|---|

---

**Description**

Apply an element-wise aggregation function to an array column (this is essentially a dplyr wrapper for the `aggregate(array<T>, A, function<A, T, A>[, function<A, R>])`: R built-in Spark SQL functions)

**Usage**

```
hof_aggregate(
  x,
  start,
  merge,
  finish = NULL,
  expr = NULL,
  dest_col = NULL,
  ...
)
```

**Arguments**

|          |   |
|----------|---|
| x        | The Spark data frame to run aggregation on  |
| start    | The starting value of the aggregation   |
| merge    | The aggregation function  |
| finish   | Optional param specifying a transformation to apply on the final value of the aggregation   |
| expr     | The array being aggregated, could be any SQL expression evaluating to an array (default: the last column of the Spark data frame) |
| dest_col | Column to store the aggregated result (default: expr)   |
| ...      | Additional params to <code>dplyr::mutate</code>   |

**Examples**

```
## Not run:

library(sparklyr)
sc <- spark_connect(master = "local")
# concatenates all numbers of each array in `array_column` and add parentheses
# around the resulting string
copy_to(sc, dplyr::tibble(array_column = list(1:5, 21:25))) %>%
  hof_aggregate(
    start = "",
    merge = ~ CONCAT(.y, .x),
    finish = ~ CONCAT("(", .x, ")")
  )

## End(Not run)
```

---

hof\_array\_sort

*Sorts array using a custom comparator*


---

**Description**

Applies a custom comparator function to sort an array (this is essentially a dplyr wrapper to the `'array_sort(expr, func)'` higher- order function, which is supported since Spark 3.0)

**Usage**

```
hof_array_sort(x, func, expr = NULL, dest_col = NULL, ...)
```

**Arguments**

|          |  |
|----------|--|
| x        | The Spark data frame to be processed   |
| func     | The comparator function to apply (it should take 2 array elements as arguments and return an integer, with a return value of -1 indicating the first element is less than the second, 0 indicating equality, or 1 indicating the first element is greater than the second) |
| expr     | The array being sorted, could be any SQL expression evaluating to an array (default: the last column of the Spark data frame)  |
| dest_col | Column to store the sorted result (default: expr)  |
| ...      | Additional params to dplyr::mutate   |

**Examples**

```
## Not run:

library(sparklyr)
sc <- spark_connect(master = "local", version = "3.0.0")
```



```

copy_to(
  sc,
  dplyr::tibble(
    # x contains 2 arrays each having elements in ascending order
    x = list(1:5, 6:10)
  )
) %>%
  # now each array from x gets sorted in descending order
  hof_array_sort(~ as.integer(sign(.y - .x)))

## End(Not run)

```

---

hof\_exists

*Determine Whether Some Element Exists in an Array Column*


---

### Description

Determines whether an element satisfying the given predicate exists in each array from an array column (this is essentially a dplyr wrapper for the exists(array<T>, function<T, Boolean>): Boolean built-in Spark SQL function)

### Usage

```
hof_exists(x, pred, expr = NULL, dest_col = NULL, ...)
```

### Arguments

|          |   |
|----------|---|
| x        | The Spark data frame to search  |
| pred     | A boolean predicate   |
| expr     | The array being searched (could be any SQL expression evaluating to an array) |
| dest_col | Column to store the search result   |
| ...      | Additional params to dplyr::mutate  |

---

hof\_filter

*Filter Array Column*


---

### Description

Apply an element-wise filtering function to an array column (this is essentially a dplyr wrapper for the filter(array<T>, function<T, Boolean>): array<T> built-in Spark SQL functions)

### Usage

```
hof_filter(x, func, expr = NULL, dest_col = NULL, ...)
```

**Arguments**

|          |   |
|----------|---|
| x        | The Spark data frame to filter  |
| func     | The filtering function  |
| expr     | The array being filtered, could be any SQL expression evaluating to an array (default: the last column of the Spark data frame) |
| dest_col | Column to store the filtered result (default: expr)   |
| ...      | Additional params to dplyr::mutate  |

**Examples**

```
## Not run:

library(sparklyr)
sc <- spark_connect(master = "local")
# only keep odd elements in each array in `array_column`
copy_to(sc, dplyr::tibble(array_column = list(1:5, 21:25))) %>%
  hof_filter(~ .x %% 2 == 1)

## End(Not run)
```

---

 hof\_forall

*Checks whether all elements in an array satisfy a predicate*


---

**Description**

Checks whether the predicate specified holds for all elements in an array (this is essentially a dplyr wrapper to the 'forall(expr, pred)' higher- order function, which is supported since Spark 3.0)

**Usage**

```
hof_forall(x, pred, expr = NULL, dest_col = NULL, ...)
```

**Arguments**

|          |   |
|----------|---|
| x        | The Spark data frame to be processed  |
| pred     | The predicate to test (it should take an array element as argument and return a boolean value)                                |
| expr     | The array being tested, could be any SQL expression evaluating to an array (default: the last column of the Spark data frame) |
| dest_col | Column to store the boolean result (default: expr)  |
| ...      | Additional params to dplyr::mutate  |

**Examples**

```
## Not run:

sc <- spark_connect(master = "local", version = "3.0.0")
df <- dplyr::tibble(
  x = list(c(1, 2, 3, 4, 5), c(6, 7, 8, 9, 10)),
  y = list(c(1, 4, 2, 8, 5), c(7, 1, 4, 2, 8)),
)
sdf <- sdf_copy_to(sc, df, overwrite = TRUE)

all_positive_tbl <- sdf %>%
  hof_forall(pred = ~ .x > 0, expr = y, dest_col = all_positive) %>%
  dplyr::select(all_positive)

## End(Not run)
```

---

|                |                      |
|----------------|----------------------|
| hof_map_filter | <i>Filters a map</i> |
|----------------|----------------------|

---

**Description**

Filters entries in a map using the function specified (this is essentially a dplyr wrapper to the ‘map\_filter(expr, func)’ higher- order function, which is supported since Spark 3.0)

**Usage**

```
hof_map_filter(x, func, expr = NULL, dest_col = NULL, ...)
```

**Arguments**

|          |   |
|----------|---|
| x        | The Spark data frame to be processed  |
| func     | The filter function to apply (it should take (key, value) as arguments and return a boolean value, with FALSE indicating the key-value pair should be discarded and TRUE otherwise) |
| expr     | The map being filtered, could be any SQL expression evaluating to a map (default: the last column of the Spark data frame)  |
| dest_col | Column to store the filtered result (default: expr)   |
| ...      | Additional params to dplyr::mutate  |

**Examples**

```
## Not run:

library(sparklyr)
sc <- spark_connect(master = "local", version = "3.0.0")
sdf <- sdf_len(sc, 1) %>% dplyr::mutate(m = map(1, 0, 2, 2, 3, -1))
```

```
filtered_sdf <- sdf %>% hof_map_filter(~ .x > .y)

## End(Not run)
```

---

|                  |                                 |
|------------------|---------------------------------|
| hof_map_zip_with | <i>Merges two maps into one</i> |
|------------------|---------------------------------|

---

### Description

Merges two maps into a single map by applying the function specified to pairs of values with the same key (this is essentially a dplyr wrapper to the ‘map\_zip\_with(map1, map2, func)’ higher-order function, which is supported since Spark 3.0)

### Usage

```
hof_map_zip_with(x, func, dest_col = NULL, map1 = NULL, map2 = NULL, ...)
```

### Arguments

|          |  |
|----------|--|
| x        | The Spark data frame to be processed   |
| func     | The function to apply (it should take (key, value1, value2) as arguments, where (key, value1) is a key-value pair present in map1, (key, value2) is a key-value pair present in map2, and return a transformed value associated with key in the resulting map) |
| dest_col | Column to store the query result (default: the last column of the Spark data frame)  |
| map1     | The first map being merged, could be any SQL expression evaluating to a map (default: the first column of the Spark data frame)  |
| map2     | The second map being merged, could be any SQL expression evaluating to a map (default: the second column of the Spark data frame)  |
| ...      | Additional params to dplyr::mutate   |

### Examples

```
## Not run:

library(sparklyr)
sc <- spark_connect(master = "local", version = "3.0.0")

# create a Spark dataframe with 2 columns of type MAP<STRING, INT>
two_maps_tbl <- sdf_copy_to(
  sc,
  dplyr::tibble(
    m1 = c("{\\"1\\":2,\\"3\\":4,\\"5\\":6}", "{\\"2\\":1,\\"4\\":3,\\"6\\":5}"),
    m2 = c("{\\"1\\":1,\\"3\\":3,\\"5\\":5}", "{\\"2\\":2,\\"4\\":4,\\"6\\":6}"),
  ),
```

```

    overwrite = TRUE
  ) %>%
    dplyr::mutate(m1 = from_json(m1, "MAP<STRING, INT>"),
                  m2 = from_json(m2, "MAP<STRING, INT>"))

# create a 3rd column containing MAP<STRING, INT> values derived from the
# first 2 columns

transformed_two_maps_tbl <- two_maps_tbl %>%
  hof_map_zip_with(
    func = .(k, v1, v2) %->% (CONCAT(k, "_", v1, "_", v2)),
    dest_col = m3
  )

## End(Not run)

```

---

hof\_transform

*Transform Array Column*


---

### Description

Apply an element-wise transformation function to an array column (this is essentially a dplyr wrapper for the `transform(array<T>, function<T, U>): array<U>` and the `transform(array<T>, function<T, Int, U>): array<U>` built-in Spark SQL functions)

### Usage

```
hof_transform(x, func, expr = NULL, dest_col = NULL, ...)
```

### Arguments

|                       |  |
|-----------------------|--|
| <code>x</code>        | The Spark data frame to transform  |
| <code>func</code>     | The transformation to apply  |
| <code>expr</code>     | The array being transformed, could be any SQL expression evaluating to an array (default: the last column of the Spark data frame) |
| <code>dest_col</code> | Column to store the transformed result (default: <code>expr</code> )   |
| <code>...</code>      | Additional params to <code>dplyr::mutate</code>  |

### Examples

```

## Not run:

library(sparklyr)
sc <- spark_connect(master = "local")
# applies the (x -> x * x) transformation to elements of all arrays
copy_to(sc, dplyr::tibble(arr = list(1:5, 21:25))) %>%
  hof_transform(~ .x * .x)

```

```
## End(Not run)
```

---

hof\_transform\_keys      *Transforms keys of a map*

---

### Description

Applies the transformation function specified to all keys of a map (this is essentially a dplyr wrapper to the 'transform\_keys(expr, func)' higher- order function, which is supported since Spark 3.0)

### Usage

```
hof_transform_keys(x, func, expr = NULL, dest_col = NULL, ...)
```

### Arguments

|          |   |
|----------|---|
| x        | The Spark data frame to be processed  |
| func     | The transformation function to apply (it should take (key, value) as arguments and return a transformed key)                  |
| expr     | The map being transformed, could be any SQL expression evaluating to a map (default: the last column of the Spark data frame) |
| dest_col | Column to store the transformed result (default: expr)  |
| ...      | Additional params to dplyr::mutate  |

### Examples

```
## Not run:

library(sparklyr)
sc <- spark_connect(master = "local", version = "3.0.0")
sdf <- sdf_len(sc, 1) %>% dplyr::mutate(m = map("a", 0L, "b", 2L, "c", -1L))
transformed_sdf <- sdf %>% hof_transform_keys(~ CONCAT(.x, " == ", .y))

## End(Not run)
```

---

hof\_transform\_values *Transforms values of a map*

---

### Description

Applies the transformation function specified to all values of a map (this is essentially a dplyr wrapper to the 'transform\_values(expr, func)' higher- order function, which is supported since Spark 3.0)

### Usage

```
hof_transform_values(x, func, expr = NULL, dest_col = NULL, ...)
```

### Arguments

|          |   |
|----------|---|
| x        | The Spark data frame to be processed  |
| func     | The transformation function to apply (it should take (key, value) as arguments and return a transformed value)                |
| expr     | The map being transformed, could be any SQL expression evaluating to a map (default: the last column of the Spark data frame) |
| dest_col | Column to store the transformed result (default: expr)  |
| ...      | Additional params to dplyr::mutate  |

### Examples

```
## Not run:

library(sparklyr)
sc <- spark_connect(master = "local", version = "3.0.0")
sdf <- sdf_len(sc, 1) %>% dplyr::mutate(m = map("a", 0L, "b", 2L, "c", -1L))
transformed_sdf <- sdf %>% hof_transform_values(~ CONCAT(.x, " == ", .y))

## End(Not run)
```

---

hof\_zip\_with *Combines 2 Array Columns*

---

### Description

Applies an element-wise function to combine elements from 2 array columns (this is essentially a dplyr wrapper for the zip\_with(array<T>, array<U>, function<T, U, R>): array<R> built-in function in Spark SQL)

**Usage**

```
hof_zip_with(x, func, dest_col = NULL, left = NULL, right = NULL, ...)
```

**Arguments**

|                       |  |
|-----------------------|--|
| <code>x</code>        | The Spark data frame to process  |
| <code>func</code>     | Element-wise combining function to be applied  |
| <code>dest_col</code> | Column to store the query result (default: the last column of the Spark data frame)        |
| <code>left</code>     | Any expression evaluating to an array (default: the first column of the Spark data frame)  |
| <code>right</code>    | Any expression evaluating to an array (default: the second column of the Spark data frame) |
| <code>...</code>      | Additional params to <code>dplyr::mutate</code>  |

**Examples**

```
## Not run:

library(sparklyr)
sc <- spark_connect(master = "local")
# compute element-wise products of 2 arrays from each row of `left` and `right`
# and store the resulting array in `res`
copy_to(
  sc,
  dplyr::tibble(
    left = list(1:5, 21:25),
    right = list(6:10, 16:20),
    res = c(0, 0)
  )
) %>%
  hof_zip_with(~ .x * .y)

## End(Not run)
```

---

 inner\_join

*Inner join*


---

**Description**

See [inner\\_join](#) for more details.



---

|        |  |
|--------|--|
| invoke | <i>Invoke a Method on a JVM Object</i> |
|--------|--|

---

### Description

Invoke methods on Java object references. These functions provide a mechanism for invoking various Java object methods directly from R.

### Usage

```
invoke(jobj, method, ...)
invoke_static(sc, class, method, ...)
invoke_new(sc, class, ...)
```

### Arguments

|        |  |
|--------|--|
| jobj   | An R object acting as a Java object reference (typically, a spark_jobj). |
| method | The name of the method to be invoked.                                    |
| ...    | Optional arguments, currently unused.                                    |
| sc     | A spark_connection.  |
| class  | The name of the Java class whose methods should be invoked.              |

### Details

Use each of these functions in the following scenarios:

|               |  |
|---------------|--|
| invoke        | Execute a method on a Java object reference (typically, a spark_jobj). |
| invoke_static | Execute a static method associated with a Java class.                  |
| invoke_new    | Invoke a constructor associated with a Java class.                     |

### Examples

```
sc <- spark_connect(master = "spark://HOST:PORT")
spark_context(sc) %>%
  invoke("textFile", "file.csv", 1L) %>%
  invoke("count")
```

---

|        |   |
|--------|---|
| jarray | <i>Instantiate a Java array with a specific element type.</i> |
|--------|---|

---

**Description**

Given a list of Java object references, instantiate an `Array[T]` containing the same list of references, where `T` is a non-primitive type that is more specific than `java.lang.Object`.

**Usage**

```
jarray(sc, x, element_type)
```

**Arguments**

|                           |  |
|---------------------------|--|
| <code>sc</code>           | A <code>spark_connection</code> .  |
| <code>x</code>            | A list of Java object references.  |
| <code>element_type</code> | A valid Java class name representing the generic type parameter of the Java array to be instantiated. Each element of <code>x</code> must refer to a Java object that is assignable to <code>element_type</code> . |

**Examples**

```
sc <- spark_connect(master = "spark://HOST:PORT")

string_arr <- jarray(sc, letters, element_type = "java.lang.String")
# string_arr is now a reference to an array of type String[]
```

---

jfloat

*Instantiate a Java float type.*

---

**Description**

Instantiate a `java.lang.Float` object with the value specified. NOTE: this method is useful when one has to invoke a Java/Scala method requiring a float (instead of double) type for at least one of its parameters.

**Usage**

```
jfloat(sc, x)
```

**Arguments**

|                 |                                   |
|-----------------|-----------------------------------|
| <code>sc</code> | A <code>spark_connection</code> . |
| <code>x</code>  | A numeric value in R.             |

**Examples**

```
sc <- spark_connect(master = "spark://HOST:PORT")

jflt <- jfloat(sc, 1.23e-8)
# jflt is now a reference to a java.lang.Float object
```

---

|              |                                     |
|--------------|-------------------------------------|
| jfloat_array | <i>Instantiate an Array[Float].</i> |
|--------------|-------------------------------------|

---

**Description**

Instantiate an Array[Float] object with the value specified. NOTE: this method is useful when one has to invoke a Java/Scala method requiring an Array[Float] as one of its parameters.

**Usage**

```
jfloat_array(sc, x)
```

**Arguments**

|    |                        |
|----|------------------------|
| sc | A spark_connection.    |
| x  | A numeric vector in R. |

**Examples**

```
sc <- spark_connect(master = "spark://HOST:PORT")

jflt_arr <- jfloat_array(sc, c(-1.23e-8, 0, -1.23e-8))
# jflt_arr is now a reference an array of java.lang.Float
```

---

|                |                         |
|----------------|-------------------------|
| join.tbl_spark | <i>Join Spark tbls.</i> |
|----------------|-------------------------|

---

**Description**

These functions are wrappers around their 'dplyr' equivalents that set Spark SQL-compliant values for the 'suffix' argument by replacing dots ('.') with underscores ('\_'). See [join] for a description of the general purpose of the functions.

**Usage**

```
## S3 method for class 'tbl_spark'
inner_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c("_x", "_y"),
  auto_index = FALSE,
  ...,
  sql_on = NULL
```

```

)

## S3 method for class 'tbl_spark'
left_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c("_x", "_y"),
  auto_index = FALSE,
  ...,
  sql_on = NULL
)

## S3 method for class 'tbl_spark'
right_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c("_x", "_y"),
  auto_index = FALSE,
  ...,
  sql_on = NULL
)

## S3 method for class 'tbl_spark'
full_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c("_x", "_y"),
  auto_index = FALSE,
  ...,
  sql_on = NULL
)

```

### Arguments

|                   |   |
|-------------------|---|
| <code>x, y</code> | A pair of lazy data frames backed by database queries.  |
| <code>by</code>   | <p>A join specification created with <code>join_by()</code>, or a character vector of variables to join by.</p> <p>If <code>NULL</code>, the default, <code>*_join()</code> will perform a natural join, using all variables in common across <code>x</code> and <code>y</code>. A message lists the variables so that you can check they're correct; suppress the message by supplying <code>by</code> explicitly.</p> <p>To join on different variables between <code>x</code> and <code>y</code>, use a <code>join_by()</code> specification. For example, <code>join_by(a == b)</code> will match <code>x\$a</code> to <code>y\$b</code>.</p> |

To join by multiple variables, use a `join_by()` specification with multiple expressions. For example, `join_by(a == b, c == d)` will match `x$a` to `y$b` and `x$c` to `y$d`. If the column names are the same between `x` and `y`, you can shorten this by listing only the variable names, like `join_by(a, c)`.

`join_by()` can also be used to perform inequality, rolling, and overlap joins. See the documentation at [?join\\_by](#) for details on these types of joins.

For simple equality joins, you can alternatively specify a character vector of variable names to join by. For example, `by = c("a", "b")` joins `x$a` to `y$a` and `x$b` to `y$b`. If variable names differ between `x` and `y`, use a named character vector like `by = c("x_a" = "y_a", "x_b" = "y_b")`.

To perform a cross-join, generating all combinations of `x` and `y`, see `cross_join()`.

|                         |  |
|-------------------------|--|
| <code>copy</code>       | If <code>x</code> and <code>y</code> are not from the same data source, and <code>copy</code> is <code>TRUE</code> , then <code>y</code> will be copied into a temporary table in same database as <code>x</code> . <code>*_join()</code> will automatically run <code>ANALYZE</code> on the created table in the hope that this will make you queries as efficient as possible by giving more data to the query planner.<br>This allows you to join tables across <code>srcs</code> , but it's potentially expensive operation so you must opt into it. |
| <code>suffix</code>     | If there are non-joined duplicate variables in <code>x</code> and <code>y</code> , these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.  |
| <code>auto_index</code> | if <code>copy</code> is <code>TRUE</code> , automatically create indices for the variables in <code>by</code> . This may speed up the join if there are matching indexes in <code>x</code> .   |
| <code>...</code>        | Other parameters passed onto methods.  |
| <code>sql_on</code>     | A custom join predicate as an SQL expression. Usually joins use column equality, but you can perform more complex queries by supply <code>sql_on</code> which should be a SQL expression that uses LHS and RHS aliases to refer to the left-hand side or right-hand side of the join respectively.   |

---

|                       |                                |
|-----------------------|--------------------------------|
| <code>j_invoke</code> | <i>Invoke a Java function.</i> |
|-----------------------|--------------------------------|

---

## Description

Invoke a Java function and force return value of the call to be retrieved as a Java object reference.

## Usage

```
j_invoke(jobj, method, ...)
```

```
j_invoke_static(sc, class, method, ...)
```

```
j_invoke_new(sc, class, ...)
```

**Arguments**

|        |  |
|--------|--|
| jobj   | An R object acting as a Java object reference (typically, a spark_jobj). |
| method | The name of the method to be invoked.                                    |
| ...    | Optional arguments, currently unused.                                    |
| sc     | A spark_connection.  |
| class  | The name of the Java class whose methods should be invoked.              |

---

|           |                  |
|-----------|------------------|
| left_join | <i>Left join</i> |
|-----------|------------------|

---

**Description**

See [left\\_join](#) for more details.

---

|                    |   |
|--------------------|---|
| list_sparklyr_jars | <i>list all sparklyr-*.jar files that have been built</i> |
|--------------------|---|

---

**Description**

list all sparklyr-\*.jar files that have been built

**Usage**

```
list_sparklyr_jars()
```

---

|             |  |
|-------------|--|
| livy_config | <i>Create a Spark Configuration for Livy</i> |
|-------------|--|

---

**Description**

Create a Spark Configuration for Livy

**Usage**

```
livy_config(
  config = spark_config(),
  username = NULL,
  password = NULL,
  negotiate = FALSE,
  custom_headers = list(`X-Requested-By` = "sparklyr"),
  proxy = NULL,
  curl_opts = NULL,
  ...
)
```

**Arguments**

|                |  |
|----------------|--|
| config         | Optional base configuration  |
| username       | The username to use in the Authorization header  |
| password       | The password to use in the Authorization header  |
| negotiate      | Whether to use gssnegotiate method or not  |
| custom_headers | List of custom headers to append to http requests. Defaults to <code>list("X-Requested-By" = "sparklyr")</code> .  |
| proxy          | Either NULL or a proxy specified by <code>httr::use_proxy()</code> . Defaults to NULL.   |
| curl_opts      | List of CURL options (e.g., <code>verbose</code> , <code>connecttimeout</code> , <code>dns_cache_timeout</code> , etc, see <code>httr::httr_options()</code> for a list of valid options) – NOTE: these configurations are for libcurl only and separate from HTTP headers or Livy session parameters. |
| ...            | additional Livy session parameters   |

**Details**

Extends a Spark `spark_config()` configuration with settings for Livy. For instance, `username` and `password` define the basic authentication settings for a Livy session.

The default value of `"custom_headers"` is set to `list("X-Requested-By" = "sparklyr")` in order to facilitate connection to Livy servers with CSRF protection enabled.

Additional parameters for Livy sessions are:

|                                |   |
|--------------------------------|---|
| <code>proxy_user</code>        | User to impersonate when starting the session     |
| <code>jars</code>              | jars to be used in this session                   |
| <code>py_files</code>          | Python files to be used in this session           |
| <code>files</code>             | files to be used in this session                  |
| <code>driver_memory</code>     | Amount of memory to use for the driver process    |
| <code>driver_cores</code>      | Number of cores to use for the driver process     |
| <code>executor_memory</code>   | Amount of memory to use per executor process      |
| <code>executor_cores</code>    | Number of cores to use for each executor          |
| <code>num_executors</code>     | Number of executors to launch for this session    |
| <code>archives</code>          | Archives to be used in this session               |
| <code>queue</code>             | The name of the YARN queue to which submitted     |
| <code>name</code>              | The name of this session                          |
| <code>heartbeat_timeout</code> | Timeout in seconds to which session be orphaned   |
| <code>conf</code>              | Spark configuration properties (Map of key=value) |

Note that `queue` is supported only by version 0.4.0 of Livy or newer. If you are using the older one, specify `queue` via `config` (e.g. `config = spark_config(spark.yarn.queue = "my_queue")`).

**Value**

Named list with configuration data

---

|                    |                   |
|--------------------|-------------------|
| livy_service_start | <i>Start Livy</i> |
|--------------------|-------------------|

---

**Description**

Starts the livy service.

Stops the running instances of the livy service.

**Usage**

```
livy_service_start(
  version = NULL,
  spark_version = NULL,
  stdout = "",
  stderr = "",
  ...
)
```

```
livy_service_stop()
```

**Arguments**

|                |   |
|----------------|---|
| version        | The version of 'livy' to use.   |
| spark_version  | The version of 'spark' to connect to.   |
| stdout, stderr | where output to 'stdout' or 'stderr' should be sent. Same options as system2. |
| ...            | Optional arguments; currently unused.   |

---

|           |                             |
|-----------|-----------------------------|
| ml-params | <i>Spark ML – ML Params</i> |
|-----------|-----------------------------|

---

**Description**

Helper methods for working with parameters for ML objects.

**Usage**

```
ml_is_set(x, param, ...)
```

```
ml_param_map(x, ...)
```

```
ml_param(x, param, allow_null = FALSE, ...)
```

```
ml_params(x, params = NULL, allow_null = FALSE, ...)
```



**Arguments**

|            |   |
|------------|---|
| x          | A Spark ML object, either a pipeline stage or an evaluator.   |
| param      | The parameter to extract or set.  |
| ...        | Optional arguments; currently unused.   |
| allow_null | Whether to allow NULL results when extracting parameters. If FALSE, an error will be thrown if the specified parameter is not found. Defaults to FALSE. |
| params     | A vector of parameters to extract.  |

---

|                |                                     |
|----------------|-------------------------------------|
| ml-persistence | <i>Spark ML – Model Persistence</i> |
|----------------|-------------------------------------|

---

**Description**

Save/load Spark ML objects

**Usage**

```
ml_save(x, path, overwrite = FALSE, ...)

## S3 method for class 'ml_model'
ml_save(
  x,
  path,
  overwrite = FALSE,
  type = c("pipeline_model", "pipeline"),
  ...
)

ml_load(sc, path)
```

**Arguments**

|           |   |
|-----------|---|
| x         | A ML object, which could be a ml_pipeline_stage or a ml_model |
| path      | The path where the object is to be serialized/deserialized.   |
| overwrite | Whether to overwrite the existing path, defaults to FALSE.    |
| ...       | Optional arguments; currently unused.                         |
| type      | Whether to save the pipeline model or the pipeline.           |
| sc        | A Spark connection.   |

**Value**

`ml_save()` serializes a Spark object into a format that can be read back into sparklyr or by the Scala or PySpark APIs. When called on `ml_model` objects, i.e. those that were created via the `tbl_spark - formula` signature, the associated pipeline model is serialized. In other words, the saved model contains both the data processing (`RFormulaModel`) stage and the machine learning stage.

`ml_load()` reads a saved Spark object into sparklyr. It calls the correct Scala load method based on parsing the saved metadata. Note that a `PipelineModel` object saved from a sparklyr `ml_model` via `ml_save()` will be read back in as an `ml_pipeline_model`, rather than the `ml_model` object.

---

ml-transform-methods *Spark ML – Transform, fit, and predict methods (ml\_interface)*

---

**Description**

Methods for transformation, fit, and prediction. These are mirrors of the corresponding [sdf-transform-methods](#).

**Usage**

```
is_ml_transformer(x)

is_ml_estimator(x)

ml_fit(x, dataset, ...)

## Default S3 method:
ml_fit(x, dataset, ...)

ml_transform(x, dataset, ...)

ml_fit_and_transform(x, dataset, ...)

ml_predict(x, dataset, ...)

## S3 method for class 'ml_model_classification'
ml_predict(x, dataset, probability_prefix = "probability_", ...)
```

**Arguments**

|                                 |   |
|---------------------------------|---|
| <code>x</code>                  | A <code>ml_estimator</code> , <code>ml_transformer</code> (or a list thereof), or <code>ml_model</code> object. |
| <code>dataset</code>            | A <code>tbl_spark</code> .  |
| <code>...</code>                | Optional arguments; currently unused.   |
| <code>probability_prefix</code> | String used to prepend the class probability output columns.  |

**Details**

These methods are

**Value**

When `x` is an estimator, `ml_fit()` returns a transformer whereas `ml_fit_and_transform()` returns a transformed dataset. When `x` is a transformer, `ml_transform()` and `ml_predict()` return a transformed dataset. When `ml_predict()` is called on a `ml_model` object, additional columns (e.g. probabilities in case of classification models) are appended to the transformed output for the user's convenience.

---

`ml-tuning`*Spark ML – Tuning*

---

**Description**

Perform hyper-parameter tuning using either K-fold cross validation or train-validation split.

**Usage**

```
ml_sub_models(model)
```

```
ml_validation_metrics(model)
```

```
ml_cross_validator(  
  x,  
  estimator = NULL,  
  estimator_param_maps = NULL,  
  evaluator = NULL,  
  num_folds = 3,  
  collect_sub_models = FALSE,  
  parallelism = 1,  
  seed = NULL,  
  uid = random_string("cross_validator_"),  
  ...  
)
```

```
ml_train_validation_split(  
  x,  
  estimator = NULL,  
  estimator_param_maps = NULL,  
  evaluator = NULL,  
  train_ratio = 0.75,  
  collect_sub_models = FALSE,  
  parallelism = 1,  
  seed = NULL,  
  uid = random_string("train_validation_split_"),
```

```
    ...
  )
```

## Arguments

|                                   |  |
|-----------------------------------|--|
| <code>model</code>                | A cross validation or train-validation-split model.  |
| <code>x</code>                    | A <code>spark_connection</code> , <code>ml_pipeline</code> , or a <code>tbl_spark</code> .   |
| <code>estimator</code>            | A <code>ml_estimator</code> object.  |
| <code>estimator_param_maps</code> | A named list of stages and hyper-parameter sets to tune. See details.  |
| <code>evaluator</code>            | A <code>ml_evaluator</code> object, see <a href="#">ml_evaluator</a> .   |
| <code>num_folds</code>            | Number of folds for cross validation. Must be $\geq 2$ . Default: 3  |
| <code>collect_sub_models</code>   | Whether to collect a list of sub-models trained during tuning. If set to <code>FALSE</code> , then only the single best sub-model will be available after fitting. If set to <code>true</code> , then all sub-models will be available. Warning: For large models, collecting all sub-models can cause OOMs on the Spark driver. |
| <code>parallelism</code>          | The number of threads to use when running parallel algorithms. Default is 1 for serial execution.  |
| <code>seed</code>                 | A random seed. Set this value if you need your results to be reproducible across repeated calls.   |
| <code>uid</code>                  | A character string used to uniquely identify the ML estimator.   |
| <code>...</code>                  | Optional arguments; currently unused.  |
| <code>train_ratio</code>          | Ratio between train and validation data. Must be between 0 and 1. Default: 0.75  |

## Details

`ml_cross_validator()` performs k-fold cross validation while `ml_train_validation_split()` performs tuning on one pair of train and validation datasets.

## Value

The object returned depends on the class of `x`.

- `spark_connection`: When `x` is a `spark_connection`, the function returns an instance of a `ml_cross_validator` or `ml_train_validation_split` object.
- `ml_pipeline`: When `x` is a `ml_pipeline`, the function returns a `ml_pipeline` with the tuning estimator appended to the pipeline.
- `tbl_spark`: When `x` is a `tbl_spark`, a tuning estimator is constructed then immediately fit with the input `tbl_spark`, returning a `ml_cross_validation_model` or a `ml_train_validation_split_model` object.

For cross validation, `ml_sub_models()` returns a nested list of models, where the first layer represents fold indices and the second layer represents param maps. For train-validation split, `ml_sub_models()` returns a list of models, corresponding to the order of the estimator param maps.

`ml_validation_metrics()` returns a data frame of performance metrics and hyperparameter combinations.

**Examples**

```

## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

# Create a pipeline
pipeline <- ml_pipeline(sc) %>%
  ft_r_formula(Species ~ .) %>%
  ml_random_forest_classifier()

# Specify hyperparameter grid
grid <- list(
  random_forest = list(
    num_trees = c(5, 10),
    max_depth = c(5, 10),
    impurity = c("entropy", "gini")
  )
)

# Create the cross validator object
cv <- ml_cross_validator(
  sc,
  estimator = pipeline, estimator_param_maps = grid,
  evaluator = ml_multiclass_classification_evaluator(sc),
  num_folds = 3,
  parallelism = 4
)

# Train the models
cv_model <- ml_fit(cv, iris_tbl)

# Print the metrics
ml_validation_metrics(cv_model)

## End(Not run)

```

---

ml\_aft\_survival\_regression

*Spark ML – Survival Regression*


---

**Description**

Fit a parametric survival regression model named accelerated failure time (AFT) model (see [Accelerated failure time model \(Wikipedia\)](#)) based on the Weibull distribution of the survival time.

**Usage**

```
ml_aft_survival_regression(
```

```

x,
formula = NULL,
censor_col = "censor",
quantile_probabilities = c(0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99),
fit_intercept = TRUE,
max_iter = 100L,
tol = 1e-06,
aggregation_depth = 2,
quantiles_col = NULL,
features_col = "features",
label_col = "label",
prediction_col = "prediction",
uid = random_string("aft_survival_regression_"),
...
)

ml_survival_regression(
  x,
  formula = NULL,
  censor_col = "censor",
  quantile_probabilities = c(0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99),
  fit_intercept = TRUE,
  max_iter = 100L,
  tol = 1e-06,
  aggregation_depth = 2,
  quantiles_col = NULL,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  uid = random_string("aft_survival_regression_"),
  response = NULL,
  features = NULL,
  ...
)

```

### Arguments

|                        |   |
|------------------------|---|
| x                      | A spark_connection, ml_pipeline, or a tbl_spark.  |
| formula                | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details. |
| censor_col             | Censor column name. The value of this column could be 0 or 1. If the value is 1, it means the event has occurred i.e. uncensored; otherwise censored.                                 |
| quantile_probabilities | Quantile probabilities array. Values of the quantile probabilities array should be in the range (0, 1) and the array should be non-empty.   |
| fit_intercept          | Boolean; should the model be fit with an intercept term?  |
| max_iter               | The maximum number of iterations to use.  |

|                   |  |
|-------------------|--|
| tol               | Param for the convergence tolerance for iterative algorithms.  |
| aggregation_depth | (Spark 2.1.0+) Suggested depth for treeAggregate ( $\geq 2$ ).   |
| quantiles_col     | Quantiles column name. This column will output quantiles of corresponding quantileProbabilities if it is set.  |
| features_col      | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> . |
| label_col         | Label column name. The column should be a numeric column. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| prediction_col    | Prediction column name.  |
| uid               | A character string used to uniquely identify the ML estimator.   |
| ...               | Optional arguments; see Details.   |
| response          | (Deprecated) The name of the response column (as a length-one character vector.)   |
| features          | (Deprecated) The name of features (terms) to use for the model fit.  |

### Details

`ml_survival_regression()` is an alias for `ml_aft_survival_regression()` for backwards compatibility.

### Value

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the predictor appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the predictions added to it.

### See Also

Other ml algorithms: [ml\\_decision\\_tree\\_classifier\(\)](#), [ml\\_gbt\\_classifier\(\)](#), [ml\\_generalized\\_linear\\_regression\(\)](#), [ml\\_isotonic\\_regression\(\)](#), [ml\\_linear\\_regression\(\)](#), [ml\\_linear\\_svc\(\)](#), [ml\\_logistic\\_regression\(\)](#), [ml\\_multilayer\\_perceptron\\_classifier\(\)](#), [ml\\_naive\\_bayes\(\)](#), [ml\\_one\\_vs\\_rest\(\)](#), [ml\\_random\\_forest\\_classifier\(\)](#)

### Examples

```
## Not run:

library(survival)
library(sparklyr)

sc <- spark_connect(master = "local")
ovarian_tbl <- sdf_copy_to(sc, ovarian, name = "ovarian_tbl", overwrite = TRUE)

partitions <- ovarian_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

ovarian_training <- partitions$training
```

```

ovarian_test <- partitions$test

sur_reg <- ovarian_training %>%
  ml_aft_survival_regression(futime ~ ecog_ps + rx + age + resid_ds, censor_col = "fustat")

pred <- ml_predict(sur_reg, ovarian_test)
pred

## End(Not run)

```

---

ml\_als

*Spark ML – ALS*


---

## Description

Perform recommendation using Alternating Least Squares (ALS) matrix factorization.

## Usage

```

ml_als(
  x,
  formula = NULL,
  rating_col = "rating",
  user_col = "user",
  item_col = "item",
  rank = 10,
  reg_param = 0.1,
  implicit_prefs = FALSE,
  alpha = 1,
  nonnegative = FALSE,
  max_iter = 10,
  num_user_blocks = 10,
  num_item_blocks = 10,
  checkpoint_interval = 10,
  cold_start_strategy = "nan",
  intermediate_storage_level = "MEMORY_AND_DISK",
  final_storage_level = "MEMORY_AND_DISK",
  uid = random_string("als_"),
  ...
)

ml_recommend(model, type = c("items", "users"), n = 1)

```

## Arguments

x                    A spark\_connection, ml\_pipeline, or a tbl\_spark.



|                            |  |
|----------------------------|--|
| formula                    | Used when x is a <code>tbl_spark</code> . R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details. The ALS model requires a specific formula format, please use <code>rating_col ~ user_col + item_col</code> .   |
| rating_col                 | Column name for ratings. Default: "rating"   |
| user_col                   | Column name for user ids. Ids must be integers. Other numeric types are supported for this column, but will be cast to integers as long as they fall within the integer value range. Default: "user"   |
| item_col                   | Column name for item ids. Ids must be integers. Other numeric types are supported for this column, but will be cast to integers as long as they fall within the integer value range. Default: "item"   |
| rank                       | Rank of the matrix factorization (positive). Default: 10   |
| reg_param                  | Regularization parameter.  |
| implicit_prefs             | Whether to use implicit preference. Default: FALSE.  |
| alpha                      | Alpha parameter in the implicit preference formulation (nonnegative).  |
| nonnegative                | Whether to apply nonnegativity constraints. Default: FALSE.  |
| max_iter                   | Maximum number of iterations.  |
| num_user_blocks            | Number of user blocks (positive). Default: 10  |
| num_item_blocks            | Number of item blocks (positive). Default: 10  |
| checkpoint_interval        | Set checkpoint interval ( $\geq 1$ ) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations, defaults to 10.   |
| cold_start_strategy        | (Spark 2.2.0+) Strategy for dealing with unknown or new users/items at prediction time. This may be useful in cross-validation or production scenarios, for handling user/item ids the model has not seen in the training data. Supported values: - "nan": predicted value for unknown ids will be NaN. - "drop": rows in the input DataFrame containing unknown ids will be dropped from the output DataFrame containing predictions. Default: "nan". |
| intermediate_storage_level | (Spark 2.0.0+) <code>StorageLevel</code> for intermediate datasets. Pass in a string representation of <code>StorageLevel</code> . Cannot be "NONE". Default: "MEMORY_AND_DISK".   |
| final_storage_level        | (Spark 2.0.0+) <code>StorageLevel</code> for ALS model factors. Pass in a string representation of <code>StorageLevel</code> . Default: "MEMORY_AND_DISK".   |
| uid                        | A character string used to uniquely identify the ML estimator.   |
| ...                        | Optional arguments; currently unused.  |
| model                      | An ALS model object  |
| type                       | What to recommend, one of <code>items</code> or <code>users</code>   |
| n                          | Maximum number of recommendations to return  |

## Details

ml\_recommend() returns the top n users/items recommended for each item/user, for all items/users. The output has been transformed (exploded and separated) from the default Spark outputs to be more user friendly.

## Value

ALS attempts to estimate the ratings matrix  $R$  as the product of two lower-rank matrices,  $X$  and  $Y$ , i.e.  $X * Y^t = R$ . Typically these approximations are called 'factor' matrices. The general approach is iterative. During each iteration, one of the factor matrices is held constant, while the other is solved for using least squares. The newly-solved factor matrix is then held constant while solving for the other factor matrix.

This is a blocked implementation of the ALS factorization algorithm that groups the two sets of factors (referred to as "users" and "products") into blocks and reduces communication by only sending one copy of each user vector to each product block on each iteration, and only for the product blocks that need that user's feature vector. This is achieved by pre-computing some information about the ratings matrix to determine the "out-links" of each user (which blocks of products it will contribute to) and "in-link" information for each product (which of the feature vectors it receives from each user block it will depend on). This allows us to send only an array of feature vectors between each user block and product block, and have the product block find the users' ratings and update the products based on these messages.

For implicit preference data, the algorithm used is based on "Collaborative Filtering for Implicit Feedback Datasets", available at [doi:10.1109/ICDM.2008.22](https://doi.org/10.1109/ICDM.2008.22), adapted for the blocked approach used here.

Essentially instead of finding the low-rank approximations to the rating matrix  $R$ , this finds the approximations for a preference matrix  $P$  where the elements of  $P$  are 1 if  $r$  is greater than 0 and 0 if  $r$  is less than or equal to 0. The ratings then act as 'confidence' values related to strength of indicated user preferences rather than explicit ratings given to items.

The object returned depends on the class of  $x$ .

- spark\_connection: When  $x$  is a spark\_connection, the function returns an instance of a ml\_als recommender object, which is an Estimator.
- ml\_pipeline: When  $x$  is a ml\_pipeline, the function returns a ml\_pipeline with the recommender appended to the pipeline.
- tbl\_spark: When  $x$  is a tbl\_spark, a recommender estimator is constructed then immediately fit with the input tbl\_spark, returning a recommendation model, i.e. ml\_als\_model.

## Examples

```
## Not run:

library(sparklyr)
sc <- spark_connect(master = "local")

movies <- data.frame(
  user = c(1, 2, 0, 1, 2, 0),
  item = c(1, 1, 1, 2, 2, 0),
```

```

    rating = c(3, 1, 2, 4, 5, 4)
  )
  movies_tbl <- sdf_copy_to(sc, movies)

  model <- ml_als(movies_tbl, rating ~ user + item)

  ml_predict(model, movies_tbl)

  ml_recommend(model, type = "item", 1)

  ## End(Not run)

```

---

ml\_als\_tidiers

*Tidying methods for Spark ML ALS*


---

### Description

These methods summarize the results of Spark ML models into tidy forms.

### Usage

```

## S3 method for class 'ml_model_als'
tidy(x, ...)

## S3 method for class 'ml_model_als'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_als'
glance(x, ...)

```

### Arguments

|         |  |
|---------|--|
| x       | a Spark ML model.                              |
| ...     | extra arguments (not used.)                    |
| newdata | a tbl_spark of new data to use for prediction. |

---

ml\_bisecting\_kmeans

*Spark ML – Bisecting K-Means Clustering*


---

### Description

A bisecting k-means algorithm based on the paper "A comparison of document clustering techniques" by Steinbach, Karypis, and Kumar, with modification to fit Spark. The algorithm starts from a single cluster that contains all points. Iteratively it finds divisible clusters on the bottom level and bisects each of them using k-means, until there are k leaf clusters in total or no leaf clusters are divisible. The bisecting steps of clusters on the same level are grouped together to increase parallelism. If bisecting all divisible clusters on the bottom level would result more than k leaf clusters, larger clusters get higher priority.

**Usage**

```
ml_bisecting_kmeans(
  x,
  formula = NULL,
  k = 4,
  max_iter = 20,
  seed = NULL,
  min_divisible_cluster_size = 1,
  features_col = "features",
  prediction_col = "prediction",
  uid = random_string("bisecting_bisecting_kmeans_"),
  ...
)
```

**Arguments**

|                            |   |
|----------------------------|---|
| x                          | A spark_connection, ml_pipeline, or a tbl_spark.  |
| formula                    | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details.   |
| k                          | The number of clusters to create  |
| max_iter                   | The maximum number of iterations to use.  |
| seed                       | A random seed. Set this value if you need your results to be reproducible across repeated calls.  |
| min_divisible_cluster_size | The minimum number of points (if greater than or equal to 1.0) or the minimum proportion of points (if less than 1.0) of a divisible cluster (default: 1.0).  |
| features_col               | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| prediction_col             | Prediction column name.   |
| uid                        | A character string used to uniquely identify the ML estimator.  |
| ...                        | Optional arguments, see Details. #' @return The object returned depends on the class of x. If it is a spark_connection, the function returns a ml_estimator object. If it is a ml_pipeline, it will return a pipeline with the predictor appended to it. If a tbl_spark, it will return a tbl_spark with the predictions added to it. |

**Examples**

```
## Not run:
library(dplyr)

sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)
```

```
iris_tbl %>%
  select(-Species) %>%
  ml_bisecting_kmeans(k = 4, Species ~ .)

## End(Not run)
```

---

ml\_chisquare\_test      *Chi-square hypothesis testing for categorical data.*

---

### Description

Conduct Pearson's independence test for every feature against the label. For each feature, the (feature, label) pairs are converted into a contingency matrix for which the Chi-squared statistic is computed. All label and feature values must be categorical.

### Usage

```
ml_chisquare_test(x, features, label)
```

### Arguments

|          |  |
|----------|--|
| x        | A tbl_spark.   |
| features | The name(s) of the feature columns. This can also be the name of a single vector column created using <code>ft_vector_assembler()</code> . |
| label    | The name of the label column.  |

### Value

A data frame with one row for each (feature, label) pair with p-values, degrees of freedom, and test statistics.

### Examples

```
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

features <- c("Petal_Width", "Petal_Length", "Sepal_Length", "Sepal_Width")

ml_chisquare_test(iris_tbl, features = features, label = "Species")

## End(Not run)
```

---

ml\_clustering\_evaluator

*Spark ML - Clustering Evaluator*


---

## Description

Evaluator for clustering results. The metric computes the Silhouette measure using the squared Euclidean distance. The Silhouette is a measure for the validation of the consistency within clusters. It ranges between 1 and -1, where a value close to 1 means that the points in a cluster are close to the other points in the same cluster and far from the points of the other clusters.

## Usage

```
ml_clustering_evaluator(
  x,
  features_col = "features",
  prediction_col = "prediction",
  metric_name = "silhouette",
  uid = random_string("clustering_evaluator_"),
  ...
)
```

## Arguments

|                |   |
|----------------|---|
| x              | A spark_connection object or a tbl_spark containing label and prediction columns. The latter should be the output of <code>sdf_predict</code> . |
| features_col   | Name of features column.  |
| prediction_col | Name of the prediction column.  |
| metric_name    | The performance metric. Currently supports "silhouette".  |
| uid            | A character string used to uniquely identify the ML estimator.  |
| ...            | Optional arguments; currently unused.   |

## Value

The calculated performance metric

## Examples

```
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

iris_training <- partitions$training
```

```
iris_test <- partitions$test

formula <- Species ~ .

# Train the models
kmeans_model <- ml_kmeans(iris_training, formula = formula)
b_kmeans_model <- ml_bisecting_kmeans(iris_training, formula = formula)
gmm_model <- ml_gaussian_mixture(iris_training, formula = formula)

# Predict
pred_kmeans <- ml_predict(kmeans_model, iris_test)
pred_b_kmeans <- ml_predict(b_kmeans_model, iris_test)
pred_gmm <- ml_predict(gmm_model, iris_test)

# Evaluate
ml_clustering_evaluator(pred_kmeans)
ml_clustering_evaluator(pred_b_kmeans)
ml_clustering_evaluator(pred_gmm)

## End(Not run)
```

---

ml\_corr

*Compute correlation matrix*

---

## Description

Compute correlation matrix

## Usage

```
ml_corr(x, columns = NULL, method = c("pearson", "spearman"))
```

## Arguments

|         |  |
|---------|--|
| x       | A tbl_spark.   |
| columns | The names of the columns to calculate correlations of. If only one column is specified, it must be a vector column (for example, assembled using <code>ft_vector_assembler()</code> ). |
| method  | The method to use, either "pearson" or "spearman".   |

## Value

A correlation matrix organized as a data frame.

## Examples

```
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

features <- c("Petal_Width", "Petal_Length", "Sepal_Length", "Sepal_Width")

ml_corr(iris_tbl, columns = features, method = "pearson")

## End(Not run)
```

---

ml\_decision\_tree\_classifier

*Spark ML – Decision Trees*

---

## Description

Perform classification and regression using decision trees.

## Usage

```
ml_decision_tree_classifier(
  x,
  formula = NULL,
  max_depth = 5,
  max_bins = 32,
  min_instances_per_node = 1,
  min_info_gain = 0,
  impurity = "gini",
  seed = NULL,
  thresholds = NULL,
  cache_node_ids = FALSE,
  checkpoint_interval = 10,
  max_memory_in_mb = 256,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  probability_col = "probability",
  raw_prediction_col = "rawPrediction",
  uid = random_string("decision_tree_classifier_"),
  ...
)

ml_decision_tree(
  x,
  formula = NULL,
  type = c("auto", "regression", "classification"),
```



```

    features_col = "features",
    label_col = "label",
    prediction_col = "prediction",
    variance_col = NULL,
    probability_col = "probability",
    raw_prediction_col = "rawPrediction",
    checkpoint_interval = 10L,
    impurity = "auto",
    max_bins = 32L,
    max_depth = 5L,
    min_info_gain = 0,
    min_instances_per_node = 1L,
    seed = NULL,
    thresholds = NULL,
    cache_node_ids = FALSE,
    max_memory_in_mb = 256L,
    uid = random_string("decision_tree_"),
    response = NULL,
    features = NULL,
    ...
)

ml_decision_tree_regressor(
  x,
  formula = NULL,
  max_depth = 5,
  max_bins = 32,
  min_instances_per_node = 1,
  min_info_gain = 0,
  impurity = "variance",
  seed = NULL,
  cache_node_ids = FALSE,
  checkpoint_interval = 10,
  max_memory_in_mb = 256,
  variance_col = NULL,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  uid = random_string("decision_tree_regressor_"),
  ...
)

```

### Arguments

|         |   |
|---------|---|
| x       | A spark_connection, ml_pipeline, or a tbl_spark.  |
| formula | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details. |

|                                     |   |
|-------------------------------------|---|
| <code>max_depth</code>              | Maximum depth of the tree ( $\geq 0$ ); that is, the maximum number of nodes separating any leaves from the root of the tree.   |
| <code>max_bins</code>               | The maximum number of bins used for discretizing continuous features and for choosing how to split on features at each node. More bins give higher granularity.   |
| <code>min_instances_per_node</code> | Minimum number of instances each child must have after split.   |
| <code>min_info_gain</code>          | Minimum information gain for a split to be considered at a tree node. Should be $\geq 0$ , defaults to 0.   |
| <code>impurity</code>               | Criterion used for information gain calculation. Supported: "entropy" and "gini" (default) for classification and "variance" (default) for regression. For <code>ml_decision_tree</code> , setting "auto" will default to the appropriate criterion based on model type.  |
| <code>seed</code>                   | Seed for random numbers.  |
| <code>thresholds</code>             | Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values $> 0$ excepting that at most one value may be 0. The class with largest value $p/t$ is predicted, where $p$ is the original probability of that class and $t$ is the class's threshold. |
| <code>cache_node_ids</code>         | If FALSE, the algorithm will pass trees to executors to match instances with nodes. If TRUE, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Defaults to FALSE.   |
| <code>checkpoint_interval</code>    | Set checkpoint interval ( $\geq 1$ ) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations, defaults to 10.  |
| <code>max_memory_in_mb</code>       | Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size. Defaults to 256.   |
| <code>features_col</code>           | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <code>ft_r_formula</code> .   |
| <code>label_col</code>              | Label column name. The column should be a numeric column. Usually this column is output by <code>ft_r_formula</code> .  |
| <code>prediction_col</code>         | Prediction column name.   |
| <code>probability_col</code>        | Column name for predicted class conditional probabilities.  |
| <code>raw_prediction_col</code>     | Raw prediction (a.k.a. confidence) column name.   |
| <code>uid</code>                    | A character string used to uniquely identify the ML estimator.  |
| <code>...</code>                    | Optional arguments; see Details.  |
| <code>type</code>                   | The type of model to fit. "regression" treats the response as a continuous variable, while "classification" treats the response as a categorical variable. When "auto" is used, the model type is inferred based on the response variable type – if it is a numeric type, then regression is used; classification otherwise.                            |

|              |  |
|--------------|--|
| variance_col | (Optional) Column name for the biased sample variance of prediction.             |
| response     | (Deprecated) The name of the response column (as a length-one character vector.) |
| features     | (Deprecated) The name of features (terms) to use for the model fit.              |

## Details

ml\_decision\_tree is a wrapper around ml\_decision\_tree\_regressor.tbl\_spark and ml\_decision\_tree\_classifier and calls the appropriate method based on model type.

## Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the predictor appended to it. If a tbl\_spark, it will return a tbl\_spark with the predictions added to it.

## See Also

Other ml algorithms: [ml\\_aft\\_survival\\_regression\(\)](#), [ml\\_gbt\\_classifier\(\)](#), [ml\\_generalized\\_linear\\_regression\(\)](#), [ml\\_isotonic\\_regression\(\)](#), [ml\\_linear\\_regression\(\)](#), [ml\\_linear\\_svc\(\)](#), [ml\\_logistic\\_regression\(\)](#), [ml\\_multilayer\\_perceptron\\_classifier\(\)](#), [ml\\_naive\\_bayes\(\)](#), [ml\\_one\\_vs\\_rest\(\)](#), [ml\\_random\\_forest\\_classifier\(\)](#)

## Examples

```
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

iris_training <- partitions$training
iris_test <- partitions$test

dt_model <- iris_training %>%
  ml_decision_tree(Species ~ .)

pred <- ml_predict(dt_model, iris_test)

ml_multiclass_classification_evaluator(pred)

## End(Not run)
```

---

ml\_default\_stop\_words *Default stop words*

---

### Description

Loads the default stop words for the given language.

### Usage

```
ml_default_stop_words(  
    sc,  
    language = c("english", "danish", "dutch", "finnish", "french", "german", "hungarian",  
                "italian", "norwegian", "portuguese", "russian", "spanish", "swedish", "turkish"),  
    ...  
)
```

### Arguments

|          |                                       |
|----------|---------------------------------------|
| sc       | A spark_connection                    |
| language | A character string.                   |
| ...      | Optional arguments; currently unused. |

### Details

Supported languages: danish, dutch, english, finnish, french, german, hungarian, italian, norwegian, portuguese, russian, spanish, swedish, turkish. Defaults to English. See <https://anoncv.s.postgresql.org/cvsweb.cgi/pgsql/src/backend/snowball/stopwords/> for more details

### Value

A list of stop words.

### See Also

[ft\\_stop\\_words\\_remover](#)

---

`ml_evaluate`*Evaluate the Model on a Validation Set*

---

**Description**

Compute performance metrics.

**Usage**

```
ml_evaluate(x, dataset)

## S3 method for class 'ml_model_logistic_regression'
ml_evaluate(x, dataset)

## S3 method for class 'ml_logistic_regression_model'
ml_evaluate(x, dataset)

## S3 method for class 'ml_model_linear_regression'
ml_evaluate(x, dataset)

## S3 method for class 'ml_linear_regression_model'
ml_evaluate(x, dataset)

## S3 method for class 'ml_model_generalized_linear_regression'
ml_evaluate(x, dataset)

## S3 method for class 'ml_generalized_linear_regression_model'
ml_evaluate(x, dataset)

## S3 method for class 'ml_model_clustering'
ml_evaluate(x, dataset)

## S3 method for class 'ml_model_classification'
ml_evaluate(x, dataset)

## S3 method for class 'ml_evaluator'
ml_evaluate(x, dataset)
```

**Arguments**

|                      |  |
|----------------------|--|
| <code>x</code>       | An ML model object or an evaluator object. |
| <code>dataset</code> | The dataset to be validate the model on.   |

**Examples**

```
## Not run:
sc <- spark_connect(master = "local")
```

```

iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

ml_gaussian_mixture(iris_tbl, Species ~ .) %>%
  ml_evaluate(iris_tbl)

ml_kmeans(iris_tbl, Species ~ .) %>%
  ml_evaluate(iris_tbl)

ml_bisecting_kmeans(iris_tbl, Species ~ .) %>%
  ml_evaluate(iris_tbl)

## End(Not run)

```

---

ml\_evaluator

*Spark ML - Evaluators*


---

### Description

A set of functions to calculate performance metrics for prediction models. Also see the Spark ML Documentation <https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.ml.evaluation.package>

### Usage

```

ml_binary_classification_evaluator(
  x,
  label_col = "label",
  raw_prediction_col = "rawPrediction",
  metric_name = "areaUnderROC",
  uid = random_string("binary_classification_evaluator_"),
  ...
)

ml_binary_classification_eval(
  x,
  label_col = "label",
  prediction_col = "prediction",
  metric_name = "areaUnderROC"
)

ml_multiclass_classification_evaluator(
  x,
  label_col = "label",
  prediction_col = "prediction",
  metric_name = "f1",
  uid = random_string("multiclass_classification_evaluator_"),
  ...
)

```

```

ml_classification_eval(
  x,
  label_col = "label",
  prediction_col = "prediction",
  metric_name = "f1"
)

ml_regression_evaluator(
  x,
  label_col = "label",
  prediction_col = "prediction",
  metric_name = "rmse",
  uid = random_string("regression_evaluator_"),
  ...
)

```

### Arguments

|                    |  |
|--------------------|--|
| x                  | A spark_connection object or a tbl_spark containing label and prediction columns. The latter should be the output of <a href="#">sdf_predict</a> . |
| label_col          | Name of column string specifying which column contains the true labels or values.  |
| raw_prediction_col | Raw prediction (a.k.a. confidence) column name.  |
| metric_name        | The performance metric. See details.   |
| uid                | A character string used to uniquely identify the ML estimator.   |
| ...                | Optional arguments; currently unused.  |
| prediction_col     | Name of the column that contains the predicted label or value NOT the scored probability. Column should be of type Double.                         |

### Details

The following metrics are supported

- Binary Classification: `areaUnderROC` (default) or `areaUnderPR` (not available in Spark 2.X.)
- Multiclass Classification: `f1` (default), `precision`, `recall`, `weightedPrecision`, `weightedRecall` or `accuracy`; for Spark 2.X: `f1` (default), `weightedPrecision`, `weightedRecall` or `accuracy`.
- Regression: `rmse` (root mean squared error, default), `mse` (mean squared error), `r2`, or `mae` (mean absolute error.)

`ml_binary_classification_eval()` is an alias for `ml_binary_classification_evaluator()` for backwards compatibility.

`ml_classification_eval()` is an alias for `ml_multiclass_classification_evaluator()` for backwards compatibility.

**Value**

The calculated performance metric

**Examples**

```
## Not run:
sc <- spark_connect(master = "local")
mtcars_tbl <- sdf_copy_to(sc, mtcars, name = "mtcars_tbl", overwrite = TRUE)

partitions <- mtcars_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

mtcars_training <- partitions$training
mtcars_test <- partitions$test

# for multiclass classification
rf_model <- mtcars_training %>%
  ml_random_forest(cyl ~ ., type = "classification")

pred <- ml_predict(rf_model, mtcars_test)

ml_multiclass_classification_evaluator(pred)

# for regression
rf_model <- mtcars_training %>%
  ml_random_forest(cyl ~ ., type = "regression")

pred <- ml_predict(rf_model, mtcars_test)

ml_regression_evaluator(pred, label_col = "cyl")

# for binary classification
rf_model <- mtcars_training %>%
  ml_random_forest(am ~ gear + carb, type = "classification")

pred <- ml_predict(rf_model, mtcars_test)

ml_binary_classification_evaluator(pred)

## End(Not run)
```

---

ml\_feature\_importances

*Spark ML - Feature Importance for Tree Models*

---

**Description**

Spark ML - Feature Importance for Tree Models



**Usage**

```
ml_feature_importances(model, ...)

ml_tree_feature_importance(model, ...)
```

**Arguments**

|       |                                       |
|-------|---------------------------------------|
| model | A decision tree-based model.          |
| ...   | Optional arguments; currently unused. |

**Value**

For `ml_model`, a sorted data frame with feature labels and their relative importance. For `ml_prediction_model`, a vector of relative importances.

---

|             |   |
|-------------|---|
| ml_fpgrowth | <i>Frequent Pattern Mining – FPGrowth</i> |
|-------------|---|

---

**Description**

A parallel FP-growth algorithm to mine frequent itemsets.

**Usage**

```
ml_fpgrowth(
  x,
  items_col = "items",
  min_confidence = 0.8,
  min_support = 0.3,
  prediction_col = "prediction",
  uid = random_string("fpgrowth"),
  ...
)

ml_association_rules(model)

ml_freq_itemsets(model)
```

**Arguments**

|                |   |
|----------------|---|
| x              | A spark_connection, ml_pipeline, or a tbl_spark.  |
| items_col      | Items column name. Default: "items"   |
| min_confidence | Minimal confidence for generating Association Rule. min_confidence will not affect the mining for frequent itemsets, but will affect the association rules generation. Default: 0.8 |

|                |   |
|----------------|---|
| min_support    | Minimal support level of the frequent pattern. [0.0, 1.0]. Any pattern that appears more than (min_support * size-of-the-dataset) times will be output in the frequent itemsets. Default: 0.3 |
| prediction_col | Prediction column name.   |
| uid            | A character string used to uniquely identify the ML estimator.  |
| ...            | Optional arguments; currently unused.   |
| model          | A fitted FPGrowth model returned by ml_fpgrowth()   |

---

ml\_gaussian\_mixture     *Spark ML – Gaussian Mixture clustering.*

---

### Description

This class performs expectation maximization for multivariate Gaussian Mixture Models (GMMs). A GMM represents a composite distribution of independent Gaussian distributions with associated "mixing" weights specifying each's contribution to the composite. Given a set of sample points, this class will maximize the log-likelihood for a mixture of k Gaussians, iterating until the log-likelihood changes by less than tol, or until it has reached the max number of iterations. While this process is generally guaranteed to converge, it is not guaranteed to find a global optimum.

### Usage

```
ml_gaussian_mixture(
  x,
  formula = NULL,
  k = 2,
  max_iter = 100,
  tol = 0.01,
  seed = NULL,
  features_col = "features",
  prediction_col = "prediction",
  probability_col = "probability",
  uid = random_string("gaussian_mixture_"),
  ...
)
```

### Arguments

|          |   |
|----------|---|
| x        | A spark_connection, ml_pipeline, or a tbl_spark.  |
| formula  | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details. |
| k        | The number of clusters to create  |
| max_iter | The maximum number of iterations to use.  |
| tol      | Param for the convergence tolerance for iterative algorithms.   |

|                 |   |
|-----------------|---|
| seed            | A random seed. Set this value if you need your results to be reproducible across repeated calls.  |
| features_col    | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| prediction_col  | Prediction column name.   |
| probability_col | Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.  |
| uid             | A character string used to uniquely identify the ML estimator.  |
| ...             | Optional arguments, see Details. #' @return The object returned depends on the class of x. If it is a spark_connection, the function returns a ml_estimator object. If it is a ml_pipeline, it will return a pipeline with the predictor appended to it. If a tbl_spark, it will return a tbl_spark with the predictions added to it. |

### Examples

```
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

gmm_model <- ml_gaussian_mixture(iris_tbl, Species ~ .)
pred <- sdf_predict(iris_tbl, gmm_model)
ml_clustering_evaluator(pred)

## End(Not run)
```

---

ml\_gbt\_classifier      *Spark ML – Gradient Boosted Trees*

---

### Description

Perform binary classification and regression using gradient boosted trees. Multiclass classification is not supported yet.

### Usage

```
ml_gbt_classifier(
  x,
  formula = NULL,
  max_iter = 20,
  max_depth = 5,
  step_size = 0.1,
  subsampling_rate = 1,
```

```
feature_subset_strategy = "auto",
min_instances_per_node = 1L,
max_bins = 32,
min_info_gain = 0,
loss_type = "logistic",
seed = NULL,
thresholds = NULL,
checkpoint_interval = 10,
cache_node_ids = FALSE,
max_memory_in_mb = 256,
features_col = "features",
label_col = "label",
prediction_col = "prediction",
probability_col = "probability",
raw_prediction_col = "rawPrediction",
uid = random_string("gbt_classifier_"),
...
)

ml_gradient_boosted_trees(
  x,
  formula = NULL,
  type = c("auto", "regression", "classification"),
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  probability_col = "probability",
  raw_prediction_col = "rawPrediction",
  checkpoint_interval = 10,
  loss_type = c("auto", "logistic", "squared", "absolute"),
  max_bins = 32,
  max_depth = 5,
  max_iter = 20L,
  min_info_gain = 0,
  min_instances_per_node = 1,
  step_size = 0.1,
  subsampling_rate = 1,
  feature_subset_strategy = "auto",
  seed = NULL,
  thresholds = NULL,
  cache_node_ids = FALSE,
  max_memory_in_mb = 256,
  uid = random_string("gradient_boosted_trees_"),
  response = NULL,
  features = NULL,
  ...
)
```

```

ml_gbt_regressor(
  x,
  formula = NULL,
  max_iter = 20,
  max_depth = 5,
  step_size = 0.1,
  subsampling_rate = 1,
  feature_subset_strategy = "auto",
  min_instances_per_node = 1,
  max_bins = 32,
  min_info_gain = 0,
  loss_type = "squared",
  seed = NULL,
  checkpoint_interval = 10,
  cache_node_ids = FALSE,
  max_memory_in_mb = 256,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  uid = random_string("gbt_regressor_"),
  ...
)

```

### Arguments

|                         |   |
|-------------------------|---|
| x                       | A spark_connection, ml_pipeline, or a tbl_spark.  |
| formula                 | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details. |
| max_iter                | Maximum number of iterations.   |
| max_depth               | Maximum depth of the tree ( $\geq 0$ ); that is, the maximum number of nodes separating any leaves from the root of the tree.   |
| step_size               | Step size (a.k.a. learning rate) in interval (0, 1] for shrinking the contribution of each estimator. (default = 0.1)   |
| subsampling_rate        | Fraction of the training data used for learning each decision tree, in range (0, 1]. (default = 1.0)  |
| feature_subset_strategy | The number of features to consider for splits at each tree node. See details for options.   |
| min_instances_per_node  | Minimum number of instances each child must have after split.   |
| max_bins                | The maximum number of bins used for discretizing continuous features and for choosing how to split on features at each node. More bins give higher granularity.                       |
| min_info_gain           | Minimum information gain for a split to be considered at a tree node. Should be $\geq 0$ , defaults to 0.   |

|                     |   |
|---------------------|---|
| loss_type           | Loss function which GBT tries to minimize. Supported: "squared" (L2) and "absolute" (L1) (default = squared) for regression and "logistic" (default) for classification. For ml_gradient_boosted_trees, setting "auto" will default to the appropriate loss type based on model type.   |
| seed                | Seed for random numbers.  |
| thresholds          | Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value $p/t$ is predicted, where $p$ is the original probability of that class and $t$ is the class's threshold. |
| checkpoint_interval | Set checkpoint interval ( $\geq 1$ ) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations, defaults to 10.  |
| cache_node_ids      | If FALSE, the algorithm will pass trees to executors to match instances with nodes. If TRUE, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Defaults to FALSE.   |
| max_memory_in_mb    | Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size. Defaults to 256.   |
| features_col        | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| label_col           | Label column name. The column should be a numeric column. Usually this column is output by <a href="#">ft_r_formula</a> .   |
| prediction_col      | Prediction column name.   |
| probability_col     | Column name for predicted class conditional probabilities.  |
| raw_prediction_col  | Raw prediction (a.k.a. confidence) column name.   |
| uid                 | A character string used to uniquely identify the ML estimator.  |
| ...                 | Optional arguments; see Details.  |
| type                | The type of model to fit. "regression" treats the response as a continuous variable, while "classification" treats the response as a categorical variable. When "auto" is used, the model type is inferred based on the response variable type – if it is a numeric type, then regression is used; classification otherwise.                          |
| response            | (Deprecated) The name of the response column (as a length-one character vector.)  |
| features            | (Deprecated) The name of features (terms) to use for the model fit.   |

## Details

The supported options for feature\_subset\_strategy are

- "auto": Choose automatically for task: If num\_trees == 1, set to "all". If num\_trees > 1 (forest), set to "sqrt" for classification and to "onethird" for regression.

- "all": use all features
- "onethird": use 1/3 of the features
- "sqrt": use use sqrt(number of features)
- "log2": use log2(number of features)
- "n": when n is in the range (0, 1.0], use n \* number of features. When n is in the range (1, number of features), use n features. (default = "auto")

ml\_gradient\_boosted\_trees is a wrapper around ml\_gbt\_regressor.tbl\_spark and ml\_gbt\_classifier.tbl\_spark and calls the appropriate method based on model type.

### Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the predictor appended to it. If a tbl\_spark, it will return a tbl\_spark with the predictions added to it.

### See Also

Other ml algorithms: [ml\\_aft\\_survival\\_regression\(\)](#), [ml\\_decision\\_tree\\_classifier\(\)](#), [ml\\_generalized\\_linear\\_regression\(\)](#), [ml\\_isotonic\\_regression\(\)](#), [ml\\_linear\\_regression\(\)](#), [ml\\_linear\\_svc\(\)](#), [ml\\_logistic\\_regression\(\)](#), [ml\\_multilayer\\_perceptron\\_classifier\(\)](#), [ml\\_naive\\_bayes\(\)](#), [ml\\_one\\_vs\\_rest\(\)](#), [ml\\_random\\_forest\\_classifier\(\)](#)

### Examples

```
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

iris_training <- partitions$training
iris_test <- partitions$test

gbt_model <- iris_training %>%
  ml_gradient_boosted_trees(Sepal_Length ~ Petal_Length + Petal_Width)

pred <- ml_predict(gbt_model, iris_test)

ml_regression_evaluator(pred, label_col = "Sepal_Length")

## End(Not run)
```

---

 ml\_generalized\_linear\_regression

*Spark ML – Generalized Linear Regression*


---

## Description

Perform regression using Generalized Linear Model (GLM).

## Usage

```
ml_generalized_linear_regression(
  x,
  formula = NULL,
  family = "gaussian",
  link = NULL,
  fit_intercept = TRUE,
  offset_col = NULL,
  link_power = NULL,
  link_prediction_col = NULL,
  reg_param = 0,
  max_iter = 25,
  weight_col = NULL,
  solver = "irls",
  tol = 1e-06,
  variance_power = 0,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  uid = random_string("generalized_linear_regression_"),
  ...
)
```

## Arguments

|               |  |
|---------------|--|
| x             | A spark_connection, ml_pipeline, or a tbl_spark.   |
| formula       | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details.        |
| family        | Name of family which is a description of the error distribution to be used in the model. Supported options: "gaussian", "binomial", "poisson", "gamma" and "tweedie". Default is "gaussian". |
| link          | Name of link function which provides the relationship between the linear predictor and the mean of the distribution function. See for supported link functions.                              |
| fit_intercept | Boolean; should the model be fit with an intercept term?   |
| offset_col    | Offset column name. If this is not set, we treat all instance offsets as 0.0. The feature specified as offset has a constant coefficient of 1.0.   |



|                     |  |
|---------------------|--|
| link_power          | Index in the power link function. Only applicable to the Tweedie family. Note that link power 0, 1, -1 or 0.5 corresponds to the Log, Identity, Inverse or Sqrt link, respectively. When not set, this value defaults to 1 - variancePower, which matches the R "statmod" package.   |
| link_prediction_col | Link prediction (linear predictor) column name. Default is not set, which means we do not output link prediction.  |
| reg_param           | Regularization parameter (aka lambda)  |
| max_iter            | The maximum number of iterations to use.   |
| weight_col          | The name of the column to use as weights for the model fit.  |
| solver              | Solver algorithm for optimization.   |
| tol                 | Param for the convergence tolerance for iterative algorithms.  |
| variance_power      | Power in the variance function of the Tweedie distribution which provides the relationship between the variance and mean of the distribution. Only applicable to the Tweedie family. (see <a href="#">Tweedie Distribution (Wikipedia)</a> ) Supported values: 0 and [1, Inf). Note that variance power 0, 1, or 2 corresponds to the Gaussian, Poisson or Gamma family, respectively. |
| features_col        | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> .   |
| label_col           | Label column name. The column should be a numeric column. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| prediction_col      | Prediction column name.  |
| uid                 | A character string used to uniquely identify the ML estimator.   |
| ...                 | Optional arguments; see Details.   |

## Details

Valid link functions for each family is listed below. The first link function of each family is the default one.

- gaussian: "identity", "log", "inverse"
- binomial: "logit", "probit", "loglog"
- poisson: "log", "identity", "sqrt"
- gamma: "inverse", "identity", "log"
- tweedie: power link function specified through link\_power. The default link power in the tweedie family is 1 - variance\_power.

## Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the predictor appended to it. If a tbl\_spark, it will return a tbl\_spark with the predictions added to it.

**See Also**

Other ml algorithms: `ml_aft_survival_regression()`, `ml_decision_tree_classifier()`, `ml_gbt_classifier()`, `ml_isotonic_regression()`, `ml_linear_regression()`, `ml_linear_svc()`, `ml_logistic_regression()`, `ml_multilayer_perceptron_classifier()`, `ml_naive_bayes()`, `ml_one_vs_rest()`, `ml_random_forest_classifier()`

**Examples**

```
## Not run:
library(sparklyr)

sc <- spark_connect(master = "local")
mtcars_tbl <- sdf_copy_to(sc, mtcars, name = "mtcars_tbl", overwrite = TRUE)

partitions <- mtcars_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

mtcars_training <- partitions$training
mtcars_test <- partitions$test

# Specify the grid
family <- c("gaussian", "gamma", "poisson")
link <- c("identity", "log")
family_link <- expand_grid(family = family, link = link, stringsAsFactors = FALSE)
family_link <- data.frame(family_link, rmse = 0)

# Train the models
for (i in seq_len(nrow(family_link))) {
  glm_model <- mtcars_training %>%
    ml_generalized_linear_regression(mpg ~ .,
      family = family_link[i, 1],
      link = family_link[i, 2]
    )

  pred <- ml_predict(glm_model, mtcars_test)
  family_link[i, 3] <- ml_regression_evaluator(pred, label_col = "mpg")
}

family_link

## End(Not run)
```

**Description**

These methods summarize the results of Spark ML models into tidy forms.

**Usage**

```
## S3 method for class 'ml_model_generalized_linear_regression'
tidy(x, exponentiate = FALSE, ...)

## S3 method for class 'ml_model_linear_regression'
tidy(x, ...)

## S3 method for class 'ml_model_generalized_linear_regression'
augment(
  x,
  newdata = NULL,
  type.residuals = c("working", "deviance", "pearson", "response"),
  ...
)

## S3 method for class '`_ml_model_linear_regression`'
augment(
  x,
  new_data = NULL,
  type.residuals = c("working", "deviance", "pearson", "response"),
  ...
)

## S3 method for class 'ml_model_linear_regression'
augment(
  x,
  newdata = NULL,
  type.residuals = c("working", "deviance", "pearson", "response"),
  ...
)

## S3 method for class 'ml_model_generalized_linear_regression'
glance(x, ...)

## S3 method for class 'ml_model_linear_regression'
glance(x, ...)
```

**Arguments**

|                |   |
|----------------|---|
| x              | a Spark ML model.   |
| exponentiate   | For GLM, whether to exponentiate the coefficient estimates (typical for logistic regression.) |
| ...            | extra arguments (not used.)   |
| newdata        | a <code>tbl_spark</code> of new data to use for prediction.                                   |
| type.residuals | type of residuals, defaults to "working". Must be set to "working" when newdata is supplied.  |
| new_data       | a <code>tbl_spark</code> of new data to use for prediction.                                   |

**Details**

The residuals attached by `augment` are of type "working" by default, which is different from the default of "deviance" for `residuals()` or `sdf_residuals()`.

---

ml\_isotonic\_regression

*Spark ML – Isotonic Regression*

---

**Description**

Currently implemented using parallelized pool adjacent violators algorithm. Only univariate (single feature) algorithm supported.

**Usage**

```
ml_isotonic_regression(
  x,
  formula = NULL,
  feature_index = 0,
  isotonic = TRUE,
  weight_col = NULL,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  uid = random_string("isotonic_regression_"),
  ...
)
```

**Arguments**

|                            |  |
|----------------------------|--|
| <code>x</code>             | A <code>spark_connection</code> , <code>ml_pipeline</code> , or a <code>tbl_spark</code> .   |
| <code>formula</code>       | Used when <code>x</code> is a <code>tbl_spark</code> . R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details. |
| <code>feature_index</code> | Index of the feature if <code>features_col</code> is a vector column (default: 0), no effect otherwise.  |
| <code>isotonic</code>      | Whether the output sequence should be isotonic/increasing (true) or antitonic/decreasing (false). Default: true  |
| <code>weight_col</code>    | The name of the column to use as weights for the model fit.  |
| <code>features_col</code>  | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> .                             |
| <code>label_col</code>     | Label column name. The column should be a numeric column. Usually this column is output by <a href="#">ft_r_formula</a> .  |

prediction\_col Prediction column name.  
 uid A character string used to uniquely identify the ML estimator.  
 ... Optional arguments; see Details.

### Value

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the predictor appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the predictions added to it.

### See Also

Other ml algorithms: `ml_aft_survival_regression()`, `ml_decision_tree_classifier()`, `ml_gbt_classifier()`, `ml_generalized_linear_regression()`, `ml_linear_regression()`, `ml_linear_svc()`, `ml_logistic_regression()`, `ml_multilayer_perceptron_classifier()`, `ml_naive_bayes()`, `ml_one_vs_rest()`, `ml_random_forest_classifier()`

### Examples

```
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

iris_training <- partitions$training
iris_test <- partitions$test

iso_res <- iris_tbl %>%
  ml_isotonic_regression(Petal_Length ~ Petal_Width)

pred <- ml_predict(iso_res, iris_test)

pred

## End(Not run)
```

---

ml\_isotonic\_regression\_tidiers

*Tidying methods for Spark ML Isotonic Regression*

---

### Description

These methods summarize the results of Spark ML models into tidy forms.

**Usage**

```
## S3 method for class 'ml_model_isotonic_regression'
tidy(x, ...)

## S3 method for class 'ml_model_isotonic_regression'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_isotonic_regression'
glance(x, ...)
```

**Arguments**

|         |  |
|---------|--|
| x       | a Spark ML model.                              |
| ...     | extra arguments (not used.)                    |
| newdata | a tbl_spark of new data to use for prediction. |

---

ml\_kmeans

*Spark ML – K-Means Clustering*


---

**Description**

K-means clustering with support for k-means|| initialization proposed by Bahmani et al. Using 'ml\_kmeans()' with the formula interface requires Spark 2.0+.

**Usage**

```
ml_kmeans(
  x,
  formula = NULL,
  k = 2,
  max_iter = 20,
  tol = 1e-04,
  init_steps = 2,
  init_mode = "k-means||",
  seed = NULL,
  features_col = "features",
  prediction_col = "prediction",
  uid = random_string("kmeans_"),
  ...
)

ml_compute_cost(model, dataset)

ml_compute_silhouette_measure(
  model,
  dataset,
  distance_measure = c("squaredEuclidean", "cosine")
)
```

**Arguments**

|                  |   |
|------------------|---|
| x                | A spark_connection, ml_pipeline, or a tbl_spark.  |
| formula          | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details.   |
| k                | The number of clusters to create  |
| max_iter         | The maximum number of iterations to use.  |
| tol              | Param for the convergence tolerance for iterative algorithms.   |
| init_steps       | Number of steps for the k-meansll initialization mode. This is an advanced setting – the default of 2 is almost always enough. Must be > 0. Default: 2.   |
| init_mode        | Initialization algorithm. This can be either "random" to choose random points as initial cluster centers, or "k-meansll" to use a parallel variant of k-means++ (Bahmani et al., Scalable K-Means++, VLDB 2012). Default: k-meansll.  |
| seed             | A random seed. Set this value if you need your results to be reproducible across repeated calls.  |
| features_col     | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| prediction_col   | Prediction column name.   |
| uid              | A character string used to uniquely identify the ML estimator.  |
| ...              | Optional arguments, see Details. #' @return The object returned depends on the class of x. If it is a spark_connection, the function returns a ml_estimator object. If it is a ml_pipeline, it will return a pipeline with the predictor appended to it. If a tbl_spark, it will return a tbl_spark with the predictions added to it. |
| model            | A fitted K-means model returned by ml_kmeans()  |
| dataset          | Dataset on which to calculate K-means cost  |
| distance_measure | Distance measure to apply when computing the Silhouette measure.  |

**Value**

ml\_compute\_cost() returns the K-means cost (sum of squared distances of points to their nearest center) for the model on the given data.

ml\_compute\_silhouette\_measure() returns the Silhouette measure of the clustering on the given data.

**Examples**

```
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)
ml_kmeans(iris_tbl, Species ~ .)

## End(Not run)
```

---

`ml_kmeans_cluster_eval`*Evaluate a K-mean clustering*

---

**Description**

Evaluate a K-mean clustering

**Arguments**

|                      |   |
|----------------------|---|
| <code>model</code>   | A fitted K-means model returned by <code>ml_kmeans()</code> |
| <code>dataset</code> | Dataset on which to calculate K-means cost                  |

---

`ml_lda`*Spark ML – Latent Dirichlet Allocation*

---

**Description**

Latent Dirichlet Allocation (LDA), a topic model designed for text documents.

**Usage**

```
ml_lda(  
  x,  
  formula = NULL,  
  k = 10,  
  max_iter = 20,  
  doc_concentration = NULL,  
  topic_concentration = NULL,  
  subsampling_rate = 0.05,  
  optimizer = "online",  
  checkpoint_interval = 10,  
  keep_last_checkpoint = TRUE,  
  learning_decay = 0.51,  
  learning_offset = 1024,  
  optimize_doc_concentration = TRUE,  
  seed = NULL,  
  features_col = "features",  
  topic_distribution_col = "topicDistribution",  
  uid = random_string("lda"),  
  ...  
)  
  
ml_describe_topics(model, max_terms_per_topic = 10)
```



```
ml_log_likelihood(model, dataset)
```

```
ml_log_perplexity(model, dataset)
```

```
ml_topics_matrix(model)
```

## Arguments

|                      |   |
|----------------------|---|
| x                    | A spark_connection, ml_pipeline, or a tbl_spark.  |
| formula              | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details.   |
| k                    | The number of clusters to create  |
| max_iter             | The maximum number of iterations to use.  |
| doc_concentration    | Concentration parameter (commonly named "alpha") for the prior placed on documents' distributions over topics ("theta"). See details.   |
| topic_concentration  | Concentration parameter (commonly named "beta" or "eta") for the prior placed on topics' distributions over terms.  |
| subsampling_rate     | (For Online optimizer only) Fraction of the corpus to be sampled and used in each iteration of mini-batch gradient descent, in range (0, 1]. Note that this should be adjusted in synch with max_iter so the entire corpus is used. Specifically, set both so that maxIterations * miniBatchFraction greater than or equal to 1.                  |
| optimizer            | Optimizer or inference algorithm used to estimate the LDA model. Supported: "online" for Online Variational Bayes (default) and "em" for Expectation-Maximization.  |
| checkpoint_interval  | Set checkpoint interval ( $\geq 1$ ) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations, defaults to 10.  |
| keep_last_checkpoint | (Spark 2.0.0+) (For EM optimizer only) If using checkpointing, this indicates whether to keep the last checkpoint. If FALSE, then the checkpoint will be deleted. Deleting the checkpoint can cause failures if a data partition is lost, so set this bit with care. Note that checkpoints will be cleaned up via reference counting, regardless. |
| learning_decay       | (For Online optimizer only) Learning rate, set as an exponential decay rate. This should be between (0.5, 1.0] to guarantee asymptotic convergence. This is called "kappa" in the Online LDA paper (Hoffman et al., 2010). Default: 0.51, based on Hoffman et al.   |
| learning_offset      | (For Online optimizer only) A (positive) learning parameter that downweights early iterations. Larger values make early iterations count less. This is called "tau0" in the Online LDA paper (Hoffman et al., 2010) Default: 1024, following Hoffman et al.   |

|                            |   |
|----------------------------|---|
| optimize_doc_concentration | (For Online optimizer only) Indicates whether the doc_concentration (Dirichlet parameter for document-topic distribution) will be optimized during training. Setting this to true will make the model more expressive and fit the training data better. Default: FALSE  |
| seed                       | A random seed. Set this value if you need your results to be reproducible across repeated calls.  |
| features_col               | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| topic_distribution_col     | Output column with estimates of the topic mixture distribution for each document (often called "theta" in the literature). Returns a vector of zeros for an empty document.   |
| uid                        | A character string used to uniquely identify the ML estimator.  |
| ...                        | Optional arguments, see Details. #' @return The object returned depends on the class of x. If it is a spark_connection, the function returns a ml_estimator object. If it is a ml_pipeline, it will return a pipeline with the predictor appended to it. If a tbl_spark, it will return a tbl_spark with the predictions added to it. |
| model                      | A fitted LDA model returned by ml_lda().  |
| max_terms_per_topic        | Maximum number of terms to collect for each topic. Default value of 10.   |
| dataset                    | test corpus to use for calculating log likelihood or log perplexity   |

## Details

For 'ml\_lda.tbl\_spark' with the formula interface, you can specify named arguments in '...' that will be passed 'ft\_regex\_tokenizer()', 'ft\_stop\_words\_remover()', and 'ft\_count\_vectorizer()'. For example, to increase the default 'min\_token\_length', you can use 'ml\_lda(dataset, ~ text, min\_token\_length = 4)'.

Terminology for LDA:

- "term" = "word": an element of the vocabulary
- "token": instance of a term appearing in a document
- "topic": multinomial distribution over terms representing some concept
- "document": one piece of text, corresponding to one row in the input data

Original LDA paper (journal version): Blei, Ng, and Jordan. "Latent Dirichlet Allocation." JMLR, 2003.

Input data (features\_col): LDA is given a collection of documents as input data, via the features\_col parameter. Each document is specified as a Vector of length vocab\_size, where each entry is the count for the corresponding term (word) in the document. Feature transformers such as [ft\\_tokenizer](#) and [ft\\_count\\_vectorizer](#) can be useful for converting text to word count vectors

**Value**

ml\_describe\_topics returns a DataFrame with topics and their top-weighted terms.

ml\_log\_likelihood calculates a lower bound on the log likelihood of the entire corpus

**Parameter details**

**doc\_concentration:** This is the parameter to a Dirichlet distribution, where larger values mean more smoothing (more regularization). If not set by the user, then doc\_concentration is set automatically. If set to singleton vector [alpha], then alpha is replicated to a vector of length k in fitting. Otherwise, the doc\_concentration vector must be length k. (default = automatic)

Optimizer-specific parameter settings:

EM

- Currently only supports symmetric distributions, so all values in the vector should be the same.
- Values should be greater than 1.0
- default = uniformly  $(50/k) + 1$ , where  $50/k$  is common in LDA libraries and +1 follows from Asuncion et al. (2009), who recommend a +1 adjustment for EM.

Online

- Values should be greater than or equal to 0
- default = uniformly  $(1.0/k)$ , following the implementation from [here](#)

**topic\_concentration:**

This is the parameter to a symmetric Dirichlet distribution.

Note: The topics' distributions over terms are called "beta" in the original LDA paper by Blei et al., but are called "phi" in many later papers such as Asuncion et al., 2009.

If not set by the user, then topic\_concentration is set automatically. (default = automatic)

Optimizer-specific parameter settings:

EM

- Value should be greater than 1.0
- default =  $0.1 + 1$ , where 0.1 gives a small amount of smoothing and +1 follows Asuncion et al. (2009), who recommend a +1 adjustment for EM.

Online

- Value should be greater than or equal to 0
- default =  $(1.0/k)$ , following the implementation from [here](#).

**topic\_distribution\_col:** This uses a variational approximation following Hoffman et al. (2010), where the approximate distribution is called "gamma." Technically, this method returns this approximation "gamma" for each document.

**Examples**

```
## Not run:
library(janeaustenr)
library(dplyr)
sc <- spark_connect(master = "local")
```

```

lines_tbl <- sdf_copy_to(sc,
  austen_books()[c(1:30), ],
  name = "lines_tbl",
  overwrite = TRUE
)

# transform the data in a tidy form
lines_tbl_tidy <- lines_tbl %>%
  ft_tokenizer(
    input_col = "text",
    output_col = "word_list"
  ) %>%
  ft_stop_words_remover(
    input_col = "word_list",
    output_col = "wo_stop_words"
  ) %>%
  mutate(text = explode(wo_stop_words)) %>%
  filter(text != "") %>%
  select(text, book)

lda_model <- lines_tbl_tidy %>%
  ml_lda(~text, k = 4)

# vocabulary and topics
tidy(lda_model)

## End(Not run)

```

---

ml\_lda\_tidiers

*Tidying methods for Spark ML LDA models*


---

## Description

These methods summarize the results of Spark ML models into tidy forms.

## Usage

```
## S3 method for class 'ml_model_lda'
tidy(x, ...)
```

```
## S3 method for class 'ml_model_lda'
augment(x, newdata = NULL, ...)
```

```
## S3 method for class 'ml_model_lda'
glance(x, ...)
```

**Arguments**

|         |  |
|---------|--|
| x       | a Spark ML model.                              |
| ...     | extra arguments (not used.)                    |
| newdata | a tbl_spark of new data to use for prediction. |

---

ml\_linear\_regression *Spark ML – Linear Regression*

---

**Description**

Perform regression using linear regression.

**Usage**

```
ml_linear_regression(
  x,
  formula = NULL,
  fit_intercept = TRUE,
  elastic_net_param = 0,
  reg_param = 0,
  max_iter = 100,
  weight_col = NULL,
  loss = "squaredError",
  solver = "auto",
  standardization = TRUE,
  tol = 1e-06,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  uid = random_string("linear_regression_"),
  ...
)
```

**Arguments**

|                   |   |
|-------------------|---|
| x                 | A spark_connection, ml_pipeline, or a tbl_spark.  |
| formula           | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details. |
| fit_intercept     | Boolean; should the model be fit with an intercept term?  |
| elastic_net_param | ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.  |
| reg_param         | Regularization parameter (aka lambda)   |
| max_iter          | The maximum number of iterations to use.  |

|                 |  |
|-----------------|--|
| weight_col      | The name of the column to use as weights for the model fit.  |
| loss            | The loss function to be optimized. Supported options: "squaredError" and "huber". Default: "squaredError"  |
| solver          | Solver algorithm for optimization.   |
| standardization | Whether to standardize the training features before fitting the model.   |
| tol             | Param for the convergence tolerance for iterative algorithms.  |
| features_col    | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> . |
| label_col       | Label column name. The column should be a numeric column. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| prediction_col  | Prediction column name.  |
| uid             | A character string used to uniquely identify the ML estimator.   |
| ...             | Optional arguments; see Details.   |

### Value

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the predictor appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the predictions added to it.

### See Also

Other ml algorithms: [ml\\_aft\\_survival\\_regression\(\)](#), [ml\\_decision\\_tree\\_classifier\(\)](#), [ml\\_gbt\\_classifier\(\)](#), [ml\\_generalized\\_linear\\_regression\(\)](#), [ml\\_isotonic\\_regression\(\)](#), [ml\\_linear\\_svc\(\)](#), [ml\\_logistic\\_regression\(\)](#), [ml\\_multilayer\\_perceptron\\_classifier\(\)](#), [ml\\_naive\\_bayes\(\)](#), [ml\\_one\\_vs\\_rest\(\)](#), [ml\\_random\\_forest\\_classifier\(\)](#)

### Examples

```
## Not run:
sc <- spark_connect(master = "local")
mtcars_tbl <- sdf_copy_to(sc, mtcars, name = "mtcars_tbl", overwrite = TRUE)

partitions <- mtcars_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

mtcars_training <- partitions$training
mtcars_test <- partitions$test

lm_model <- mtcars_training %>%
  ml_linear_regression(mpg ~ .)

pred <- ml_predict(lm_model, mtcars_test)

ml_regression_evaluator(pred, label_col = "mpg")

## End(Not run)
```

---

|               |                             |
|---------------|-----------------------------|
| ml_linear_svc | <i>Spark ML – LinearSVC</i> |
|---------------|-----------------------------|

---

### Description

Perform classification using linear support vector machines (SVM). This binary classifier optimizes the Hinge Loss using the OWLQN optimizer. Only supports L2 regularization currently.

### Usage

```
ml_linear_svc(
  x,
  formula = NULL,
  fit_intercept = TRUE,
  reg_param = 0,
  max_iter = 100,
  standardization = TRUE,
  weight_col = NULL,
  tol = 1e-06,
  threshold = 0,
  aggregation_depth = 2,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  raw_prediction_col = "rawPrediction",
  uid = random_string("linear_svc_"),
  ...
)
```

### Arguments

|                   |  |
|-------------------|--|
| x                 | A spark_connection, ml_pipeline, or a tbl_spark.   |
| formula           | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">fit_r_formula</a> for details. |
| fit_intercept     | Boolean; should the model be fit with an intercept term?   |
| reg_param         | Regularization parameter (aka lambda)  |
| max_iter          | The maximum number of iterations to use.   |
| standardization   | Whether to standardize the training features before fitting the model.   |
| weight_col        | The name of the column to use as weights for the model fit.  |
| tol               | Param for the convergence tolerance for iterative algorithms.  |
| threshold         | in binary classification prediction, in range [0, 1].  |
| aggregation_depth | (Spark 2.1.0+) Suggested depth for treeAggregate (>= 2).   |

|                    |  |
|--------------------|--|
| features_col       | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> . |
| label_col          | Label column name. The column should be a numeric column. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| prediction_col     | Prediction column name.  |
| raw_prediction_col | Raw prediction (a.k.a. confidence) column name.  |
| uid                | A character string used to uniquely identify the ML estimator.   |
| ...                | Optional arguments; see Details.   |

### Value

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the predictor appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the predictions added to it.

### See Also

Other ml algorithms: [ml\\_aft\\_survival\\_regression\(\)](#), [ml\\_decision\\_tree\\_classifier\(\)](#), [ml\\_gbt\\_classifier\(\)](#), [ml\\_generalized\\_linear\\_regression\(\)](#), [ml\\_isotonic\\_regression\(\)](#), [ml\\_linear\\_regression\(\)](#), [ml\\_logistic\\_regression\(\)](#), [ml\\_multilayer\\_perceptron\\_classifier\(\)](#), [ml\\_naive\\_bayes\(\)](#), [ml\\_one\\_vs\\_rest\(\)](#), [ml\\_random\\_forest\\_classifier\(\)](#)

### Examples

```
## Not run:
library(dplyr)

sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

partitions <- iris_tbl %>%
  filter(Species != "setosa") %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

iris_training <- partitions$training
iris_test <- partitions$test

svc_model <- iris_training %>%
  ml_linear_svc(Species ~ .)

pred <- ml_predict(svc_model, iris_test)

ml_binary_classification_evaluator(pred)

## End(Not run)
```



---

ml\_linear\_svc\_tidiers *Tidying methods for Spark ML linear svc*

---

### Description

These methods summarize the results of Spark ML models into tidy forms.

### Usage

```
## S3 method for class 'ml_model_linear_svc'  
tidy(x, ...)  
  
## S3 method for class 'ml_model_linear_svc'  
augment(x, newdata = NULL, ...)  
  
## S3 method for class 'ml_model_linear_svc'  
glance(x, ...)
```

### Arguments

|         |  |
|---------|--|
| x       | a Spark ML model.                              |
| ...     | extra arguments (not used.)                    |
| newdata | a tbl_spark of new data to use for prediction. |

---

ml\_logistic\_regression  
*Spark ML – Logistic Regression*

---

### Description

Perform classification using logistic regression.

### Usage

```
ml_logistic_regression(  
  x,  
  formula = NULL,  
  fit_intercept = TRUE,  
  elastic_net_param = 0,  
  reg_param = 0,  
  max_iter = 100,  
  threshold = 0.5,  
  thresholds = NULL,  
  tol = 1e-06,  
  weight_col = NULL,
```

```

aggregation_depth = 2,
lower_bounds_on_coefficients = NULL,
lower_bounds_on_intercepts = NULL,
upper_bounds_on_coefficients = NULL,
upper_bounds_on_intercepts = NULL,
features_col = "features",
label_col = "label",
family = "auto",
prediction_col = "prediction",
probability_col = "probability",
raw_prediction_col = "rawPrediction",
uid = random_string("logistic_regression_"),
...
)

```

### Arguments

|                              |   |
|------------------------------|---|
| x                            | A spark_connection, ml_pipeline, or a tbl_spark.  |
| formula                      | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details.   |
| fit_intercept                | Boolean; should the model be fit with an intercept term?  |
| elastic_net_param            | ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.  |
| reg_param                    | Regularization parameter (aka lambda)   |
| max_iter                     | The maximum number of iterations to use.  |
| threshold                    | in binary classification prediction, in range [0, 1].   |
| thresholds                   | Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold. |
| tol                          | Param for the convergence tolerance for iterative algorithms.   |
| weight_col                   | The name of the column to use as weights for the model fit.   |
| aggregation_depth            | (Spark 2.1.0+) Suggested depth for treeAggregate (>= 2).  |
| lower_bounds_on_coefficients | (Spark 2.2.0+) Lower bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) for multinomial regression.   |
| lower_bounds_on_intercepts   | (Spark 2.2.0+) Lower bounds on intercepts if fitting under bound constrained optimization. The bounds vector size must be equal with 1 for binomial regression, or the number of classes for multinomial regression.  |

|                              |   |
|------------------------------|---|
| upper_bounds_on_coefficients | (Spark 2.2.0+) Upper bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) for multinomial regression. |
| upper_bounds_on_intercepts   | (Spark 2.2.0+) Upper bounds on intercepts if fitting under bound constrained optimization. The bounds vector size must be equal with 1 for binomial regression, or the number of classes for multinomial regression.  |
| features_col                 | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| label_col                    | Label column name. The column should be a numeric column. Usually this column is output by <a href="#">ft_r_formula</a> .   |
| family                       | (Spark 2.1.0+) Param for the name of family which is a description of the label distribution to be used in the model. Supported options: "auto", "binomial", and "multinomial."   |
| prediction_col               | Prediction column name.   |
| probability_col              | Column name for predicted class conditional probabilities.  |
| raw_prediction_col           | Raw prediction (a.k.a. confidence) column name.   |
| uid                          | A character string used to uniquely identify the ML estimator.  |
| ...                          | Optional arguments; see Details.  |

**Value**

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the predictor appended to it. If a tbl\_spark, it will return a tbl\_spark with the predictions added to it.

**See Also**

Other ml algorithms: [ml\\_aft\\_survival\\_regression\(\)](#), [ml\\_decision\\_tree\\_classifier\(\)](#), [ml\\_gbt\\_classifier\(\)](#), [ml\\_generalized\\_linear\\_regression\(\)](#), [ml\\_isotonic\\_regression\(\)](#), [ml\\_linear\\_regression\(\)](#), [ml\\_linear\\_svc\(\)](#), [ml\\_multilayer\\_perceptron\\_classifier\(\)](#), [ml\\_naive\\_bayes\(\)](#), [ml\\_one\\_vs\\_rest\(\)](#), [ml\\_random\\_forest\\_classifier\(\)](#)

**Examples**

```
## Not run:
sc <- spark_connect(master = "local")
mtcars_tbl <- sdf_copy_to(sc, mtcars, name = "mtcars_tbl", overwrite = TRUE)

partitions <- mtcars_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

mtcars_training <- partitions$training
```

```
mtcars_test <- partitions$test

lr_model <- mtcars_training %>%
  ml_logistic_regression(am ~ gear + carb)

pred <- ml_predict(lr_model, mtcars_test)

ml_binary_classification_evaluator(pred)

## End(Not run)
```

---

ml\_logistic\_regression\_tidiers

*Tidying methods for Spark ML Logistic Regression*

---

## Description

These methods summarize the results of Spark ML models into tidy forms.

## Usage

```
## S3 method for class 'ml_model_logistic_regression'
tidy(x, ...)

## S3 method for class 'ml_model_logistic_regression'
augment(x, newdata = NULL, ...)

## S3 method for class '`ml_model_logistic_regression`'
augment(x, new_data = NULL, ...)

## S3 method for class 'ml_model_logistic_regression'
glance(x, ...)
```

## Arguments

|          |  |
|----------|--|
| x        | a Spark ML model.                              |
| ...      | extra arguments (not used.)                    |
| newdata  | a tbl_spark of new data to use for prediction. |
| new_data | a tbl_spark of new data to use for prediction. |

---

|                   |   |
|-------------------|---|
| ml_metrics_binary | <i>Extracts metrics from a fitted table</i> |
|-------------------|---|

---

### Description

The function works best when passed a ‘tbl\_spark’ created by ‘ml\_predict()’. The output ‘tbl\_spark’ will contain the correct variable types and format that the given Spark model "evaluator" expects.

### Usage

```
ml_metrics_binary(
  x,
  truth = label,
  estimate = rawPrediction,
  metrics = c("roc_auc", "pr_auc"),
  ...
)
```

### Arguments

|          |  |
|----------|--|
| x        | A ‘tbl_spark’ containing the estimate (prediction) and the truth (value of what actually happened)   |
| truth    | The name of the column from ‘x’ with an integer field containing the binary response (0 or 1). The ‘ml_predict()’ function will create a new field named ‘label’ which contains the expected type and values. ‘truth’ defaults to ‘label’. |
| estimate | The name of the column from ‘x’ that contains the prediction. Defaults to ‘rawPrediction’, since its type and expected values will match ‘truth’.  |
| metrics  | A character vector with the metrics to calculate. For binary models the possible values are: ‘roc_auc’ (Area under the Receiver Operator curve), ‘pr_auc’ (Area under the Precesion Recall curve). Defaults to: ‘roc_auc’, ‘pr_auc’        |
| ...      | Optional arguments; currently unused.  |

### Details

The ‘ml\_metrics’ family of functions implement Spark’s ‘evaluate’ closer to how the ‘yardstick’ package works. The functions expect a table containing the truth and estimate, and return a ‘tibble’ with the results. The ‘tibble’ has the same format and variable names as the output of the ‘yardstick’ functions.

### Examples

```
## Not run:
sc <- spark_connect("local")
tbl_iris <- copy_to(sc, iris)
prep_iris <- tbl_iris %>%
  mutate(is_setosa = ifelse(Species == "setosa", 1, 0))
iris_split <- sdf_random_split(prepare_iris, training = 0.5, test = 0.5)
```

```

model <- ml_logistic_regression(iris_split$training, "is_setosa ~ Sepal_Length")
tbl_predictions <- ml_predict(model, iris_split$test)
ml_metrics_binary(tbl_predictions)

## End(Not run)

```

---

ml\_metrics\_multiclass *Extracts metrics from a fitted table*

---

## Description

The function works best when passed a ‘tbl\_spark’ created by ‘ml\_predict()’. The output ‘tbl\_spark’ will contain the correct variable types and format that the given Spark model "evaluator" expects.

## Usage

```

ml_metrics_multiclass(
  x,
  truth = label,
  estimate = prediction,
  metrics = c("accuracy"),
  beta = NULL,
  ...
)

```

## Arguments

|          |   |
|----------|---|
| x        | A ‘tbl_spark’ containing the estimate (prediction) and the truth (value of what actually happened)  |
| truth    | The name of the column from ‘x’ with an integer field containing an the indexed value for each outcome . The ‘ml_predict()’ function will create a new field named ‘label’ which contains the expected type and values. ‘truth’ defaults to ‘label’.  |
| estimate | The name of the column from ‘x’ that contains the prediction. Defaults to ‘prediction’, since its type and indexed values will match ‘truth’.   |
| metrics  | A character vector with the metrics to calculate. For multiclass models the possible values are: ‘accuracy’, ‘f_meas’ (F-score), ‘recall’ and ‘precision’. This function translates the argument into an acceptable Spark parameter. If no translation is found, then the raw value of the argument is passed to Spark. This makes it possible to request a metric that is not listed here but, depending on version, it is available in Spark. Other metrics form multi-class models are: ‘weightedTruePositiveRate’, ‘weightedFalsePositiveRate’, ‘weightedFMeasure’, ‘truePositiveRateByLabel’, ‘falsePositiveRateByLabel’, ‘precisionByLabel’, ‘recallByLabel’, ‘fMeasureByLabel’, ‘logLoss’, ‘hammingLoss’ |
| beta     | Numerical value used for precision and recall. Defaults to NULL, but if the Spark session’s version is 3.0 and above, then NULL is changed to 1, unless something different is supplied in this argument.   |
| ...      | Optional arguments; currently unused.   |

## Details

The ‘ml\_metrics’ family of functions implement Spark’s ‘evaluate’ closer to how the ‘yardstick’ package works. The functions expect a table containing the truth and estimate, and return a ‘tibble’ with the results. The ‘tibble’ has the same format and variable names as the output of the ‘yardstick’ functions.

## Examples

```
## Not run:
sc <- spark_connect("local")
tbl_iris <- copy_to(sc, iris)
iris_split <- sdf_random_split(tbl_iris, training = 0.5, test = 0.5)
model <- ml_random_forest(iris_split$training, "Species ~ .")
tbl_predictions <- ml_predict(model, iris_split$test)

ml_metrics_multiclass(tbl_predictions)

# Request different metrics
ml_metrics_multiclass(tbl_predictions, metrics = c("recall", "precision"))

# Request metrics not translated by the function, but valid in Spark
ml_metrics_multiclass(tbl_predictions, metrics = c("logLoss", "hammingLoss"))

## End(Not run)
```

---

ml\_metrics\_regression *Extracts metrics from a fitted table*

---

## Description

The function works best when passed a ‘tbl\_spark’ created by ‘ml\_predict()’. The output ‘tbl\_spark’ will contain the correct variable types and format that the given Spark model "evaluator" expects.

## Usage

```
ml_metrics_regression(
  x,
  truth,
  estimate = prediction,
  metrics = c("rmse", "rsq", "mae"),
  ...
)
```

## Arguments

x A ‘tbl\_spark’ containing the estimate (prediction) and the truth (value of what actually happened)

|          |  |
|----------|--|
| truth    | The name of the column from 'x' that contains the value of what actually happened  |
| estimate | The name of the column from 'x' that contains the prediction. Defaults to 'prediction', since it is the default that 'ml_predict()' uses.  |
| metrics  | A character vector with the metrics to calculate. For regression models the possible values are: 'rmse' (Root mean squared error), 'mse' (Mean squared error), 'rsq' (R squared), 'mae' (Mean absolute error), and 'var' (Explained variance). Defaults to: 'rmse', 'rsq', 'mae' |
| ...      | Optional arguments; currently unused.  |

### Details

The 'ml\_metrics' family of functions implement Spark's 'evaluate' closer to how the 'yardstick' package works. The functions expect a table containing the truth and estimate, and return a 'tibble' with the results. The 'tibble' has the same format and variable names as the output of the 'yardstick' functions.

### Examples

```
## Not run:
sc <- spark_connect("local")
tbl_iris <- copy_to(sc, iris)
iris_split <- sdf_random_split(tbl_iris, training = 0.5, test = 0.5)
training <- iris_split$training
reg_formula <- "Sepal_Length ~ Sepal_Width + Petal_Length + Petal_Width"
model <- ml_generalized_linear_regression(training, reg_formula)
tbl_predictions <- ml_predict(model, iris_split$test)
tbl_predictions %>%
  ml_metrics_regression(Sepal_Length)

## End(Not run)
```

---

ml\_model\_data

*Extracts data associated with a Spark ML model*

---

### Description

Extracts data associated with a Spark ML model

### Usage

```
ml_model_data(object)
```

### Arguments

object            a Spark ML model



**Value**

A tbl\_spark

---

ml\_multilayer\_perceptron\_classifier

*Spark ML – Multilayer Perceptron*

---

**Description**

Classification model based on the Multilayer Perceptron. Each layer has sigmoid activation function, output layer has softmax.

**Usage**

```
ml_multilayer_perceptron_classifier(  
  x,  
  formula = NULL,  
  layers = NULL,  
  max_iter = 100,  
  step_size = 0.03,  
  tol = 1e-06,  
  block_size = 128,  
  solver = "l-bfgs",  
  seed = NULL,  
  initial_weights = NULL,  
  thresholds = NULL,  
  features_col = "features",  
  label_col = "label",  
  prediction_col = "prediction",  
  probability_col = "probability",  
  raw_prediction_col = "rawPrediction",  
  uid = random_string("multilayer_perceptron_classifier_"),  
  ...  
)
```

```
ml_multilayer_perceptron(  
  x,  
  formula = NULL,  
  layers,  
  max_iter = 100,  
  step_size = 0.03,  
  tol = 1e-06,  
  block_size = 128,  
  solver = "l-bfgs",  
  seed = NULL,  
  initial_weights = NULL,
```

```

features_col = "features",
label_col = "label",
thresholds = NULL,
prediction_col = "prediction",
probability_col = "probability",
raw_prediction_col = "rawPrediction",
uid = random_string("multilayer_perceptron_classifier_"),
response = NULL,
features = NULL,
...
)

```

### Arguments

|                 |   |
|-----------------|---|
| x               | A spark_connection, ml_pipeline, or a tbl_spark.  |
| formula         | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details.   |
| layers          | A numeric vector describing the layers – each element in the vector gives the size of a layer. For example, c(4, 5, 2) would imply three layers, with an input (feature) layer of size 4, an intermediate layer of size 5, and an output (class) layer of size 2.   |
| max_iter        | The maximum number of iterations to use.  |
| step_size       | Step size to be used for each iteration of optimization (> 0).  |
| tol             | Param for the convergence tolerance for iterative algorithms.   |
| block_size      | Block size for stacking input data in matrices to speed up the computation. Data is stacked within partitions. If block size is more than remaining data in a partition then it is adjusted to the size of this data. Recommended size is between 10 and 1000. Default: 128   |
| solver          | The solver algorithm for optimization. Supported options: "gd" (minibatch gradient descent) or "l-bfgs". Default: "l-bfgs"  |
| seed            | A random seed. Set this value if you need your results to be reproducible across repeated calls.  |
| initial_weights | The initial weights of the model.   |
| thresholds      | Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold. |
| features_col    | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| label_col       | Label column name. The column should be a numeric column. Usually this column is output by <a href="#">ft_r_formula</a> .   |

|                    |  |
|--------------------|--|
| prediction_col     | Prediction column name.  |
| probability_col    | Column name for predicted class conditional probabilities.                       |
| raw_prediction_col | Raw prediction (a.k.a. confidence) column name.                                  |
| uid                | A character string used to uniquely identify the ML estimator.                   |
| ...                | Optional arguments; see Details.   |
| response           | (Deprecated) The name of the response column (as a length-one character vector.) |
| features           | (Deprecated) The name of features (terms) to use for the model fit.              |

### Details

`ml_multilayer_perceptron()` is an alias for `ml_multilayer_perceptron_classifier()` for backwards compatibility.

### Value

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the predictor appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the predictions added to it.

### See Also

Other ml algorithms: [ml\\_aft\\_survival\\_regression\(\)](#), [ml\\_decision\\_tree\\_classifier\(\)](#), [ml\\_gbt\\_classifier\(\)](#), [ml\\_generalized\\_linear\\_regression\(\)](#), [ml\\_isotonic\\_regression\(\)](#), [ml\\_linear\\_regression\(\)](#), [ml\\_linear\\_svc\(\)](#), [ml\\_logistic\\_regression\(\)](#), [ml\\_naive\\_bayes\(\)](#), [ml\\_one\\_vs\\_rest\(\)](#), [ml\\_random\\_forest\\_classifier\(\)](#)

### Examples

```
## Not run:
sc <- spark_connect(master = "local")

iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)
partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

iris_training <- partitions$training
iris_test <- partitions$test

mlp_model <- iris_training %>%
  ml_multilayer_perceptron_classifier(Species ~ ., layers = c(4, 3, 3))

pred <- ml_predict(mlp_model, iris_test)

ml_multiclass_classification_evaluator(pred)

## End(Not run)
```

---

 ml\_multilayer\_perceptron\_tidiers

*Tidying methods for Spark ML MLP*


---

### Description

These methods summarize the results of Spark ML models into tidy forms.

### Usage

```
## S3 method for class 'ml_model_multilayer_perceptron_classification'
tidy(x, ...)
```

```
## S3 method for class 'ml_model_multilayer_perceptron_classification'
augment(x, newdata = NULL, ...)
```

```
## S3 method for class 'ml_model_multilayer_perceptron_classification'
glance(x, ...)
```

### Arguments

|         |  |
|---------|--|
| x       | a Spark ML model.                              |
| ...     | extra arguments (not used.)                    |
| newdata | a tbl_spark of new data to use for prediction. |

---

 ml\_naive\_bayes

*Spark ML – Naive-Bayes*


---

### Description

Naive Bayes Classifiers. It supports Multinomial NB (see [here](#)) which can handle finitely supported discrete data. For example, by converting documents into TF-IDF vectors, it can be used for document classification. By making every vector a binary (0/1) data, it can also be used as Bernoulli NB (see [here](#)). The input feature values must be nonnegative.

### Usage

```
ml_naive_bayes(
  x,
  formula = NULL,
  model_type = "multinomial",
  smoothing = 1,
  thresholds = NULL,
  weight_col = NULL,
  features_col = "features",
```

```

    label_col = "label",
    prediction_col = "prediction",
    probability_col = "probability",
    raw_prediction_col = "rawPrediction",
    uid = random_string("naive_bayes_"),
    ...
)

```

### Arguments

|                    |   |
|--------------------|---|
| x                  | A spark_connection, ml_pipeline, or a tbl_spark.  |
| formula            | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details.   |
| model_type         | The model type. Supported options: "multinomial" and "bernoulli". (default = multinomial)   |
| smoothing          | The (Laplace) smoothing parameter. Defaults to 1.   |
| thresholds         | Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0 excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold. |
| weight_col         | (Spark 2.1.0+) Weight column name. If this is not set or empty, we treat all instance weights as 1.0.   |
| features_col       | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| label_col          | Label column name. The column should be a numeric column. Usually this column is output by <a href="#">ft_r_formula</a> .   |
| prediction_col     | Prediction column name.   |
| probability_col    | Column name for predicted class conditional probabilities.  |
| raw_prediction_col | Raw prediction (a.k.a. confidence) column name.   |
| uid                | A character string used to uniquely identify the ML estimator.  |
| ...                | Optional arguments; see Details.  |

### Value

The object returned depends on the class of x. If it is a spark\_connection, the function returns a ml\_estimator object. If it is a ml\_pipeline, it will return a pipeline with the predictor appended to it. If a tbl\_spark, it will return a tbl\_spark with the predictions added to it.

**See Also**

Other ml algorithms: `ml_aft_survival_regression()`, `ml_decision_tree_classifier()`, `ml_gbt_classifier()`, `ml_generalized_linear_regression()`, `ml_isotonic_regression()`, `ml_linear_regression()`, `ml_linear_svc()`, `ml_logistic_regression()`, `ml_multilayer_perceptron_classifier()`, `ml_one_vs_rest()`, `ml_random_forest_classifier()`

**Examples**

```
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

iris_training <- partitions$training
iris_test <- partitions$test

nb_model <- iris_training %>%
  ml_naive_bayes(Species ~ .)

pred <- ml_predict(nb_model, iris_test)

ml_multiclass_classification_evaluator(pred)

## End(Not run)
```

---

ml\_naive\_bayes\_tidiers

*Tidying methods for Spark ML Naive Bayes*

---

**Description**

These methods summarize the results of Spark ML models into tidy forms.

**Usage**

```
## S3 method for class 'ml_model_naive_bayes'
tidy(x, ...)

## S3 method for class 'ml_model_naive_bayes'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_naive_bayes'
glance(x, ...)
```

**Arguments**

|         |  |
|---------|--|
| x       | a Spark ML model.                              |
| ...     | extra arguments (not used.)                    |
| newdata | a tbl_spark of new data to use for prediction. |

---

|                |                             |
|----------------|-----------------------------|
| ml_one_vs_rest | <i>Spark ML – OneVsRest</i> |
|----------------|-----------------------------|

---

**Description**

Reduction of Multiclass Classification to Binary Classification. Performs reduction using one against all strategy. For a multiclass classification with k classes, train k models (one per class). Each example is scored against all k models and the model with highest score is picked to label the example.

**Usage**

```
ml_one_vs_rest(
  x,
  formula = NULL,
  classifier = NULL,
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  uid = random_string("one_vs_rest_"),
  ...
)
```

**Arguments**

|                |   |
|----------------|---|
| x              | A spark_connection, ml_pipeline, or a tbl_spark.  |
| formula        | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details. |
| classifier     | Object of class ml_estimator. Base binary classifier that we reduce multiclass classification into.   |
| features_col   | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| label_col      | Label column name. The column should be a numeric column. Usually this column is output by <a href="#">ft_r_formula</a> .   |
| prediction_col | Prediction column name.   |
| uid            | A character string used to uniquely identify the ML estimator.  |
| ...            | Optional arguments; see Details.  |

**Value**

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the predictor appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the predictions added to it.

**See Also**

Other ml algorithms: `ml_aft_survival_regression()`, `ml_decision_tree_classifier()`, `ml_gbt_classifier()`, `ml_generalized_linear_regression()`, `ml_isotonic_regression()`, `ml_linear_regression()`, `ml_linear_svc()`, `ml_logistic_regression()`, `ml_multilayer_perceptron_classifier()`, `ml_naive_bayes()`, `ml_random_forest_classifier()`

---

ml\_pca\_tidiers

*Tidying methods for Spark ML Principal Component Analysis*


---

**Description**

These methods summarize the results of Spark ML models into tidy forms.

**Usage**

```
## S3 method for class 'ml_model_pca'
tidy(x, ...)

## S3 method for class 'ml_model_pca'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_pca'
glance(x, ...)
```

**Arguments**

|                      |   |
|----------------------|---|
| <code>x</code>       | a Spark ML model.   |
| <code>...</code>     | extra arguments (not used.)                                 |
| <code>newdata</code> | a <code>tbl_spark</code> of new data to use for prediction. |



---

|             |                             |
|-------------|-----------------------------|
| ml_pipeline | <i>Spark ML – Pipelines</i> |
|-------------|-----------------------------|

---

**Description**

Create Spark ML Pipelines

**Usage**

```
ml_pipeline(x, ..., uid = random_string("pipeline_"))
```

**Arguments**

|     |  |
|-----|--|
| x   | Either a spark_connection or ml_pipeline_stage objects         |
| ... | ml_pipeline_stage objects.                                     |
| uid | A character string used to uniquely identify the ML estimator. |

**Value**

When x is a spark\_connection, ml\_pipeline() returns an empty pipeline object. When x is a ml\_pipeline\_stage, ml\_pipeline() returns an ml\_pipeline with the stages set to x and any transformers or estimators given in ....

---

|                    |  |
|--------------------|--|
| ml_power_iteration | <i>Spark ML – Power Iteration Clustering</i> |
|--------------------|--|

---

**Description**

Power iteration clustering (PIC) is a scalable and efficient algorithm for clustering vertices of a graph given pairwise similarities as edge properties, described in the paper "Power Iteration Clustering" by Frank Lin and William W. Cohen. It computes a pseudo-eigenvector of the normalized affinity matrix of the graph via power iteration and uses it to cluster vertices. spark.mllib includes an implementation of PIC using GraphX as its backend. It takes an RDD of (srcId, dstId, similarity) tuples and outputs a model with the clustering assignments. The similarities must be nonnegative. PIC assumes that the similarity measure is symmetric. A pair (srcId, dstId) regardless of the ordering should appear at most once in the input data. If a pair is missing from input, their similarity is treated as zero.

**Usage**

```
ml_power_iteration(
  x,
  k = 4,
  max_iter = 20,
  init_mode = "random",
  src_col = "src",
  dst_col = "dst",
  weight_col = "weight",
  ...
)
```

**Arguments**

|            |   |
|------------|---|
| x          | A 'spark_connection' or a 'tbl_spark'.  |
| k          | The number of clusters to create.   |
| max_iter   | The maximum number of iterations to run.  |
| init_mode  | This can be either "random", which is the default, to use a random vector as vertex properties, or "degree" to use normalized sum similarities. |
| src_col    | Column in the input Spark dataframe containing 0-based indexes of all source vertices in the affinity matrix described in the PIC paper.        |
| dst_col    | Column in the input Spark dataframe containing 0-based indexes of all destination vertices in the affinity matrix described in the PIC paper.   |
| weight_col | Column in the input Spark dataframe containing non-negative edge weights in the affinity matrix described in the PIC paper.                     |
| ...        | Optional arguments. Currently unused.   |

**Value**

A 2-column R dataframe with columns named "id" and "cluster" describing the resulting cluster assignments

**Examples**

```
## Not run:

library(sparklyr)

sc <- spark_connect(master = "local")

r1 <- 1
n1 <- 80L
r2 <- 4
n2 <- 80L

gen_circle <- function(radius, num_pts) {
  # generate evenly distributed points on a circle centered at the origin
  seq(0, num_pts - 1) %>%
```

```

    lapply(
      function(pt) {
        theta <- 2 * pi * pt / num_pts

        radius * c(cos(theta), sin(theta))
      }
    )
  }

gaussian_similarity <- function(pt1, pt2) {
  dist2 <- sum((pt2 - pt1)^2)

  exp(-dist2 / 2)
}

gen_pic_data <- function() {
  # generate points on 2 concentric circle centered at the origin and then
  # compute pairwise Gaussian similarity values of all unordered pair of
  # points
  n <- n1 + n2
  pts <- append(gen_circle(r1, n1), gen_circle(r2, n2))
  num_unordered_pairs <- n * (n - 1) / 2

  src <- rep(0L, num_unordered_pairs)
  dst <- rep(0L, num_unordered_pairs)
  sim <- rep(0, num_unordered_pairs)

  idx <- 1
  for (i in seq(2, n)) {
    for (j in seq(i - 1)) {
      src[[idx]] <- i - 1L
      dst[[idx]] <- j - 1L
      sim[[idx]] <- gaussian_similarity(pts[[i]], pts[[j]])
      idx <- idx + 1
    }
  }

  dplyr::tibble(src = src, dst = dst, sim = sim)
}

pic_data <- copy_to(sc, gen_pic_data())

clusters <- ml_power_iteration(
  pic_data,
  src_col = "src", dst_col = "dst", weight_col = "sim", k = 2, max_iter = 40
)
print(clusters)

## End(Not run)

```

ml\_prefixspan

*Frequent Pattern Mining – PrefixSpan***Description**

PrefixSpan algorithm for mining frequent itemsets.

**Usage**

```
ml_prefixspan(
  x,
  seq_col = "sequence",
  min_support = 0.1,
  max_pattern_length = 10,
  max_local_proj_db_size = 3.2e+07,
  uid = random_string("prefixspan_"),
  ...
)

ml_freq_seq_patterns(model)
```

**Arguments**

|                        |   |
|------------------------|---|
| x                      | A spark_connection, ml_pipeline, or a tbl_spark.  |
| seq_col                | The name of the sequence column in dataset (defaults to "sequence"). Rows with nulls in this column are ignored.  |
| min_support            | The minimum support required to be considered a frequent sequential pattern.  |
| max_pattern_length     | The maximum length of a frequent sequential pattern. Any frequent pattern exceeding this length will not be included in the results.  |
| max_local_proj_db_size | The maximum number of items allowed in a prefix-projected database before local iterative processing of the projected database begins. This parameter should be tuned with respect to the size of your executors. |
| uid                    | A character string used to uniquely identify the ML estimator.  |
| ...                    | Optional arguments; currently unused.   |
| model                  | A Prefix Span model.  |

**Examples**

```
## Not run:
library(sparklyr)
sc <- spark_connect(master = "local", version = "2.4.0")

items_df <- dplyr::tibble(
  seq = list(
```

```

    list(list(1, 2), list(3)),
    list(list(1), list(3, 2), list(1, 2)),
    list(list(1, 2), list(5)),
    list(list(6))
  )
)
items_sdf <- copy_to(sc, items_df, overwrite = TRUE)

prefix_span_model <- ml_prefixspan(
  sc,
  seq_col = "seq",
  min_support = 0.5,
  max_pattern_length = 5,
  max_local_proj_db_size = 32000000
)

frequent_items <- prefix_span_model$frequent_sequential_patterns(items_sdf) %>% collect()

## End(Not run)

```

---

```
ml_random_forest_classifier
```

*Spark ML – Random Forest*

---

## Description

Perform classification and regression using random forests.

## Usage

```
ml_random_forest_classifier(
  x,
  formula = NULL,
  num_trees = 20,
  subsampling_rate = 1,
  max_depth = 5,
  min_instances_per_node = 1,
  feature_subset_strategy = "auto",
  impurity = "gini",
  min_info_gain = 0,
  max_bins = 32,
  seed = NULL,
  thresholds = NULL,
  checkpoint_interval = 10,
  cache_node_ids = FALSE,
  max_memory_in_mb = 256,
  features_col = "features",

```

```
    label_col = "label",
    prediction_col = "prediction",
    probability_col = "probability",
    raw_prediction_col = "rawPrediction",
    uid = random_string("random_forest_classifier_"),
    ...
)
```

```
ml_random_forest(
  x,
  formula = NULL,
  type = c("auto", "regression", "classification"),
  features_col = "features",
  label_col = "label",
  prediction_col = "prediction",
  probability_col = "probability",
  raw_prediction_col = "rawPrediction",
  feature_subset_strategy = "auto",
  impurity = "auto",
  checkpoint_interval = 10,
  max_bins = 32,
  max_depth = 5,
  num_trees = 20,
  min_info_gain = 0,
  min_instances_per_node = 1,
  subsampling_rate = 1,
  seed = NULL,
  thresholds = NULL,
  cache_node_ids = FALSE,
  max_memory_in_mb = 256,
  uid = random_string("random_forest_"),
  response = NULL,
  features = NULL,
  ...
)
```

```
ml_random_forest_regressor(
  x,
  formula = NULL,
  num_trees = 20,
  subsampling_rate = 1,
  max_depth = 5,
  min_instances_per_node = 1,
  feature_subset_strategy = "auto",
  impurity = "variance",
  min_info_gain = 0,
  max_bins = 32,
  seed = NULL,
```

```

    checkpoint_interval = 10,
    cache_node_ids = FALSE,
    max_memory_in_mb = 256,
    features_col = "features",
    label_col = "label",
    prediction_col = "prediction",
    uid = random_string("random_forest_regressor_"),
    ...
)

```

## Arguments

|                         |   |
|-------------------------|---|
| x                       | A spark_connection, ml_pipeline, or a tbl_spark.  |
| formula                 | Used when x is a tbl_spark. R formula as a character string or a formula. This is used to transform the input dataframe before fitting, see <a href="#">ft_r_formula</a> for details.   |
| num_trees               | Number of trees to train ( $\geq 1$ ). If 1, then no bootstrapping is used. If $> 1$ , then bootstrapping is done.  |
| subsampling_rate        | Fraction of the training data used for learning each decision tree, in range (0, 1]. (default = 1.0)  |
| max_depth               | Maximum depth of the tree ( $\geq 0$ ); that is, the maximum number of nodes separating any leaves from the root of the tree.   |
| min_instances_per_node  | Minimum number of instances each child must have after split.   |
| feature_subset_strategy | The number of features to consider for splits at each tree node. See details for options.   |
| impurity                | Criterion used for information gain calculation. Supported: "entropy" and "gini" (default) for classification and "variance" (default) for regression. For ml_decision_tree, setting "auto" will default to the appropriate criterion based on model type.  |
| min_info_gain           | Minimum information gain for a split to be considered at a tree node. Should be $\geq 0$ , defaults to 0.   |
| max_bins                | The maximum number of bins used for discretizing continuous features and for choosing how to split on features at each node. More bins give higher granularity.   |
| seed                    | Seed for random numbers.  |
| thresholds              | Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values $> 0$ excepting that at most one value may be 0. The class with largest value $p/t$ is predicted, where p is the original probability of that class and t is the class's threshold. |
| checkpoint_interval     | Set checkpoint interval ( $\geq 1$ ) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations, defaults to 10.  |

|                    |  |
|--------------------|--|
| cache_node_ids     | If FALSE, the algorithm will pass trees to executors to match instances with nodes. If TRUE, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Defaults to FALSE.  |
| max_memory_in_mb   | Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size. Defaults to 256.  |
| features_col       | Features column name, as a length-one character vector. The column should be single vector column of numeric values. Usually this column is output by <a href="#">ft_r_formula</a> .   |
| label_col          | Label column name. The column should be a numeric column. Usually this column is output by <a href="#">ft_r_formula</a> .  |
| prediction_col     | Prediction column name.  |
| probability_col    | Column name for predicted class conditional probabilities.   |
| raw_prediction_col | Raw prediction (a.k.a. confidence) column name.  |
| uid                | A character string used to uniquely identify the ML estimator.   |
| ...                | Optional arguments; see Details.   |
| type               | The type of model to fit. "regression" treats the response as a continuous variable, while "classification" treats the response as a categorical variable. When "auto" is used, the model type is inferred based on the response variable type – if it is a numeric type, then regression is used; classification otherwise. |
| response           | (Deprecated) The name of the response column (as a length-one character vector.)   |
| features           | (Deprecated) The name of features (terms) to use for the model fit.  |

## Details

The supported options for `feature_subset_strategy` are

- "auto": Choose automatically for task: If `num_trees == 1`, set to "all". If `num_trees > 1` (forest), set to "sqrt" for classification and to "onethird" for regression.
- "all": use all features
- "onethird": use 1/3 of the features
- "sqrt": use use `sqrt(number of features)`
- "log2": use `log2(number of features)`
- "n": when `n` is in the range `(0, 1.0]`, use `n * number of features`. When `n` is in the range `(1, number of features)`, use `n` features. (default = "auto")

`ml_random_forest` is a wrapper around `ml_random_forest_regressor.tbl_spark` and `ml_random_forest_classifier` and calls the appropriate method based on model type.



**Value**

The object returned depends on the class of `x`. If it is a `spark_connection`, the function returns a `ml_estimator` object. If it is a `ml_pipeline`, it will return a pipeline with the predictor appended to it. If a `tbl_spark`, it will return a `tbl_spark` with the predictions added to it.

**See Also**

Other ml algorithms: `ml_aft_survival_regression()`, `ml_decision_tree_classifier()`, `ml_gbt_classifier()`, `ml_generalized_linear_regression()`, `ml_isotonic_regression()`, `ml_linear_regression()`, `ml_linear_svc()`, `ml_logistic_regression()`, `ml_multilayer_perceptron_classifier()`, `ml_naive_bayes()`, `ml_one_vs_rest()`

**Examples**

```
## Not run:
sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

partitions <- iris_tbl %>%
  sdf_random_split(training = 0.7, test = 0.3, seed = 1111)

iris_training <- partitions$training
iris_test <- partitions$test

rf_model <- iris_training %>%
  ml_random_forest(Species ~ ., type = "classification")

pred <- ml_predict(rf_model, iris_test)

ml_multiclass_classification_evaluator(pred)

## End(Not run)
```

---

`ml_stage`*Spark ML – Pipeline stage extraction*

---

**Description**

Extraction of stages from a Pipeline or PipelineModel object.

**Usage**

```
ml_stage(x, stage)
```

```
ml_stages(x, stages = NULL)
```

**Arguments**

|        |   |
|--------|---|
| x      | A ml_pipeline or a ml_pipeline_model object               |
| stage  | The UID of a stage in the pipeline.                       |
| stages | The UIDs of stages in the pipeline as a character vector. |

**Value**

For ml\_stage(): The stage specified.

For ml\_stages(): A list of stages. If stages is not set, the function returns all stages of the pipeline in a list.

---

|            |   |
|------------|---|
| ml_summary | <i>Spark ML – Extraction of summary metrics</i> |
|------------|---|

---

**Description**

Extracts a metric from the summary object of a Spark ML model.

**Usage**

```
ml_summary(x, metric = NULL, allow_null = FALSE)
```

**Arguments**

|            |   |
|------------|---|
| x          | A Spark ML model that has a summary.  |
| metric     | The name of the metric to extract. If not set, returns the summary object.    |
| allow_null | Whether null results are allowed when the metric is not found in the summary. |

---

|                                |   |
|--------------------------------|---|
| ml_survival_regression_tidiers | <i>Tidying methods for Spark ML Survival Regression</i> |
|--------------------------------|---|

---

**Description**

These methods summarize the results of Spark ML models into tidy forms.

**Usage**

```
## S3 method for class 'ml_model_aft_survival_regression'
tidy(x, ...)
```

```
## S3 method for class 'ml_model_aft_survival_regression'
augment(x, newdata = NULL, ...)
```

```
## S3 method for class 'ml_model_aft_survival_regression'
glance(x, ...)
```

**Arguments**

|         |  |
|---------|--|
| x       | a Spark ML model.                              |
| ...     | extra arguments (not used.)                    |
| newdata | a tbl_spark of new data to use for prediction. |

---

|                 |   |
|-----------------|---|
| ml_tree_tidiers | <i>Tidying methods for Spark ML tree models</i> |
|-----------------|---|

---

**Description**

These methods summarize the results of Spark ML models into tidy forms.

**Usage**

```
## S3 method for class 'ml_model_decision_tree_classification'
tidy(x, ...)

## S3 method for class 'ml_model_decision_tree_regression'
tidy(x, ...)

## S3 method for class 'ml_model_decision_tree_classification'
augment(x, newdata = NULL, ...)

## S3 method for class '`_ml_model_decision_tree_classification`'
augment(x, new_data = NULL, ...)

## S3 method for class 'ml_model_decision_tree_regression'
augment(x, newdata = NULL, ...)

## S3 method for class '`_ml_model_decision_tree_regression`'
augment(x, new_data = NULL, ...)

## S3 method for class 'ml_model_decision_tree_classification'
glance(x, ...)

## S3 method for class 'ml_model_decision_tree_regression'
glance(x, ...)

## S3 method for class 'ml_model_random_forest_classification'
tidy(x, ...)

## S3 method for class 'ml_model_random_forest_regression'
tidy(x, ...)

## S3 method for class 'ml_model_random_forest_classification'
augment(x, newdata = NULL, ...)
```

```

## S3 method for class '`_ml_model_random_forest_classification`'
augment(x, new_data = NULL, ...)

## S3 method for class 'ml_model_random_forest_regression'
augment(x, newdata = NULL, ...)

## S3 method for class '`_ml_model_random_forest_regression`'
augment(x, new_data = NULL, ...)

## S3 method for class 'ml_model_random_forest_classification'
glance(x, ...)

## S3 method for class 'ml_model_random_forest_regression'
glance(x, ...)

## S3 method for class 'ml_model_gbt_classification'
tidy(x, ...)

## S3 method for class 'ml_model_gbt_regression'
tidy(x, ...)

## S3 method for class 'ml_model_gbt_classification'
augment(x, newdata = NULL, ...)

## S3 method for class '`_ml_model_gbt_classification`'
augment(x, new_data = NULL, ...)

## S3 method for class 'ml_model_gbt_regression'
augment(x, newdata = NULL, ...)

## S3 method for class '`_ml_model_gbt_regression`'
augment(x, new_data = NULL, ...)

## S3 method for class 'ml_model_gbt_classification'
glance(x, ...)

## S3 method for class 'ml_model_gbt_regression'
glance(x, ...)

```

### Arguments

|          |  |
|----------|--|
| x        | a Spark ML model.                              |
| ...      | extra arguments (not used.)                    |
| newdata  | a tbl_spark of new data to use for prediction. |
| new_data | a tbl_spark of new data to use for prediction. |

---

|        |                       |
|--------|-----------------------|
| ml_uid | <i>Spark ML – UID</i> |
|--------|-----------------------|

---

**Description**

Extracts the UID of an ML object.

**Usage**

```
ml_uid(x)
```

**Arguments**

|   |                   |
|---|-------------------|
| x | A Spark ML object |
|---|-------------------|

---

|                         |   |
|-------------------------|---|
| ml_unsupervised_tidiers | <i>Tidying methods for Spark ML unsupervised models</i> |
|-------------------------|---|

---

**Description**

These methods summarize the results of Spark ML models into tidy forms.

**Usage**

```
## S3 method for class 'ml_model_kmeans'
tidy(x, ...)
```

```
## S3 method for class 'ml_model_kmeans'
augment(x, newdata = NULL, ...)
```

```
## S3 method for class 'ml_model_kmeans'
glance(x, ...)
```

```
## S3 method for class 'ml_model_bisecting_kmeans'
tidy(x, ...)
```

```
## S3 method for class 'ml_model_bisecting_kmeans'
augment(x, newdata = NULL, ...)
```

```
## S3 method for class 'ml_model_bisecting_kmeans'
glance(x, ...)
```

```
## S3 method for class 'ml_model_gaussian_mixture'
tidy(x, ...)
```

```
## S3 method for class 'ml_model_gaussian_mixture'
augment(x, newdata = NULL, ...)

## S3 method for class 'ml_model_gaussian_mixture'
glance(x, ...)
```

### Arguments

|         |  |
|---------|--|
| x       | a Spark ML model.                              |
| ...     | extra arguments (not used.)                    |
| newdata | a tbl_spark of new data to use for prediction. |

---

|        |               |
|--------|---------------|
| mutate | <i>Mutate</i> |
|--------|---------------|

---

### Description

See [mutate](#) for more details.

---

|            |  |
|------------|--|
| na.replace | <i>Replace Missing Values in Objects</i> |
|------------|--|

---

### Description

This S3 generic provides an interface for replacing [NA](#) values within an object.

### Usage

```
na.replace(object, ...)
```

### Arguments

|        |   |
|--------|---|
| object | An R object.                                    |
| ...    | Arguments passed along to implementing methods. |

---

|      |             |
|------|-------------|
| nest | <i>Nest</i> |
|------|-------------|

---

### Description

See [nest](#) for more details.

---

|                           |                     |
|---------------------------|---------------------|
| <code>pivot_longer</code> | <i>Pivot longer</i> |
|---------------------------|---------------------|

---

**Description**

See [pivot\\_longer](#) for more details.

---

|                          |                    |
|--------------------------|--------------------|
| <code>pivot_wider</code> | <i>Pivot wider</i> |
|--------------------------|--------------------|

---

**Description**

See [pivot\\_wider](#) for more details.

---

|                            |                                 |
|----------------------------|---------------------------------|
| <code>random_string</code> | <i>Random string generation</i> |
|----------------------------|---------------------------------|

---

**Description**

Generate a random string with a given prefix.

**Usage**

```
random_string(prefix = "table")
```

**Arguments**

|                     |                                |
|---------------------|--------------------------------|
| <code>prefix</code> | A length-one character vector. |
|---------------------|--------------------------------|

---

|               |                              |
|---------------|------------------------------|
| reactiveSpark | <i>Reactive spark reader</i> |
|---------------|------------------------------|

---

**Description**

Given a spark object, returns a reactive data source for the contents of the spark object. This function is most useful to read Spark streams.

**Usage**

```
reactiveSpark(x, intervalMillis = 1000, session = NULL)
```

**Arguments**

|                |   |
|----------------|---|
| x              | An object coercable to a Spark DataFrame.   |
| intervalMillis | Approximate number of milliseconds to wait to retrieve updated data frame. This can be a numeric value, or a function that returns a numeric value. |
| session        | The user session to associate this file reader with, or NULL if none. If non-null, the reader will automatically stop when the session ends.        |

---

|                 |                                    |
|-----------------|------------------------------------|
| registerDoSpark | <i>Register a Parallel Backend</i> |
|-----------------|------------------------------------|

---

**Description**

Registers a parallel backend using the foreach package.

**Usage**

```
registerDoSpark(spark_conn, parallelism = NULL, ...)
```

**Arguments**

|             |   |
|-------------|---|
| spark_conn  | Spark connection to use   |
| parallelism | Level of parallelism to use for task execution (if unspecified, then it will take the value of 'SparkContext.defaultParallelism()' which by default is the number of cores available to the 'sparklyr' application) |
| ...         | additional options for sparklyr parallel backend (currently only the only valid option is 'nocompile')  |

**Value**

None



**Examples**

```
## Not run:  
  
sc <- spark_connect(master = "local")  
registerDoSpark(sc, nocompile = FALSE)  
  
## End(Not run)
```

---

|                    |   |
|--------------------|---|
| register_extension | <i>Register a Package that Implements a Spark Extension</i> |
|--------------------|---|

---

**Description**

Registering an extension package will result in the package being automatically scanned for spark dependencies when a connection to Spark is created.

**Usage**

```
register_extension(package)  
  
registered_extensions()
```

**Arguments**

|         |                             |
|---------|-----------------------------|
| package | The package(s) to register. |
|---------|-----------------------------|

**Note**

Packages should typically register their extensions in their `.onLoad` hook – this ensures that their extensions are registered when their namespaces are loaded.

---

|            |                   |
|------------|-------------------|
| replace_na | <i>Replace NA</i> |
|------------|-------------------|

---

**Description**

See [replace\\_na](#) for more details.

---

|            |                   |
|------------|-------------------|
| right_join | <i>Right join</i> |
|------------|-------------------|

---

**Description**

See [right\\_join](#) for more details.

---

|              |                                      |
|--------------|--------------------------------------|
| sdf-saveload | <i>Save / Load a Spark DataFrame</i> |
|--------------|--------------------------------------|

---

### Description

Routines for saving and loading Spark DataFrames.

### Usage

```
sdf_save_table(x, name, overwrite = FALSE, append = FALSE)
```

```
sdf_load_table(sc, name)
```

```
sdf_save_parquet(x, path, overwrite = FALSE, append = FALSE)
```

```
sdf_load_parquet(sc, path)
```

### Arguments

|           |   |
|-----------|---|
| x         | A spark_connection, ml_pipeline, or a tbl_spark.          |
| name      | The table name to assign to the saved Spark DataFrame.    |
| overwrite | Boolean; overwrite a pre-existing table of the same name? |
| append    | Boolean; append to a pre-existing table of the same name? |
| sc        | A spark_connection object.                                |
| path      | The path where the Spark DataFrame should be saved.       |

---

|                       |  |
|-----------------------|--|
| sdf-transform-methods | <i>Spark ML – Transform, fit, and predict methods (sdf_ interface)</i> |
|-----------------------|--|

---

### Description

Deprecated methods for transformation, fit, and prediction. These are mirrors of the corresponding [ml-transform-methods](#).

### Usage

```
sdf_predict(x, model, ...)
```

```
sdf_transform(x, transformer, ...)
```

```
sdf_fit(x, estimator, ...)
```

```
sdf_fit_and_transform(x, estimator, ...)
```

**Arguments**

|             |  |
|-------------|--|
| x           | A tbl_spark.   |
| model       | A ml_transformer or a ml_model object.                     |
| ...         | Optional arguments passed to the corresponding ml_methods. |
| transformer | A ml_transformer object.                                   |
| estimator   | A ml_estimator object.                                     |

**Value**

sdf\_predict(), sdf\_transform(), and sdf\_fit\_and\_transform() return a transformed dataframe whereas sdf\_fit() returns a ml\_transformer.

---

|           |  |
|-----------|--|
| sdf_along | <i>Create DataFrame for along Object</i> |
|-----------|--|

---

**Description**

Creates a DataFrame along the given object.

**Usage**

```
sdf_along(sc, along, repartition = NULL, type = c("integer", "integer64"))
```

**Arguments**

|             |  |
|-------------|--|
| sc          | The associated Spark connection.   |
| along       | Takes the length from the length of this argument.                                   |
| repartition | The number of partitions to use when distributing the data across the Spark cluster. |
| type        | The data type to use for the index, either "integer" or "integer64".                 |

---

|          |   |
|----------|---|
| sdf_bind | <i>Bind multiple Spark DataFrames by row and column</i> |
|----------|---|

---

**Description**

sdf\_bind\_rows() and sdf\_bind\_cols() are implementation of the common pattern of do.call(rbind, sdfs) or do.call(cbind, sdfs) for binding many Spark DataFrames into one.

**Usage**

```
sdf_bind_rows(..., id = NULL)
```

```
sdf_bind_cols(...)
```

**Arguments**

|     |   |
|-----|---|
| ... | Spark tbls to combine.<br>Each argument can either be a Spark DataFrame or a list of Spark DataFrames<br>When row-binding, columns are matched by name, and any missing columns with be filled with NA.<br>When column-binding, rows are matched by position, so all data frames must have the same number of rows.   |
| id  | Data frame identifier.<br>When id is supplied, a new column of identifiers is created to link each row to its original Spark DataFrame. The labels are taken from the named arguments to sdf_bind_rows(). When a list of Spark DataFrames is supplied, the labels are taken from the names of the list. If no names are found a numeric sequence is used instead. |

**Details**

The output of sdf\_bind\_rows() will contain a column if that column appears in any of the inputs.

**Value**

sdf\_bind\_rows() and sdf\_bind\_cols() return tbl\_spark

---

sdf\_broadcast

*Broadcast hint*


---

**Description**

Used to force broadcast hash joins.

**Usage**

```
sdf_broadcast(x)
```

**Arguments**

|   |  |
|---|--|
| x | A spark_connection, ml_pipeline, or a tbl_spark. |
|---|--|

---

|                |                                     |
|----------------|-------------------------------------|
| sdf_checkpoint | <i>Checkpoint a Spark DataFrame</i> |
|----------------|-------------------------------------|

---

**Description**

Checkpoint a Spark DataFrame

**Usage**

```
sdf_checkpoint(x, eager = TRUE)
```

**Arguments**

|       |  |
|-------|--|
| x     | an object coercible to a Spark DataFrame         |
| eager | whether to truncate the lineage of the DataFrame |

---

|              |                                    |
|--------------|------------------------------------|
| sdf_coalesce | <i>Coalesces a Spark DataFrame</i> |
|--------------|------------------------------------|

---

**Description**

Coalesces a Spark DataFrame

**Usage**

```
sdf_coalesce(x, partitions)
```

**Arguments**

|            |  |
|------------|--|
| x          | A spark_connection, ml_pipeline, or a tbl_spark. |
| partitions | number of partitions                             |

---

|             |  |
|-------------|--|
| sdf_collect | <i>Collect a Spark DataFrame into R.</i> |
|-------------|--|

---

### Description

Collects a Spark dataframe into R.

### Usage

```
sdf_collect(object, impl = c("row-wise", "row-wise-iter", "column-wise"), ...)
```

### Arguments

|        |   |
|--------|---|
| object | Spark dataframe to collect  |
| impl   | Which implementation to use while collecting Spark dataframe - row-wise: fetch the entire dataframe into memory and then process it row-by-row - row-wise-iter: iterate through the dataframe using RDD local iterator, processing one row at a time (hence reducing memory footprint) - column-wise: fetch the entire dataframe into memory and then process it column-by-column NOTE: (1) this will not apply to streaming or arrow use cases (2) this parameter will only affect implementation detail, and will not affect result of 'sdf_collect', and should only be set if performance profiling indicates any particular choice will be significantly better than the default choice ("row-wise") |
| ...    | Additional options.   |

---

|             |                                  |
|-------------|----------------------------------|
| sdf_copy_to | <i>Copy an Object into Spark</i> |
|-------------|----------------------------------|

---

### Description

Copy an object into Spark, and return an R object wrapping the copied object (typically, a Spark DataFrame).

### Usage

```
sdf_copy_to(sc, x, name, memory, repartition, overwrite, struct_columns, ...)
```

```
sdf_import(x, sc, name, memory, repartition, overwrite, struct_columns, ...)
```

**Arguments**

|                             |   |
|-----------------------------|---|
| <code>sc</code>             | The associated Spark connection.  |
| <code>x</code>              | An R object from which a Spark DataFrame can be generated.  |
| <code>name</code>           | The name to assign to the copied table in Spark.  |
| <code>memory</code>         | Boolean; should the table be cached into memory?  |
| <code>repartition</code>    | The number of partitions to use when distributing the table across the Spark cluster. The default (0) can be used to avoid partitioning.  |
| <code>overwrite</code>      | Boolean; overwrite a pre-existing table with the name <code>name</code> if one already exists?  |
| <code>struct_columns</code> | (only supported with Spark 2.4.0 or higher) A list of columns from the source data frame that should be converted to Spark SQL StructType columns. The source columns can contain either json strings or nested lists. All rows within each source column should have identical schemas (because otherwise the conversion result will contain unexpected null values or missing values as Spark currently does not support schema discovery on individual rows within a struct column). |
| <code>...</code>            | Optional arguments, passed to implementing methods.   |

**Advanced Usage**

`sdf_copy_to` is an S3 generic that, by default, dispatches to `sdf_import`. Package authors that would like to implement `sdf_copy_to` for a custom object type can accomplish this by implementing the associated method on `sdf_import`.

**See Also**

Other Spark data frames: [sdf\\_distinct\(\)](#), [sdf\\_random\\_split\(\)](#), [sdf\\_register\(\)](#), [sdf\\_sample\(\)](#), [sdf\\_sort\(\)](#), [sdf\\_weighted\\_sample\(\)](#)

**Examples**

```
## Not run:
sc <- spark_connect(master = "spark://HOST:PORT")
sdf_copy_to(sc, iris)

## End(Not run)
```

---

sdf\_crosstab

*Cross Tabulation*


---

**Description**

Builds a contingency table at each combination of factor levels.

**Usage**

```
sdf_crosstab(x, col1, col2)
```

**Arguments**

|      |  |
|------|--|
| x    | A Spark DataFrame  |
| col1 | The name of the first column. Distinct items will make the first item of each row.         |
| col2 | The name of the second column. Distinct items will make the column names of the DataFrame. |

**Value**

A DataFrame containing the contingency table.

---

|                  |                                       |
|------------------|---------------------------------------|
| sdf_debug_string | <i>Debug Info for Spark DataFrame</i> |
|------------------|---------------------------------------|

---

**Description**

Prints plan of execution to generate x. This plan will, among other things, show the number of partitions in parenthesis at the far left and indicate stages using indentation.

**Usage**

```
sdf_debug_string(x, print = TRUE)
```

**Arguments**

|       |   |
|-------|---|
| x     | An R object wrapping, or containing, a Spark DataFrame. |
| print | Print debug information?                                |

---

|              |   |
|--------------|---|
| sdf_describe | <i>Compute summary statistics for columns of a data frame</i> |
|--------------|---|

---

**Description**

Compute summary statistics for columns of a data frame

**Usage**

```
sdf_describe(x, cols = colnames(x))
```

**Arguments**

|      |  |
|------|--|
| x    | An object coercible to a Spark DataFrame                       |
| cols | Columns to compute statistics for, given as a character vector |



---

|         |   |
|---------|---|
| sdf_dim | <i>Support for Dimension Operations</i> |
|---------|---|

---

**Description**

sdf\_dim(), sdf\_nrow() and sdf\_ncol() provide similar functionality to dim(), nrow() and ncol().

**Usage**

```
sdf_dim(x)
```

```
sdf_nrow(x)
```

```
sdf_ncol(x)
```

**Arguments**

|   |                                  |
|---|----------------------------------|
| x | An object (usually a spark_tbl). |
|---|----------------------------------|

---

|              |   |
|--------------|---|
| sdf_distinct | <i>Invoke distinct on a Spark DataFrame</i> |
|--------------|---|

---

**Description**

Invoke distinct on a Spark DataFrame

**Usage**

```
sdf_distinct(x, ..., name)
```

**Arguments**

|      |  |
|------|--|
| x    | A Spark DataFrame.   |
| ...  | Optional variables to use when determining uniqueness. If there are multiple rows for a given combination of inputs, only the first row will be preserved. If omitted, will use all variables. |
| name | A name to assign this table. Passed to [sdf_register()].   |

**See Also**

Other Spark data frames: [sdf\\_copy\\_to\(\)](#), [sdf\\_random\\_split\(\)](#), [sdf\\_register\(\)](#), [sdf\\_sample\(\)](#), [sdf\\_sort\(\)](#), [sdf\\_weighted\\_sample\(\)](#)

---

sdf\_drop\_duplicates     *Remove duplicates from a Spark DataFrame*

---

**Description**

Remove duplicates from a Spark DataFrame

**Usage**

```
sdf_drop_duplicates(x, cols = NULL)
```

**Arguments**

|      |  |
|------|--|
| x    | An object coercible to a Spark DataFrame                   |
| cols | Subset of Columns to consider, given as a character vector |

---

sdf\_expand\_grid     *Create a Spark dataframe containing all combinations of inputs*

---

**Description**

Given one or more R vectors/factors or single-column Spark dataframes, perform an `expand.grid` operation on all of them and store the result in a Spark dataframe

**Usage**

```
sdf_expand_grid(
  sc,
  ...,
  broadcast_vars = NULL,
  memory = TRUE,
  repartition = NULL,
  partition_by = NULL
)
```

**Arguments**

|                |  |
|----------------|--|
| sc             | The associated Spark connection.   |
| ...            | Each input variable can be either a R vector/factor or a Spark dataframe. Un-named inputs will assume the default names of 'Var1', 'Var2', etc in the result, similar to what 'expand.grid' does for unnamed inputs. |
| broadcast_vars | Indicates which input(s) should be broadcasted to all nodes of the Spark cluster during the join process (default: none).  |
| memory         | Boolean; whether the resulting Spark dataframe should be cached into memory (default: TRUE)  |

|              |   |
|--------------|---|
| repartition  | Number of partitions the resulting Spark dataframe should have  |
| partition_by | Vector of column names used for partitioning the resulting Spark dataframe, only supported for Spark 2.0+ |

### Examples

```
## Not run:
sc <- spark_connect(master = "local")
grid_sdf <- sdf_expand_grid(sc, seq(5), rnorm(10), letters)

## End(Not run)
```

---

|               |   |
|---------------|---|
| sdf_from_avro | <i>Convert column(s) from avro format</i> |
|---------------|---|

---

### Description

Convert column(s) from avro format

### Usage

```
sdf_from_avro(x, cols)
```

### Arguments

|      |  |
|------|--|
| x    | An object coercible to a Spark DataFrame   |
| cols | Named list of columns to transform from Avro format plus a valid Avro schema string for each column, where column names are keys and column schema strings are values (e.g., c(example_primitive_col = "string", example_complex_col = "{ \"type\": \"record\", \"name\": \"person\", \"fields\": [ { \"name\": \"person_name\", \"type\": \"string\" }, { \"name\": \"person_id\", \"type\": \"long\" } ] }") |

---

|                  |                                     |
|------------------|-------------------------------------|
| sdf_is_streaming | <i>Spark DataFrame is Streaming</i> |
|------------------|-------------------------------------|

---

### Description

Is the given Spark DataFrame a streaming data?

### Usage

```
sdf_is_streaming(x)
```

### Arguments

|   |  |
|---|--|
| x | A spark_connection, ml_pipeline, or a tbl_spark. |
|---|--|

---

|                |  |
|----------------|--|
| sdf_last_index | <i>Returns the last index of a Spark DataFrame</i> |
|----------------|--|

---

### Description

Returns the last index of a Spark DataFrame. The Spark `mapPartitionsWithIndex` function is used to iterate through the last nonempty partition of the RDD to find the last record.

### Usage

```
sdf_last_index(x, id = "id")
```

### Arguments

|    |  |
|----|--|
| x  | A <code>spark_connection</code> , <code>ml_pipeline</code> , or a <code>tbl_spark</code> . |
| id | The name of the index column.  |

---

|         |                                    |
|---------|------------------------------------|
| sdf_len | <i>Create DataFrame for Length</i> |
|---------|------------------------------------|

---

### Description

Creates a DataFrame for the given length.

### Usage

```
sdf_len(sc, length, repartition = NULL, type = c("integer", "integer64"))
```

### Arguments

|             |  |
|-------------|--|
| sc          | The associated Spark connection.   |
| length      | The desired length of the sequence.  |
| repartition | The number of partitions to use when distributing the data across the Spark cluster. |
| type        | The data type to use for the index, either "integer" or "integer64".                 |

---

|                    |   |
|--------------------|---|
| sdf_num_partitions | <i>Gets number of partitions of a Spark DataFrame</i> |
|--------------------|---|

---

**Description**

Gets number of partitions of a Spark DataFrame

**Usage**

```
sdf_num_partitions(x)
```

**Arguments**

x                    A spark\_connection, ml\_pipeline, or a tbl\_spark.

---

|                     |   |
|---------------------|---|
| sdf_partition_sizes | <i>Compute the number of records within each partition of a Spark DataFrame</i> |
|---------------------|---|

---

**Description**

Compute the number of records within each partition of a Spark DataFrame

**Usage**

```
sdf_partition_sizes(x)
```

**Arguments**

x                    A spark\_connection, ml\_pipeline, or a tbl\_spark.

**Examples**

```
## Not run:
library(sparklyr)
sc <- spark_connect(master = "spark://HOST:PORT")
example_sdf <- sdf_len(sc, 100L, repartition = 10L)
example_sdf %>%
  sdf_partition_sizes() %>%
  print()

## End(Not run)
```

---

sdf\_persist *Persist a Spark DataFrame*

---

### Description

Persist a Spark DataFrame, forcing any pending computations and (optionally) serializing the results to disk.

### Usage

```
sdf_persist(x, storage.level = "MEMORY_AND_DISK", name = NULL)
```

### Arguments

|               |  |
|---------------|--|
| x             | A spark_connection, ml_pipeline, or a tbl_spark.   |
| storage.level | The storage level to be used. Please view the <a href="#">Spark Documentation</a> for information on what storage levels are accepted. |
| name          | A name to assign this table. Passed to [sdf_register()].   |

### Details

Spark DataFrames invoke their operations lazily – pending operations are deferred until their results are actually needed. Persisting a Spark DataFrame effectively ‘forces’ any pending computations, and then persists the generated Spark DataFrame as requested (to memory, to disk, or otherwise).

Users of Spark should be careful to persist the results of any computations which are non-deterministic – otherwise, one might see that the values within a column seem to ‘change’ as new operations are performed on that data set.

---

sdf\_pivot *Pivot a Spark DataFrame*

---

### Description

Construct a pivot table over a Spark Dataframe, using a syntax similar to that from reshape2::dcast.

### Usage

```
sdf_pivot(x, formula, fun.aggregate = "count")
```

**Arguments**

|               |  |
|---------------|--|
| x             | A spark_connection, ml_pipeline, or a tbl_spark.   |
| formula       | A two-sided R formula of the form $x_1 + x_2 + \dots \sim y_1$ . The left-hand side of the formula indicates which variables are used for grouping, and the right-hand side indicates which variable is used for pivoting. Currently, only a single pivot column is supported. |
| fun.aggregate | How should the grouped dataset be aggregated? Can be a length-one character vector, giving the name of a Spark aggregation function to be called; a named R list mapping column names to an aggregation method, or an R function that is invoked on the grouped dataset.       |

**Examples**

```
## Not run:
library(sparklyr)
library(dplyr)

sc <- spark_connect(master = "local")
iris_tbl <- sdf_copy_to(sc, iris, name = "iris_tbl", overwrite = TRUE)

# aggregating by mean
iris_tbl %>%
  mutate(Petal_Width = ifelse(Petal_Width > 1.5, "High", "Low")) %>%
  sdf_pivot(Petal_Width ~ Species,
    fun.aggregate = list(Petal_Length = "mean")
  )

# aggregating all observations in a list
iris_tbl %>%
  mutate(Petal_Width = ifelse(Petal_Width > 1.5, "High", "Low")) %>%
  sdf_pivot(Petal_Width ~ Species,
    fun.aggregate = list(Petal_Length = "collect_list")
  )

## End(Not run)
```

---

sdf\_project

*Project features onto principal components*


---

**Description**

Project features onto principal components

**Usage**

```
sdf_project(
  object,
  newdata,
  features = dimnames(object$pc)[[1]],
  feature_prefix = NULL,
  ...
)
```

**Arguments**

|                |   |
|----------------|---|
| object         | A Spark PCA model object                      |
| newdata        | An object coercible to a Spark DataFrame      |
| features       | A vector of names of columns to be projected  |
| feature_prefix | The prefix used in naming the output features |
| ...            | Optional arguments; currently unused.         |

---

sdf\_quantile

---

*Compute (Approximate) Quantiles with a Spark DataFrame*


---

**Description**

Given a numeric column within a Spark DataFrame, compute approximate quantiles.

**Usage**

```
sdf_quantile(
  x,
  column,
  probabilities = c(0, 0.25, 0.5, 0.75, 1),
  relative.error = 1e-05,
  weight.column = NULL
)
```

**Arguments**

|                |  |
|----------------|--|
| x              | A spark_connection, ml_pipeline, or a tbl_spark.   |
| column         | The column(s) for which quantiles should be computed. Multiple columns are only supported in Spark 2.0+.   |
| probabilities  | A numeric vector of probabilities, for which quantiles should be computed.   |
| relative.error | The maximal possible difference between the actual percentile of a result and its expected percentile (e.g., if 'relative.error' is 0.01 and 'probabilities' is 0.95, then any value between the 94th and 96th percentile will be considered an acceptable approximation). |



`weight.column` If not NULL, then a generalized version of the Greenwald- Khanna algorithm will be run to compute weighted percentiles, with each sample from ‘column’ having a relative weight specified by the corresponding value in ‘weight.column’. The weights can be considered as relative frequencies of sample data points.

---

sdf\_random\_split      *Partition a Spark Dataframe*

---

## Description

Partition a Spark DataFrame into multiple groups. This routine is useful for splitting a DataFrame into, for example, training and test datasets.

## Usage

```
sdf_random_split(
  x,
  ...,
  weights = NULL,
  seed = sample(.Machine$integer.max, 1)
)

sdf_partition(x, ..., weights = NULL, seed = sample(.Machine$integer.max, 1))
```

## Arguments

|                      |   |
|----------------------|---|
| <code>x</code>       | An object coercable to a Spark DataFrame.   |
| <code>...</code>     | Named parameters, mapping table names to weights. The weights will be normalized such that they sum to 1.                             |
| <code>weights</code> | An alternate mechanism for supplying weights – when specified, this takes precedence over the <code>...</code> arguments.             |
| <code>seed</code>    | Random seed to use for randomly partitioning the dataset. Set this if you want your partitioning to be reproducible on repeated runs. |

## Details

The sampling weights define the probability that a particular observation will be assigned to a particular partition, not the resulting size of the partition. This implies that partitioning a DataFrame with, for example,

```
sdf_random_split(x, training = 0.5, test = 0.5)
```

is not guaranteed to produce training and test partitions of equal size.

## Value

An R list of `tbl_sparks`.

## See Also

Other Spark data frames: [sdf\\_copy\\_to\(\)](#), [sdf\\_distinct\(\)](#), [sdf\\_register\(\)](#), [sdf\\_sample\(\)](#), [sdf\\_sort\(\)](#), [sdf\\_weighted\\_sample\(\)](#)

## Examples

```
## Not run:
# randomly partition data into a 'training' and 'test'
# dataset, with 60% of the observations assigned to the
# 'training' dataset, and 40% assigned to the 'test' dataset
data(diamonds, package = "ggplot2")
diamonds_tbl <- copy_to(sc, diamonds, "diamonds")
partitions <- diamonds_tbl %>%
  sdf_random_split(training = 0.6, test = 0.4)
print(partitions)

# alternate way of specifying weights
weights <- c(training = 0.6, test = 0.4)
diamonds_tbl %>% sdf_random_split(weights = weights)

## End(Not run)
```

---

sdf\_rbeta

*Generate random samples from a Beta distribution*

---

## Description

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from a Beta distribution.

## Usage

```
sdf_rbeta(
  sc,
  n,
  shape1,
  shape2,
  num_partitions = NULL,
  seed = NULL,
  output_col = "x"
)
```

## Arguments

|        |  |
|--------|--|
| sc     | A Spark connection.                                      |
| n      | Sample Size (default: 1000).                             |
| shape1 | Non-negative parameter (alpha) of the Beta distribution. |

|                |  |
|----------------|--|
| shape2         | Non-negative parameter (beta) of the Beta distribution.  |
| num_partitions | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| seed           | Random seed (default: a random long integer).  |
| output_col     | Name of the output column containing sample values (default: "x").   |

**See Also**

Other Spark statistical routines: [sdf\\_rbinom\(\)](#), [sdf\\_rcauchy\(\)](#), [sdf\\_rchisq\(\)](#), [sdf\\_rexp\(\)](#), [sdf\\_rgamma\(\)](#), [sdf\\_rgeom\(\)](#), [sdf\\_rhyper\(\)](#), [sdf\\_rlnorm\(\)](#), [sdf\\_rnorm\(\)](#), [sdf\\_rpois\(\)](#), [sdf\\_rt\(\)](#), [sdf\\_runif\(\)](#), [sdf\\_rweibull\(\)](#)

---

|            |   |
|------------|---|
| sdf_rbinom | <i>Generate random samples from a binomial distribution</i> |
|------------|---|

---

**Description**

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from a binomial distribution.

**Usage**

```
sdf_rbinom(
  sc,
  n,
  size,
  prob,
  num_partitions = NULL,
  seed = NULL,
  output_col = "x"
)
```

**Arguments**

|                |  |
|----------------|--|
| sc             | A Spark connection.  |
| n              | Sample Size (default: 1000).   |
| size           | Number of trials (zero or more).   |
| prob           | Probability of success on each trial.  |
| num_partitions | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| seed           | Random seed (default: a random long integer).  |
| output_col     | Name of the output column containing sample values (default: "x").   |

**See Also**

Other Spark statistical routines: [sdf\\_rbeta\(\)](#), [sdf\\_rcauchy\(\)](#), [sdf\\_rchisq\(\)](#), [sdf\\_rexp\(\)](#), [sdf\\_rgamma\(\)](#), [sdf\\_rgeom\(\)](#), [sdf\\_rhyper\(\)](#), [sdf\\_rlnorm\(\)](#), [sdf\\_rnorm\(\)](#), [sdf\\_rpois\(\)](#), [sdf\\_rt\(\)](#), [sdf\\_runif\(\)](#), [sdf\\_rweibull\(\)](#)

---

sdf\_rcauchy

*Generate random samples from a Cauchy distribution*


---

**Description**

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from a Cauchy distribution.

**Usage**

```
sdf_rcauchy(
  sc,
  n,
  location = 0,
  scale = 1,
  num_partitions = NULL,
  seed = NULL,
  output_col = "x"
)
```

**Arguments**

|                |  |
|----------------|--|
| sc             | A Spark connection.  |
| n              | Sample Size (default: 1000).   |
| location       | Location parameter of the distribution.  |
| scale          | Scale parameter of the distribution.   |
| num_partitions | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| seed           | Random seed (default: a random long integer).  |
| output_col     | Name of the output column containing sample values (default: "x").   |

**See Also**

Other Spark statistical routines: [sdf\\_rbeta\(\)](#), [sdf\\_rbinom\(\)](#), [sdf\\_rchisq\(\)](#), [sdf\\_rexp\(\)](#), [sdf\\_rgamma\(\)](#), [sdf\\_rgeom\(\)](#), [sdf\\_rhyper\(\)](#), [sdf\\_rlnorm\(\)](#), [sdf\\_rnorm\(\)](#), [sdf\\_rpois\(\)](#), [sdf\\_rt\(\)](#), [sdf\\_runif\(\)](#), [sdf\\_rweibull\(\)](#)

---

|            |  |
|------------|--|
| sdf_rchisq | <i>Generate random samples from a chi-squared distribution</i> |
|------------|--|

---

**Description**

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from a chi-squared distribution.

**Usage**

```
sdf_rchisq(sc, n, df, num_partitions = NULL, seed = NULL, output_col = "x")
```

**Arguments**

|                |  |
|----------------|--|
| sc             | A Spark connection.  |
| n              | Sample Size (default: 1000).   |
| df             | Degrees of freedom (non-negative, but can be non-integer).   |
| num_partitions | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| seed           | Random seed (default: a random long integer).  |
| output_col     | Name of the output column containing sample values (default: "x").   |

**See Also**

Other Spark statistical routines: [sdf\\_rbeta\(\)](#), [sdf\\_rbinom\(\)](#), [sdf\\_rcauchy\(\)](#), [sdf\\_rexp\(\)](#), [sdf\\_rgamma\(\)](#), [sdf\\_rgeom\(\)](#), [sdf\\_rhyper\(\)](#), [sdf\\_rlnorm\(\)](#), [sdf\\_rnorm\(\)](#), [sdf\\_rpois\(\)](#), [sdf\\_rt\(\)](#), [sdf\\_runif\(\)](#), [sdf\\_rweibull\(\)](#)

---

|                 |   |
|-----------------|---|
| sdf_read_column | <i>Read a Column from a Spark DataFrame</i> |
|-----------------|---|

---

**Description**

Read a single column from a Spark DataFrame, and return the contents of that column back to R.

**Usage**

```
sdf_read_column(x, column)
```

**Arguments**

|        |  |
|--------|--|
| x      | A spark_connection, ml_pipeline, or a tbl_spark. |
| column | The name of a column within x.                   |

**Details**

It is expected for this operation to preserve row order.

---

|              |                                   |
|--------------|-----------------------------------|
| sdf_register | <i>Register a Spark DataFrame</i> |
|--------------|-----------------------------------|

---

**Description**

Registers a Spark DataFrame (giving it a table name for the Spark SQL context), and returns a tbl\_spark.

**Usage**

```
sdf_register(x, name = NULL)
```

**Arguments**

|      |                              |
|------|------------------------------|
| x    | A Spark DataFrame.           |
| name | A name to assign this table. |

**See Also**

Other Spark data frames: [sdf\\_copy\\_to\(\)](#), [sdf\\_distinct\(\)](#), [sdf\\_random\\_split\(\)](#), [sdf\\_sample\(\)](#), [sdf\\_sort\(\)](#), [sdf\\_weighted\\_sample\(\)](#)

---

|                 |                                      |
|-----------------|--------------------------------------|
| sdf_repartition | <i>Repartition a Spark DataFrame</i> |
|-----------------|--------------------------------------|

---

**Description**

Repartition a Spark DataFrame

**Usage**

```
sdf_repartition(x, partitions = NULL, partition_by = NULL)
```

**Arguments**

|              |   |
|--------------|---|
| x            | A spark_connection, ml_pipeline, or a tbl_spark.                            |
| partitions   | number of partitions  |
| partition_by | vector of column names used for partitioning, only supported for Spark 2.0+ |

---

```
sdf_residuals.ml_model_generalized_linear_regression
```

*Model Residuals*

---

**Description**

This generic method returns a Spark DataFrame with model residuals added as a column to the model training data.

**Usage**

```
## S3 method for class 'ml_model_generalized_linear_regression'
sdf_residuals(
  object,
  type = c("deviance", "pearson", "working", "response"),
  ...
)

## S3 method for class 'ml_model_linear_regression'
sdf_residuals(object, ...)

sdf_residuals(object, ...)
```

**Arguments**

|        |   |
|--------|---|
| object | Spark ML model object.                      |
| type   | type of residuals which should be returned. |
| ...    | additional arguments                        |

---

```
sdf_rexp
```

*Generate random samples from an exponential distribution*

---

**Description**

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from an exponential distribution.

**Usage**

```
sdf_rexp(sc, n, rate = 1, num_partitions = NULL, seed = NULL, output_col = "x")
```

**Arguments**

|                |  |
|----------------|--|
| sc             | A Spark connection.  |
| n              | Sample Size (default: 1000).   |
| rate           | Rate of the exponential distribution (default: 1). The exponential distribution with rate lambda has mean 1 / lambda and density $f(x) = \lambda e^{-\lambda x}$ . |
| num_partitions | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster).   |
| seed           | Random seed (default: a random long integer).  |
| output_col     | Name of the output column containing sample values (default: "x").   |

**See Also**

Other Spark statistical routines: [sdf\\_rbeta\(\)](#), [sdf\\_rbinom\(\)](#), [sdf\\_rcauchy\(\)](#), [sdf\\_rchisq\(\)](#), [sdf\\_rgamma\(\)](#), [sdf\\_rgeom\(\)](#), [sdf\\_rhyper\(\)](#), [sdf\\_rlnorm\(\)](#), [sdf\\_rnorm\(\)](#), [sdf\\_rpois\(\)](#), [sdf\\_rt\(\)](#), [sdf\\_runif\(\)](#), [sdf\\_rweibull\(\)](#)

---

|            |  |
|------------|--|
| sdf_rgamma | <i>Generate random samples from a Gamma distribution</i> |
|------------|--|

---

**Description**

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from a Gamma distribution.

**Usage**

```
sdf_rgamma(
  sc,
  n,
  shape,
  rate = 1,
  num_partitions = NULL,
  seed = NULL,
  output_col = "x"
)
```

**Arguments**

|                |  |
|----------------|--|
| sc             | A Spark connection.  |
| n              | Sample Size (default: 1000).   |
| shape          | Shape parameter (greater than 0) for the Gamma distribution.   |
| rate           | Rate parameter (greater than 0) for the Gamma distribution (scale is 1/rate).                              |
| num_partitions | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| seed           | Random seed (default: a random long integer).  |
| output_col     | Name of the output column containing sample values (default: "x").   |



**See Also**

Other Spark statistical routines: [sdf\\_rbeta\(\)](#), [sdf\\_rbinom\(\)](#), [sdf\\_rcauchy\(\)](#), [sdf\\_rchisq\(\)](#), [sdf\\_rexp\(\)](#), [sdf\\_rgeom\(\)](#), [sdf\\_rhyper\(\)](#), [sdf\\_rlnorm\(\)](#), [sdf\\_rnorm\(\)](#), [sdf\\_rpois\(\)](#), [sdf\\_rt\(\)](#), [sdf\\_runif\(\)](#), [sdf\\_rweibull\(\)](#)

---

`sdf_rgeom`*Generate random samples from a geometric distribution*

---

**Description**

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from a geometric distribution.

**Usage**

```
sdf_rgeom(sc, n, prob, num_partitions = NULL, seed = NULL, output_col = "x")
```

**Arguments**

|                             |  |
|-----------------------------|--|
| <code>sc</code>             | A Spark connection.  |
| <code>n</code>              | Sample Size (default: 1000).   |
| <code>prob</code>           | Probability of success in each trial.  |
| <code>num_partitions</code> | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| <code>seed</code>           | Random seed (default: a random long integer).  |
| <code>output_col</code>     | Name of the output column containing sample values (default: "x").   |

**See Also**

Other Spark statistical routines: [sdf\\_rbeta\(\)](#), [sdf\\_rbinom\(\)](#), [sdf\\_rcauchy\(\)](#), [sdf\\_rchisq\(\)](#), [sdf\\_rexp\(\)](#), [sdf\\_rgamma\(\)](#), [sdf\\_rhyper\(\)](#), [sdf\\_rlnorm\(\)](#), [sdf\\_rnorm\(\)](#), [sdf\\_rpois\(\)](#), [sdf\\_rt\(\)](#), [sdf\\_runif\(\)](#), [sdf\\_rweibull\(\)](#)

---

`sdf_rhyper`*Generate random samples from a hypergeometric distribution*

---

### Description

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from a hypergeometric distribution.

### Usage

```
sdf_rhyper(  
  sc,  
  nn,  
  m,  
  n,  
  k,  
  num_partitions = NULL,  
  seed = NULL,  
  output_col = "x"  
)
```

### Arguments

|                             |  |
|-----------------------------|--|
| <code>sc</code>             | A Spark connection.  |
| <code>nn</code>             | Sample Size.   |
| <code>m</code>              | The number of successes among the population.  |
| <code>n</code>              | The number of failures among the population.   |
| <code>k</code>              | The number of draws.   |
| <code>num_partitions</code> | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| <code>seed</code>           | Random seed (default: a random long integer).  |
| <code>output_col</code>     | Name of the output column containing sample values (default: "x").   |

### See Also

Other Spark statistical routines: [sdf\\_rbeta\(\)](#), [sdf\\_rbinom\(\)](#), [sdf\\_rcauchy\(\)](#), [sdf\\_rchisq\(\)](#), [sdf\\_rexp\(\)](#), [sdf\\_rgamma\(\)](#), [sdf\\_rgeom\(\)](#), [sdf\\_rlnorm\(\)](#), [sdf\\_rnorm\(\)](#), [sdf\\_rpois\(\)](#), [sdf\\_rt\(\)](#), [sdf\\_runif\(\)](#), [sdf\\_rweibull\(\)](#)

---

|            |   |
|------------|---|
| sdf_rlnorm | <i>Generate random samples from a log normal distribution</i> |
|------------|---|

---

### Description

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from a log normal distribution.

### Usage

```
sdf_rlnorm(  
  sc,  
  n,  
  meanlog = 0,  
  sdlog = 1,  
  num_partitions = NULL,  
  seed = NULL,  
  output_col = "x"  
)
```

### Arguments

|                |  |
|----------------|--|
| sc             | A Spark connection.  |
| n              | Sample Size (default: 1000).   |
| meanlog        | The mean of the normally distributed natural logarithm of this distribution.                               |
| sdlog          | The Standard deviation of the normally distributed natural logarithm of this distribution.                 |
| num_partitions | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| seed           | Random seed (default: a random long integer).  |
| output_col     | Name of the output column containing sample values (default: "x").   |

### See Also

Other Spark statistical routines: [sdf\\_rbeta\(\)](#), [sdf\\_rbinom\(\)](#), [sdf\\_rcauchy\(\)](#), [sdf\\_rchisq\(\)](#), [sdf\\_rexp\(\)](#), [sdf\\_rgamma\(\)](#), [sdf\\_rgeom\(\)](#), [sdf\\_rhyper\(\)](#), [sdf\\_rnorm\(\)](#), [sdf\\_rpois\(\)](#), [sdf\\_rt\(\)](#), [sdf\\_runif\(\)](#), [sdf\\_rweibull\(\)](#)

---

`sdf_rnorm`*Generate random samples from the standard normal distribution*

---

### Description

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from the standard normal distribution.

### Usage

```
sdf_rnorm(  
  sc,  
  n,  
  mean = 0,  
  sd = 1,  
  num_partitions = NULL,  
  seed = NULL,  
  output_col = "x"  
)
```

### Arguments

|                             |  |
|-----------------------------|--|
| <code>sc</code>             | A Spark connection.  |
| <code>n</code>              | Sample Size (default: 1000).   |
| <code>mean</code>           | The mean value of the normal distribution.   |
| <code>sd</code>             | The standard deviation of the normal distribution.   |
| <code>num_partitions</code> | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| <code>seed</code>           | Random seed (default: a random long integer).  |
| <code>output_col</code>     | Name of the output column containing sample values (default: "x").   |

### See Also

Other Spark statistical routines: [sdf\\_rbeta\(\)](#), [sdf\\_rbinom\(\)](#), [sdf\\_rcauchy\(\)](#), [sdf\\_rchisq\(\)](#), [sdf\\_rexp\(\)](#), [sdf\\_rgamma\(\)](#), [sdf\\_rgeom\(\)](#), [sdf\\_rhyper\(\)](#), [sdf\\_rlnorm\(\)](#), [sdf\\_rpois\(\)](#), [sdf\\_rt\(\)](#), [sdf\\_runif\(\)](#), [sdf\\_rweibull\(\)](#)

---

sdf\_rpois                      *Generate random samples from a Poisson distribution*

---

### Description

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from a Poisson distribution.

### Usage

```
sdf_rpois(sc, n, lambda, num_partitions = NULL, seed = NULL, output_col = "x")
```

### Arguments

|                |  |
|----------------|--|
| sc             | A Spark connection.  |
| n              | Sample Size (default: 1000).   |
| lambda         | Mean, or lambda, of the Poisson distribution.  |
| num_partitions | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| seed           | Random seed (default: a random long integer).  |
| output_col     | Name of the output column containing sample values (default: "x").   |

### See Also

Other Spark statistical routines: [sdf\\_rbeta\(\)](#), [sdf\\_rbinom\(\)](#), [sdf\\_rcauchy\(\)](#), [sdf\\_rchisq\(\)](#), [sdf\\_rexp\(\)](#), [sdf\\_rgamma\(\)](#), [sdf\\_rgeom\(\)](#), [sdf\\_rhyper\(\)](#), [sdf\\_rlnorm\(\)](#), [sdf\\_rnorm\(\)](#), [sdf\\_rt\(\)](#), [sdf\\_runif\(\)](#), [sdf\\_rweibull\(\)](#)

---

sdf\_rt                              *Generate random samples from a t-distribution*

---

### Description

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from a t-distribution.

### Usage

```
sdf_rt(sc, n, df, num_partitions = NULL, seed = NULL, output_col = "x")
```

**Arguments**

|                |  |
|----------------|--|
| sc             | A Spark connection.  |
| n              | Sample Size (default: 1000).   |
| df             | Degrees of freedom (> 0, maybe non-integer).   |
| num_partitions | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| seed           | Random seed (default: a random long integer).  |
| output_col     | Name of the output column containing sample values (default: "x").   |

**See Also**

Other Spark statistical routines: [sdf\\_rbeta\(\)](#), [sdf\\_rbinom\(\)](#), [sdf\\_rcauchy\(\)](#), [sdf\\_rchisq\(\)](#), [sdf\\_rexp\(\)](#), [sdf\\_rgamma\(\)](#), [sdf\\_rgeom\(\)](#), [sdf\\_rhyper\(\)](#), [sdf\\_rlnorm\(\)](#), [sdf\\_rnorm\(\)](#), [sdf\\_rpois\(\)](#), [sdf\\_runif\(\)](#), [sdf\\_rweibull\(\)](#)

---

sdf\_runif

*Generate random samples from the uniform distribution U(0, 1).*


---

**Description**

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from the uniform distribution U(0, 1).

**Usage**

```
sdf_runif(
  sc,
  n,
  min = 0,
  max = 1,
  num_partitions = NULL,
  seed = NULL,
  output_col = "x"
)
```

**Arguments**

|                |  |
|----------------|--|
| sc             | A Spark connection.  |
| n              | Sample Size (default: 1000).   |
| min            | The lower limit of the distribution.   |
| max            | The upper limit of the distribution.   |
| num_partitions | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| seed           | Random seed (default: a random long integer).  |
| output_col     | Name of the output column containing sample values (default: "x").   |

**See Also**

Other Spark statistical routines: [sdf\\_rbeta\(\)](#), [sdf\\_rbinom\(\)](#), [sdf\\_rcauchy\(\)](#), [sdf\\_rchisq\(\)](#), [sdf\\_rexp\(\)](#), [sdf\\_rgamma\(\)](#), [sdf\\_rgeom\(\)](#), [sdf\\_rhyper\(\)](#), [sdf\\_rlnorm\(\)](#), [sdf\\_rnorm\(\)](#), [sdf\\_rpois\(\)](#), [sdf\\_rt\(\)](#), [sdf\\_rweibull\(\)](#)

---

|              |   |
|--------------|---|
| sdf_rweibull | <i>Generate random samples from a Weibull distribution.</i> |
|--------------|---|

---

**Description**

Generator method for creating a single-column Spark dataframes comprised of i.i.d. samples from a Weibull distribution.

**Usage**

```
sdf_rweibull(  
  sc,  
  n,  
  shape,  
  scale = 1,  
  num_partitions = NULL,  
  seed = NULL,  
  output_col = "x"  
)
```

**Arguments**

|                |  |
|----------------|--|
| sc             | A Spark connection.  |
| n              | Sample Size (default: 1000).   |
| shape          | The shape of the Weibull distribution.   |
| scale          | The scale of the Weibull distribution (default: 1).  |
| num_partitions | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| seed           | Random seed (default: a random long integer).  |
| output_col     | Name of the output column containing sample values (default: "x").   |

**See Also**

Other Spark statistical routines: [sdf\\_rbeta\(\)](#), [sdf\\_rbinom\(\)](#), [sdf\\_rcauchy\(\)](#), [sdf\\_rchisq\(\)](#), [sdf\\_rexp\(\)](#), [sdf\\_rgamma\(\)](#), [sdf\\_rgeom\(\)](#), [sdf\\_rhyper\(\)](#), [sdf\\_rlnorm\(\)](#), [sdf\\_rnorm\(\)](#), [sdf\\_rpois\(\)](#), [sdf\\_rt\(\)](#), [sdf\\_runif\(\)](#)

---

|            |  |
|------------|--|
| sdf_sample | <i>Randomly Sample Rows from a Spark DataFrame</i> |
|------------|--|

---

**Description**

Draw a random sample of rows (with or without replacement) from a Spark DataFrame.

**Usage**

```
sdf_sample(x, fraction = 1, replacement = TRUE, seed = NULL)
```

**Arguments**

|             |   |
|-------------|---|
| x           | An object coercable to a Spark DataFrame. |
| fraction    | The fraction to sample.                   |
| replacement | Boolean; sample with replacement?         |
| seed        | An (optional) integer seed.               |

**See Also**

Other Spark data frames: [sdf\\_copy\\_to\(\)](#), [sdf\\_distinct\(\)](#), [sdf\\_random\\_split\(\)](#), [sdf\\_register\(\)](#), [sdf\\_sort\(\)](#), [sdf\\_weighted\\_sample\(\)](#)

---

|            |   |
|------------|---|
| sdf_schema | <i>Read the Schema of a Spark DataFrame</i> |
|------------|---|

---

**Description**

Read the schema of a Spark DataFrame.

**Usage**

```
sdf_schema(x, expand_nested_cols = FALSE, expand_struct_cols = FALSE)
```

**Arguments**

|                    |  |
|--------------------|--|
| x                  | A spark_connection, ml_pipeline, or a tbl_spark.   |
| expand_nested_cols | Whether to expand columns containing nested array of structs (which are usually created by tidy::nest on a Spark data frame) |
| expand_struct_cols | Whether to expand columns containing structs   |



**Details**

The type column returned gives the string representation of the underlying Spark type for that column; for example, a vector of numeric values would be returned with the type "DoubleType". Please see the [Spark Scala API Documentation](#) for information on what types are available and exposed by Spark.

**Value**

An R list, with each list element describing the name and type of a column.

---

|                     |   |
|---------------------|---|
| sdf_separate_column | <i>Separate a Vector Column into Scalar Columns</i> |
|---------------------|---|

---

**Description**

Given a vector column in a Spark DataFrame, split that into n separate columns, each column made up of the different elements in the column column.

**Usage**

```
sdf_separate_column(x, column, into = NULL)
```

**Arguments**

|        |  |
|--------|--|
| x      | A spark_connection, ml_pipeline, or a tbl_spark.   |
| column | The name of a (vector-typed) column.   |
| into   | A specification of the columns that should be generated from column. This can either be a vector of column names, or an R list mapping column names to the (1-based) index at which a particular vector element should be extracted. |

---

|         |                                   |
|---------|-----------------------------------|
| sdf_seq | <i>Create DataFrame for Range</i> |
|---------|-----------------------------------|

---

**Description**

Creates a DataFrame for the given range

**Usage**

```
sdf_seq(
  sc,
  from = 1L,
  to = 1L,
  by = 1L,
  repartition = NULL,
  type = c("integer", "integer64")
)
```

**Arguments**

|             |  |
|-------------|--|
| sc          | The associated Spark connection.   |
| from, to    | The start and end to use as a range  |
| by          | The increment of the sequence.   |
| repartition | The number of partitions to use when distributing the data across the Spark cluster. Defaults to the minimum number of partitions. |
| type        | The data type to use for the index, either "integer" or "integer64".   |

---

|          |                               |
|----------|-------------------------------|
| sdf_sort | <i>Sort a Spark DataFrame</i> |
|----------|-------------------------------|

---

**Description**

Sort a Spark DataFrame by one or more columns, with each column sorted in ascending order.

**Usage**

```
sdf_sort(x, columns)
```

**Arguments**

|         |   |
|---------|---|
| x       | An object coercable to a Spark DataFrame. |
| columns | The column(s) to sort by.                 |

**See Also**

Other Spark data frames: [sdf\\_copy\\_to\(\)](#), [sdf\\_distinct\(\)](#), [sdf\\_random\\_split\(\)](#), [sdf\\_register\(\)](#), [sdf\\_sample\(\)](#), [sdf\\_weighted\\_sample\(\)](#)

---

|         |                                 |
|---------|---------------------------------|
| sdf_sql | <i>Spark DataFrame from SQL</i> |
|---------|---------------------------------|

---

**Description**

Defines a Spark DataFrame from a SQL query, useful to create Spark DataFrames without collecting the results immediately.

**Usage**

```
sdf_sql(sc, sql)
```

**Arguments**

|     |   |
|-----|---|
| sc  | A spark_connection.                               |
| sql | a 'SQL' query used to generate a Spark DataFrame. |

---

|             |   |
|-------------|---|
| sdf_to_avro | <i>Convert column(s) to avro format</i> |
|-------------|---|

---

**Description**

Convert column(s) to avro format

**Usage**

```
sdf_to_avro(x, cols = colnames(x))
```

**Arguments**

|      |   |
|------|---|
| x    | An object coercible to a Spark DataFrame      |
| cols | Subset of Columns to convert into avro format |

---

|                   |                      |
|-------------------|----------------------|
| sdf_unnest_longer | <i>Unnest longer</i> |
|-------------------|----------------------|

---

**Description**

Expand a struct column or an array column within a Spark dataframe into one or more rows, similar what to `tidyr::unnest_longer` does to an R dataframe. An index column, if included, will be 1-based if 'col' is an array column.

**Usage**

```
sdf_unnest_longer(
  data,
  col,
  values_to = NULL,
  indices_to = NULL,
  include_indices = NULL,
  names_repair = "check_unique",
  ptype = list(),
  transform = list()
)
```

**Arguments**

|            |   |
|------------|---|
| data       | The Spark dataframe to be unnested  |
| col        | The struct column to extract components from  |
| values_to  | Name of column to store vector values. Defaults to 'col'.   |
| indices_to | A string giving the name of column which will contain the inner names or position (if not named) of the values. Defaults to 'col' with '_id' suffix |

|                 |  |
|-----------------|--|
| include_indices | Whether to include an index column. An index column will be included by default if 'col' is a struct column. It will also be included if 'indices_to' is not 'NULL'.   |
| names_repair    | Strategy for fixing duplicate column names (the semantic will be exactly identical to that of '.name_repair' option in <a href="#">tibble</a> )  |
| ptype           | Optionally, supply an R data frame prototype for the output. Each column of the unnested result will be casted based on the Spark equivalent of the type of the column with the same name within 'ptype', e.g., if 'ptype' has a column 'x' of type 'character', then column 'x' of the unnested result will be casted from its original SQL type to StringType. |
| transform       | Optionally, a named list of transformation functions applied   |

### Examples

```
## Not run:
library(sparklyr)
sc <- spark_connect(master = "local", version = "2.4.0")

# unnesting a struct column
sdf <- copy_to(
  sc,
  dplyr::tibble(
    x = 1:3,
    y = list(list(a = 1, b = 2), list(a = 3, b = 4), list(a = 5, b = 6))
  )
)

unnested <- sdf %>% sdf_unnest_longer(y, indices_to = "attr")

# unnesting an array column
sdf <- copy_to(
  sc,
  dplyr::tibble(
    x = 1:3,
    y = list(1:10, 1:5, 1:2)
  )
)

unnested <- sdf %>% sdf_unnest_longer(y, indices_to = "array_idx")

## End(Not run)
```

**Description**

Flatten a struct column within a Spark dataframe into one or more columns, similar what to `tidyr::unnest_wider` does to an R dataframe

**Usage**

```
sdf_unnest_wider(
  data,
  col,
  names_sep = NULL,
  names_repair = "check_unique",
  ptype = list(),
  transform = list()
)
```

**Arguments**

|                           |  |
|---------------------------|--|
| <code>data</code>         | The Spark dataframe to be unnested   |
| <code>col</code>          | The struct column to extract components from   |
| <code>names_sep</code>    | If 'NULL', the default, the names will be left as is. If a string, the inner and outer names will be pasted together using 'names_sep' as the delimiter.   |
| <code>names_repair</code> | Strategy for fixing duplicate column names (the semantic will be exactly identical to that of 'name_repair' option in <a href="#">tibble</a> )   |
| <code>ptype</code>        | Optionally, supply an R data frame prototype for the output. Each column of the unnested result will be casted based on the Spark equivalent of the type of the column with the same name within 'ptype', e.g., if 'ptype' has a column 'x' of type 'character', then column 'x' of the unnested result will be casted from its original SQL type to StringType. |
| <code>transform</code>    | Optionally, a named list of transformation functions applied to each component (e.g., <code>list('x = as.character')</code> to cast column 'x' to String).   |

**Examples**

```
## Not run:
library(sparklyr)
sc <- spark_connect(master = "local", version = "2.4.0")

sdf <- copy_to(
  sc,
  dplyr::tibble(
    x = 1:3,
    y = list(list(a = 1, b = 2), list(a = 3, b = 4), list(a = 5, b = 6))
  )
)

# flatten struct column 'y' into two separate columns 'y_a' and 'y_b'
unnested <- sdf %>% sdf_unnest_wider(y, names_sep = "_")
```

```
## End(Not run)
```

---

sdf\_weighted\_sample     *Perform Weighted Random Sampling on a Spark DataFrame*

---

### Description

Draw a random sample of rows (with or without replacement) from a Spark DataFrame. If the sampling is done without replacement, then it will be conceptually equivalent to an iterative process such that in each step the probability of adding a row to the sample set is equal to its weight divided by summation of weights of all rows that are not in the sample set yet in that step.

### Usage

```
sdf_weighted_sample(x, weight_col, k, replacement = TRUE, seed = NULL)
```

### Arguments

|             |   |
|-------------|---|
| x           | An object coercable to a Spark DataFrame. |
| weight_col  | Name of the weight column                 |
| k           | Sample set size                           |
| replacement | Whether to sample with replacement        |
| seed        | An (optional) integer seed                |

### See Also

Other Spark data frames: [sdf\\_copy\\_to\(\)](#), [sdf\\_distinct\(\)](#), [sdf\\_random\\_split\(\)](#), [sdf\\_register\(\)](#), [sdf\\_sample\(\)](#), [sdf\\_sort\(\)](#)

---

sdf\_with\_sequential\_id

*Add a Sequential ID Column to a Spark DataFrame*

---

### Description

Add a sequential ID column to a Spark DataFrame. The Spark `zipWithIndex` function is used to produce these. This differs from `sdf_with_unique_id` in that the IDs generated are independent of partitioning.

### Usage

```
sdf_with_sequential_id(x, id = "id", from = 1L)
```

**Arguments**

|      |   |
|------|---|
| x    | A spark_connection, ml_pipeline, or a tbl_spark.  |
| id   | The name of the column to host the generated IDs. |
| from | The starting value of the id column               |

---

|                    |  |
|--------------------|--|
| sdf_with_unique_id | <i>Add a Unique ID Column to a Spark DataFrame</i> |
|--------------------|--|

---

**Description**

Add a unique ID column to a Spark DataFrame. The Spark `monotonicallyIncreasingId` function is used to produce these and is guaranteed to produce unique, monotonically increasing ids; however, there is no guarantee that these IDs will be sequential. The table is persisted immediately after the column is generated, to ensure that the column is stable – otherwise, it can differ across new computations.

**Usage**

```
sdf_with_unique_id(x, id = "id")
```

**Arguments**

|    |   |
|----|---|
| x  | A spark_connection, ml_pipeline, or a tbl_spark.  |
| id | The name of the column to host the generated IDs. |

---

|        |               |
|--------|---------------|
| select | <i>Select</i> |
|--------|---------------|

---

**Description**

See [select](#) for more details.

---

|          |                 |
|----------|-----------------|
| separate | <i>Separate</i> |
|----------|-----------------|

---

**Description**

See [separate](#) for more details.

---

`spark-api`*Access the Spark API*

---

## Description

Access the commonly-used Spark objects associated with a Spark instance. These objects provide access to different facets of the Spark API.

## Usage

`spark_context(sc)``java_context(sc)``hive_context(sc)``spark_session(sc)`

## Arguments

|                 |                                   |
|-----------------|-----------------------------------|
| <code>sc</code> | A <code>spark_connection</code> . |
|-----------------|-----------------------------------|

## Details

The [Scala API documentation](#) is useful for discovering what methods are available for each of these objects. Use [invoke](#) to call methods on these objects.

## Spark Context

The main entry point for Spark functionality. The **Spark Context** represents the connection to a Spark cluster, and can be used to create RDDs, accumulators and broadcast variables on that cluster.

## Java Spark Context

A Java-friendly version of the aforementioned **Spark Context**.

## Hive Context

An instance of the Spark SQL execution engine that integrates with data stored in Hive. Configuration for Hive is read from `hive-site.xml` on the classpath.

Starting with Spark  $\geq$  2.0.0, the **Hive Context** class has been deprecated – it is superseded by the **Spark Session** class, and `hive_context` will return a **Spark Session** object instead. Note that both classes share a SQL interface, and therefore one can invoke SQL through these objects.



## Spark Session

Available since Spark 2.0.0, the **Spark Session** unifies the **Spark Context** and **Hive Context** classes into a single interface. Its use is recommended over the older APIs for code targeting Spark 2.0.0 and above.

---

spark-connections      *Manage Spark Connections*

---

## Description

These routines allow you to manage your connections to Spark.

Call 'spark\_disconnect()' on each open Spark connection

## Usage

```
spark_connect(  
  master,  
  spark_home = Sys.getenv("SPARK_HOME"),  
  method = c("shell", "livy", "databricks", "test", "qubole", "synapse"),  
  app_name = "sparklyr",  
  version = NULL,  
  config = spark_config(),  
  extensions = sparklyr::registered_extensions(),  
  packages = NULL,  
  scala_version = NULL,  
  ...  
)  
  
spark_connection_is_open(sc)  
  
spark_disconnect(sc, ...)  
  
spark_disconnect_all(...)  
  
spark_submit(  
  master,  
  file,  
  spark_home = Sys.getenv("SPARK_HOME"),  
  app_name = "sparklyr",  
  version = NULL,  
  config = spark_config(),  
  extensions = sparklyr::registered_extensions(),  
  scala_version = NULL,  
  ...  
)
```

**Arguments**

|               |  |
|---------------|--|
| master        | Spark cluster url to connect to. Use "local" to connect to a local instance of Spark installed via <a href="#">spark_install</a> .   |
| spark_home    | The path to a Spark installation. Defaults to the path provided by the SPARK_HOME environment variable. If SPARK_HOME is defined, it will always be used unless the version parameter is specified to force the use of a locally installed version.  |
| method        | The method used to connect to Spark. Default connection method is "shell" to connect using spark-submit, use "livy" to perform remote connections using HTTP, or "databricks" when using a Databricks clusters.  |
| app_name      | The application name to be used while running in the Spark cluster.  |
| version       | The version of Spark to use. Required for "local" Spark connections, optional otherwise.   |
| config        | Custom configuration for the generated Spark connection. See <a href="#">spark_config</a> for details.   |
| extensions    | Extension R packages to enable for this connection. By default, all packages enabled through the use of <a href="#">sparklyr::register_extension</a> will be passed here.  |
| packages      | A list of Spark packages to load. For example, "delta" or "kafka" to enable Delta Lake or Kafka. Also supports full versions like "io.delta:delta-core_2.11:0.4.0". This is similar to adding packages into the sparklyr.shell.packages configuration option. Notice that the version parameter is used to choose the correct package, otherwise assumes the latest version is being used. |
| scala_version | Load the sparklyr jar file that is built with the version of Scala specified (this currently only makes sense for Spark 2.4, where sparklyr will by default assume Spark 2.4 on current host is built with Scala 2.11, and therefore 'scala_version = '2.12'' is needed if sparklyr is connecting to Spark 2.4 built with Scala 2.12)  |
| ...           | Additional params to be passed to each 'spark_disconnect()' call (e.g., 'terminate = TRUE')  |
| sc            | A spark_connection.  |
| file          | Path to R source file to submit for batch execution.   |

**Details**

By default, when using method = "livy", jars are downloaded from GitHub. But an alternative path (local to Livy server or on HDFS or HTTP(s)) to sparklyr JAR can also be specified through the sparklyr.livy.jar setting.

**Examples**

```
conf <- spark_config()
conf$`sparklyr.shell.conf` <- c(
  "spark.executor.extraJavaOptions=-Duser.timezone='UTC'",
  "spark.driver.extraJavaOptions=-Duser.timezone='UTC'",
  "spark.sql.session.timeZone='UTC'"
)
```

```
sc <- spark_connect(  
  master = "spark://HOST:PORT", config = conf  
)  
connection_is_open(sc)  
  
spark_disconnect(sc)
```

---

sparklyr\_get\_backend\_port

*Return the port number of a 'sparklyr' backend.*

---

### Description

Retrieve the port number of the 'sparklyr' backend associated with a Spark connection.

### Usage

```
sparklyr_get_backend_port(sc)
```

### Arguments

sc                    A spark\_connection.

### Value

The port number of the 'sparklyr' backend associated with sc.

---

spark\_adaptive\_query\_execution

*Retrieves or sets status of Spark AQE*

---

### Description

Retrieves or sets whether Spark adaptive query execution is enabled

### Usage

```
spark_adaptive_query_execution(sc, enable = NULL)
```

### Arguments

sc                    A spark\_connection.

enable                Whether to enable Spark adaptive query execution. Defaults to NULL to retrieve configuration entries.

**See Also**

Other Spark runtime configuration: [spark\\_advisory\\_shuffle\\_partition\\_size\(\)](#), [spark\\_auto\\_broadcast\\_join\\_thres](#), [spark\\_coalesce\\_initial\\_num\\_partitions\(\)](#), [spark\\_coalesce\\_min\\_num\\_partitions\(\)](#), [spark\\_coalesce\\_shuffle\\_](#), [spark\\_session\\_config\(\)](#)

---

spark\_advisory\_shuffle\_partition\_size

*Retrieves or sets advisory size of the shuffle partition*

---

**Description**

Retrieves or sets advisory size in bytes of the shuffle partition during adaptive optimization

**Usage**

```
spark_advisory_shuffle_partition_size(sc, size = NULL)
```

**Arguments**

|      |  |
|------|--|
| sc   | A spark_connection.  |
| size | Advisory size in bytes of the shuffle partition. Defaults to NULL to retrieve configuration entries. |

**See Also**

Other Spark runtime configuration: [spark\\_adaptive\\_query\\_execution\(\)](#), [spark\\_auto\\_broadcast\\_join\\_threshold\(\)](#), [spark\\_coalesce\\_initial\\_num\\_partitions\(\)](#), [spark\\_coalesce\\_min\\_num\\_partitions\(\)](#), [spark\\_coalesce\\_shuffle\\_](#), [spark\\_session\\_config\(\)](#)

---

spark\_apply

*Apply an R Function in Spark*

---

**Description**

Applies an R function to a Spark object (typically, a Spark DataFrame).

**Usage**

```

spark_apply(
  x,
  f,
  columns = NULL,
  memory = TRUE,
  group_by = NULL,
  packages = NULL,
  context = NULL,
  name = NULL,
  barrier = NULL,
  fetch_result_as_sdf = TRUE,
  partition_index_param = "",
  arrow_max_records_per_batch = NULL,
  auto_deps = FALSE,
  ...
)

```

**Arguments**

|          |   |
|----------|---|
| x        | An object (usually a <code>spark_tbl</code> ) coercable to a Spark DataFrame.   |
| f        | A function that transforms a data frame partition into a data frame. The function <code>f</code> has signature <code>f(df, context, group1, group2, ...)</code> where <code>df</code> is a data frame with the data to be processed, <code>context</code> is an optional object passed as the <code>context</code> parameter and <code>group1</code> to <code>groupN</code> contain the values of the <code>group_by</code> values. When <code>group_by</code> is not specified, <code>f</code> takes only one argument.<br>Can also be an <code>rlang</code> anonymous function. For example, as <code>~ .x + 1</code> to define an expression that adds one to the given <code>.x</code> data frame.  |
| columns  | A vector of column names or a named vector of column types for the transformed object. When not specified, a sample of 10 rows is taken to infer out the output columns automatically, to avoid this performance penalty, specify the column types. The sample size is configurable using the <code>sparklyr.apply.schema.infer</code> configuration option.  |
| memory   | Boolean; should the table be cached into memory?  |
| group_by | Column name used to group by data frame partitions.   |
| packages | Boolean to distribute <code>.libPaths()</code> packages to each node, a list of packages to distribute, or a package bundle created with <code>spark_apply_bundle()</code> .<br>Defaults to <code>TRUE</code> or the <code>sparklyr.apply.packages</code> value set in <code>spark_config()</code> .<br>For clusters using Yarn cluster mode, packages can point to a package bundle created using <code>spark_apply_bundle()</code> and made available as a Spark file using <code>config\$sparklyr.shell.files</code> . For clusters using Livy, packages can be manually installed on the driver node.<br>For offline clusters where <code>available.packages()</code> is not available, manually download the packages database from <a href="https://cran.r-project.org/web/packages/packages.rds">https://cran.r-project.org/web/packages/packages.rds</a> and set <code>Sys.setenv(sparklyr.apply.packagesdb = "&lt;path1-to-rds&gt;")</code> . Otherwise, all packages will be used by default. |

|  |   |
|--|---|
|  | For clusters where R packages already installed in every worker node, the <code>spark.r.libpaths</code> config entry can be set in <code>spark_config()</code> to the local packages library. To specify multiple paths collapse them (without spaces) with a comma delimiter (e.g., <code>"/lib/path/one,/lib/path/two"</code> ).  |
| <code>context</code>                     | Optional object to be serialized and passed back to <code>f()</code> .  |
| <code>name</code>                        | Optional table name while registering the resulting data frame.   |
| <code>barrier</code>                     | Optional to support Barrier Execution Mode in the scheduler.  |
| <code>fetch_result_as_sdf</code>         | Whether to return the transformed results in a Spark Dataframe (defaults to <code>TRUE</code> ). When set to <code>FALSE</code> , results will be returned as a list of R objects instead. NOTE: <code>fetch_result_as_sdf</code> must be set to <code>FALSE</code> when the transformation function being applied is returning R objects that cannot be stored in a Spark Dataframe (e.g., complex numbers or any other R data type that does not have an equivalent representation among Spark SQL data types).   |
| <code>partition_index_param</code>       | Optional if non-empty, then <code>f</code> also receives the index of the partition being processed as a named argument with this name, in addition to all positional argument(s) it will receive<br>NOTE: when <code>fetch_result_as_sdf</code> is set to <code>FALSE</code> , object returned from the transformation function also must be serializable by the <code>base::serialize</code> function in R.   |
| <code>arrow_max_records_per_batch</code> | Maximum size of each Arrow record batch, ignored if Arrow serialization is not enabled.   |
| <code>auto_deps</code>                   | [Experimental] Whether to infer all required R packages by examining the closure <code>f()</code> and only distribute required R and their transitive dependencies to Spark worker nodes (default: <code>FALSE</code> ). NOTE: this option will only take effect if <code>packages</code> is set to <code>TRUE</code> or is a character vector of R package names. If <code>packages</code> is a character vector of R package names, then both the set of packages specified by <code>packages</code> and the set of inferred packages will be distributed to Spark workers. |
| <code>...</code>                         | Optional arguments; currently unused.   |

## Configuration

`spark_config()` settings can be specified to change the workers environment.

For instance, to set additional environment variables to each worker node use the `sparklyr.apply.env.*config`, to launch workers without `--vanilla` use `sparklyr.apply.options.vanilla` set to `FALSE`, to run a custom script before launching Rscript use `sparklyr.apply.options.rscript.before`.

## Examples

```
## Not run:

library(sparklyr)
sc <- spark_connect(master = "local[3]")
```

```
# creates an Spark data frame with 10 elements then multiply times 10 in R
sdf_len(sc, 10) %>% spark_apply(function(df) df * 10)

# using barrier mode
sdf_len(sc, 3, repartition = 3) %>%
  spark_apply(nrow, barrier = TRUE, columns = c(id = "integer")) %>%
  collect()

## End(Not run)
```

---

spark\_apply\_bundle      *Create Bundle for Spark Apply*

---

### Description

Creates a bundle of packages for spark\_apply().

### Usage

```
spark_apply_bundle(packages = TRUE, base_path = getwd(), session_id = NULL)
```

### Arguments

|            |   |
|------------|---|
| packages   | List of packages to pack or TRUE to pack all.   |
| base_path  | Base path used to store the resulting bundle.   |
| session_id | An optional ID string to include in the bundle file name to allow the bundle to be session-specific |

---

spark\_apply\_log      *Log Writer for Spark Apply*

---

### Description

Writes data to log under spark\_apply().

### Usage

```
spark_apply_log(..., level = "INFO")
```

### Arguments

|       |   |
|-------|---|
| ...   | Arguments to write to log.  |
| level | Severity level for this entry; recommended values: INFO, ERROR or WARN. |

---

spark\_auto\_broadcast\_join\_threshold

*Retrieves or sets the auto broadcast join threshold*

---

### Description

Configures the maximum size in bytes for a table that will be broadcast to all worker nodes when performing a join. By setting this value to -1 broadcasting can be disabled. Note that currently statistics are only supported for Hive Metastore tables where the command ‘ANALYZE TABLE <tableName> COMPUTE STATISTICS noscan’ has been run, and file-based data source tables where the statistics are computed directly on the files of data.

### Usage

spark\_auto\_broadcast\_join\_threshold(sc, threshold = NULL)

### Arguments

|           |  |
|-----------|--|
| sc        | A spark_connection.  |
| threshold | Maximum size in bytes for a table that will be broadcast to all worker nodes when performing a join. Defaults to NULL to retrieve configuration entries. |

### See Also

Other Spark runtime configuration: [spark\\_adaptive\\_query\\_execution\(\)](#), [spark\\_advisory\\_shuffle\\_partition\\_size\(\)](#), [spark\\_coalesce\\_initial\\_num\\_partitions\(\)](#), [spark\\_coalesce\\_min\\_num\\_partitions\(\)](#), [spark\\_coalesce\\_shuffle\\_partitions\(\)](#), [spark\\_session\\_config\(\)](#)

---

spark\_coalesce\_initial\_num\_partitions

*Retrieves or sets initial number of shuffle partitions before coalescing*

---

### Description

Retrieves or sets initial number of shuffle partitions before coalescing

### Usage

spark\_coalesce\_initial\_num\_partitions(sc, num\_partitions = NULL)

### Arguments

|                |   |
|----------------|---|
| sc             | A spark_connection.   |
| num_partitions | Initial number of shuffle partitions before coalescing. Defaults to NULL to retrieve configuration entries. |



**See Also**

Other Spark runtime configuration: [spark\\_adaptive\\_query\\_execution\(\)](#), [spark\\_advisory\\_shuffle\\_partition\\_size\(\)](#), [spark\\_auto\\_broadcast\\_join\\_threshold\(\)](#), [spark\\_coalesce\\_min\\_num\\_partitions\(\)](#), [spark\\_coalesce\\_shuffle\\_partitions\(\)](#), [spark\\_session\\_config\(\)](#)

---

spark\_coalesce\_min\_num\_partitions

*Retrieves or sets the minimum number of shuffle partitions after coalescing*

---

**Description**

Retrieves or sets the minimum number of shuffle partitions after coalescing

**Usage**

```
spark_coalesce_min_num_partitions(sc, num_partitions = NULL)
```

**Arguments**

sc A spark\_connection.

num\_partitions Minimum number of shuffle partitions after coalescing. Defaults to NULL to retrieve configuration entries.

**See Also**

Other Spark runtime configuration: [spark\\_adaptive\\_query\\_execution\(\)](#), [spark\\_advisory\\_shuffle\\_partition\\_size\(\)](#), [spark\\_auto\\_broadcast\\_join\\_threshold\(\)](#), [spark\\_coalesce\\_initial\\_num\\_partitions\(\)](#), [spark\\_coalesce\\_shuffle\\_partitions\(\)](#), [spark\\_session\\_config\(\)](#)

---

spark\_coalesce\_shuffle\_partitions

*Retrieves or sets whether coalescing contiguous shuffle partitions is enabled*

---

**Description**

Retrieves or sets whether coalescing contiguous shuffle partitions is enabled

**Usage**

```
spark_coalesce_shuffle_partitions(sc, enable = NULL)
```

**Arguments**

|        |  |
|--------|--|
| sc     | A spark_connection.  |
| enable | Whether to enable coalescing of contiguous shuffle partitions. Defaults to NULL to retrieve configuration entries. |

**See Also**

Other Spark runtime configuration: [spark\\_adaptive\\_query\\_execution\(\)](#), [spark\\_advisory\\_shuffle\\_partition\\_size\(\)](#), [spark\\_auto\\_broadcast\\_join\\_threshold\(\)](#), [spark\\_coalesce\\_initial\\_num\\_partitions\(\)](#), [spark\\_coalesce\\_min\\_num\\_partitions\(\)](#), [spark\\_session\\_config\(\)](#)

---

spark\_compilation\_spec

*Define a Spark Compilation Specification*

---

**Description**

For use with [compile\\_package\\_jars](#). The Spark compilation specification is used when compiling Spark extension Java Archives, and defines which versions of Spark, as well as which versions of Scala, should be used for compilation.

**Usage**

```
spark_compilation_spec(
  spark_version = NULL,
  spark_home = NULL,
  scalac_path = NULL,
  scala_filter = NULL,
  jar_name = NULL,
  jar_path = NULL,
  jar_dep = NULL,
  embedded_srcs = "embedded_sources.R"
)
```

**Arguments**

|               |   |
|---------------|---|
| spark_version | The Spark version to build against. This can be left unset if the path to a suitable Spark home is supplied.  |
| spark_home    | The path to a Spark home installation. This can be left unset if spark_version is supplied; in such a case, sparklyr will attempt to discover the associated Spark installation using <a href="#">spark_home_dir</a> .                    |
| scalac_path   | The path to the scalac compiler to be used during compilation of your Spark extension. Note that you should ensure the version of scalac selected matches the version of scalac used with the version of Spark you are compiling against. |

|               |  |
|---------------|--|
| scala_filter  | An optional R function that can be used to filter which scala files are used during compilation. This can be useful if you have auxiliary files that should only be included with certain versions of Spark. |
| jar_name      | The name to be assigned to the generated jar.  |
| jar_path      | The path to the jar tool to be used during compilation of your Spark extension.  |
| jar_dep       | An optional list of additional jar dependencies.   |
| embedded_srcs | Embedded source file(s) under <R package root>/java to be included in the root of the resulting jar file as resources  |

### Details

Most Spark extensions won't need to define their own compilation specification, and can instead rely on the default behavior of `compile_package_jars`.

---

|              |                                 |
|--------------|---------------------------------|
| spark_config | <i>Read Spark Configuration</i> |
|--------------|---------------------------------|

---

### Description

Read Spark Configuration

### Usage

```
spark_config(file = "config.yml", use_default = TRUE)
```

### Arguments

|             |  |
|-------------|--|
| file        | Name of the configuration file                             |
| use_default | TRUE to use the built-in defaults provided in this package |

### Details

Read Spark configuration using the [config](#) package.

### Value

Named list with configuration data

---

 spark\_config\_kubernetes

*Kubernetes Configuration*


---

**Description**

Convenience function to initialize a Kubernetes configuration instead of `spark_config()`, exposes common properties to set in Kubernetes clusters.

**Usage**

```
spark_config_kubernetes(
  master,
  version = "3.2.3",
  image = "spark:sparklyr",
  driver = random_string("sparklyr-"),
  account = "spark",
  jars = "local:///opt/sparklyr",
  forward = TRUE,
  executors = NULL,
  conf = NULL,
  timeout = 120,
  ports = c(8880, 8881, 4040),
  fix_config = identical(.Platform$OS.type, "windows"),
  ...
)
```

**Arguments**

|           |   |
|-----------|---|
| master    | Kubernetes url to connect to, found by running <code>kubectl cluster-info</code> .  |
| version   | The version of Spark being used.  |
| image     | Container image to use to launch Spark and sparklyr. Also known as <code>spark.kubernetes.container.image</code> .  |
| driver    | Name of the driver pod. If not set, the driver pod name is set to "sparklyr" suffixed by id to avoid name conflicts. Also known as <code>spark.kubernetes.driver.pod.name</code> .  |
| account   | Service account that is used when running the driver pod. The driver pod uses this service account when requesting executor pods from the API server. Also known as <code>spark.kubernetes.authenticate.driver.serviceAccountName</code> .  |
| jars      | Path to the sparklyr jars; either, a local path inside the container image with the sparklyr jars copied when the image was created or, a path accessible by the container where the sparklyr jars were copied. You can find a path to the sparklyr jars by running <code>system.file("java/", package = "sparklyr")</code> . |
| forward   | Should ports used in sparklyr be forwarded automatically through Kubernetes? Default to TRUE which runs <code>kubectl port-forward</code> and <code>kill kubectl</code> on disconnection.   |
| executors | Number of executors to request while connecting.  |

|            |   |
|------------|---|
| conf       | A named list of additional entries to add to sparklyr.shell.conf. |
| timeout    | Total seconds to wait before giving up on connection.             |
| ports      | Ports to forward using kubectl.                                   |
| fix_config | Should the spark-defaults.conf get fixed? TRUE for Windows.       |
| ...        | Additional parameters, currently not in use.                      |

---

spark\_config\_settings *Retrieve Available Settings*

---

### Description

Retrieves available sparklyr settings that can be used in configuration files or spark\_config().

### Usage

```
spark_config_settings()
```

---

spark\_connection *Retrieve the Spark Connection Associated with an R Object*

---

### Description

Retrieve the spark\_connection associated with an R object.

### Usage

```
spark_connection(x, ...)
```

### Arguments

|     |  |
|-----|--|
| x   | An R object from which a spark_connection can be obtained. |
| ... | Optional arguments; currently unused.                      |

---

spark\_connection-class  
*spark\_connection class*

---

### Description

spark\_connection class

---

spark\_connection\_find *Find Spark Connection*

---

**Description**

Finds an active spark connection in the environment given the connection parameters.

**Usage**

```
spark_connection_find(master = NULL, app_name = NULL, method = NULL)
```

**Arguments**

|          |                                      |
|----------|--------------------------------------|
| master   | The Spark master parameter.          |
| app_name | The Spark application name.          |
| method   | The method used to connect to Spark. |

---

spark\_connect\_method *Function that negotiates the connection with the Spark back-end*

---

**Description**

Function that negotiates the connection with the Spark back-end

**Usage**

```
spark_connect_method(  
  x,  
  method,  
  master,  
  spark_home,  
  config,  
  app_name,  
  version,  
  hadoop_version,  
  extensions,  
  scala_version,  
  ...  
)
```

**Arguments**

|                |   |
|----------------|---|
| x              | A dummy method object to determine which code to use to connect   |
| method         | The method used to connect to Spark. Default connection method is "shell" to connect using spark-submit, use "livy" to perform remote connections using HTTP, or "databricks" when using a Databricks clusters.   |
| master         | Spark cluster url to connect to. Use "local" to connect to a local instance of Spark installed via <a href="#">spark_install</a> .  |
| spark_home     | The path to a Spark installation. Defaults to the path provided by the SPARK_HOME environment variable. If SPARK_HOME is defined, it will always be used unless the version parameter is specified to force the use of a locally installed version.   |
| config         | Custom configuration for the generated Spark connection. See <a href="#">spark_config</a> for details.  |
| app_name       | The application name to be used while running in the Spark cluster.   |
| version        | The version of Spark to use. Required for "local" Spark connections, optional otherwise.  |
| hadoop_version | Version of Hadoop to use  |
| extensions     | Extension R packages to enable for this connection. By default, all packages enabled through the use of <a href="#">sparklyr::register_extension</a> will be passed here.   |
| scala_version  | Load the sparklyr jar file that is built with the version of Scala specified (this currently only makes sense for Spark 2.4, where sparklyr will by default assume Spark 2.4 on current host is built with Scala 2.11, and therefore 'scala_version = '2.12'' is needed if sparklyr is connecting to Spark 2.4 built with Scala 2.12) |
| ...            | Additional params to be passed to each 'spark_disconnect()' call (e.g., 'terminate = TRUE')   |

---

spark\_context\_config *Runtime configuration interface for the Spark Context.*

---

**Description**

Retrieves the runtime configuration interface for the Spark Context.

**Usage**

```
spark_context_config(sc)
```

**Arguments**

sc A spark\_connection.

---

|                 |                                   |
|-----------------|-----------------------------------|
| spark_dataframe | <i>Retrieve a Spark DataFrame</i> |
|-----------------|-----------------------------------|

---

**Description**

This S3 generic is used to access a Spark DataFrame object (as a Java object reference) from an R object.

**Usage**

```
spark_dataframe(x, ...)
```

**Arguments**

|     |   |
|-----|---|
| x   | An R object wrapping, or containing, a Spark DataFrame. |
| ... | Optional arguments; currently unused.                   |

**Value**

A [spark\\_jobj](#) representing a Java object reference to a Spark DataFrame.

---

|                                |   |
|--------------------------------|---|
| spark_default_compilation_spec | <i>Default Compilation Specification for Spark Extensions</i> |
|--------------------------------|---|

---

**Description**

This is the default compilation specification used for Spark extensions, when used with [compile\\_package\\_jars](#).

**Usage**

```
spark_default_compilation_spec(
  pkg = infer_active_package_name(),
  locations = NULL
)
```

**Arguments**

|           |  |
|-----------|--|
| pkg       | The package containing Spark extensions to be compiled.  |
| locations | Additional locations to scan. By default, the directories <code>/opt/scala</code> and <code>/usr/local/scala</code> will be scanned. |



---

|                  |                                  |
|------------------|----------------------------------|
| spark_dependency | <i>Define a Spark dependency</i> |
|------------------|----------------------------------|

---

**Description**

Define a Spark dependency consisting of a set of custom JARs, Spark packages, and customized dbplyr SQL translation env.

**Usage**

```
spark_dependency(
  jars = NULL,
  packages = NULL,
  initializer = NULL,
  catalog = NULL,
  repositories = NULL,
  dbplyr_sql_variant = NULL,
  ...
)
```

**Arguments**

|                    |   |
|--------------------|---|
| jars               | Character vector of full paths to JAR files.  |
| packages           | Character vector of Spark packages names.   |
| initializer        | Optional callback function called when initializing a connection.   |
| catalog            | Optional location where extension JAR files can be downloaded for Livy.   |
| repositories       | Character vector of Spark package repositories.   |
| dbplyr_sql_variant | Customization of dbplyr SQL translation env. Must be a named list of the following form: <code>list( scalar = list(scalar_fn1 = ..., scalar_fn2 = ..., &lt;etc&gt;), aggregate = list(agg_fn1 = ..., agg_fn2 = ..., &lt;etc&gt;), window = list(wnd_fn1 = ..., wnd_fn2 = ..., &lt;etc&gt; )</code> See <a href="#">sql_variant</a> for details. |
| ...                | Additional optional arguments.  |

**Value**

An object of type 'spark\_dependency'

---

`spark_dependency_fallback`*Fallback to Spark Dependency*

---

**Description**

Helper function to assist falling back to previous Spark versions.

**Usage**

```
spark_dependency_fallback(spark_version, supported_versions)
```

**Arguments**

`spark_version` The Spark version being requested in `spark_dependencies`.

`supported_versions`

The Spark versions that are supported by this extension.

**Value**

A Spark version to use.

---

`spark_extension`*Create Spark Extension*

---

**Description**

Creates an R package ready to be used as an Spark extension.

**Usage**

```
spark_extension(path)
```

**Arguments**

`path`

Location where the extension will be created.

---

|                |  |
|----------------|--|
| spark_home_set | <i>Set the SPARK_HOME environment variable</i> |
|----------------|--|

---

### Description

Set the SPARK\_HOME environment variable. This slightly speeds up some operations, including the connection time.

### Usage

```
spark_home_set(path = NULL, ...)
```

### Arguments

|      |   |
|------|---|
| path | A string containing the path to the installation location of Spark. If NULL, the path to the most latest Spark/Hadoop versions is used. |
| ...  | Additional parameters not currently used.   |

### Value

The function is mostly invoked for the side-effect of setting the SPARK\_HOME environment variable. It also returns TRUE if the environment was successfully set, and FALSE otherwise.

### Examples

```
## Not run:  
# Not run due to side-effects  
spark_home_set()  
  
## End(Not run)
```

---

|                           |   |
|---------------------------|---|
| spark_ide_connection_open | <i>Set of functions to provide integration with the RStudio IDE</i> |
|---------------------------|---|

---

### Description

Set of functions to provide integration with the RStudio IDE

**Usage**

```

spark_ide_connection_open(con, env, connect_call)

spark_ide_connection_closed(con)

spark_ide_connection_updated(con, hint)

spark_ide_connection_actions(con)

spark_ide_objects(con, catalog, schema, name, type)

spark_ide_columns(
  con,
  table = NULL,
  view = NULL,
  catalog = NULL,
  schema = NULL
)

spark_ide_preview(
  con,
  rowLimit,
  table = NULL,
  view = NULL,
  catalog = NULL,
  schema = NULL
)

```

**Arguments**

|              |   |
|--------------|---|
| con          | Valid Spark connection  |
| env          | R environment of the interactive R session                              |
| connect_call | R code that can be used to re-connect to the Spark connection           |
| hint         | Name of the Spark connection that the RStudio IDE can use as reference. |
| catalog      | Name of the top level of the requested table or view                    |
| schema       | Name of the second most top level of the requested level or view        |
| name         | The new of the view or table being requested                            |
| type         | Type of the object being requested, 'view' or 'table'                   |
| table        | Name of the requested table   |
| view         | Name of the requested view  |
| rowLimit     | The number of rows to show in the 'Preview' pane of the RStudio IDE     |

**Details**

These function are meant for downstream packages, that provide additional backends to 'sparklyr', to override the opening, closing, update, and preview functionality. The arguments are driven

by what the RStudio IDE API expects them to be, so this is the reason why some use ‘type’ to designate views or tables, and others have one argument for ‘table’, and another for ‘view’.

---

spark\_insert\_table      *Inserts a Spark DataFrame into a Spark table*

---

## Description

Inserts a Spark DataFrame into a Spark table

## Usage

```
spark_insert_table(
  x,
  name,
  mode = NULL,
  overwrite = FALSE,
  options = list(),
  ...
)
```

## Arguments

|           |  |
|-----------|--|
| x         | A Spark DataFrame or dplyr operation   |
| name      | The name to assign to the newly generated table.   |
| mode      | A character element. Specifies the behavior when data or table already exists. Supported values include: ‘error’, ‘append’, ‘overwrite’ and ignore. Notice that ‘overwrite’ will also change the column structure.<br>For more details see also <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes">https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes</a> for your version of Spark. |
| overwrite | Boolean; overwrite the table with the given name if it already exists?   |
| options   | A list of strings with additional options.   |
| ...       | Optional arguments; currently unused.  |

## See Also

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

spark\_install                      *Download and install various versions of Spark*

---

### Description

Install versions of Spark for use with local Spark connections (i.e. `spark_connect(master = "local")`)

### Usage

```
spark_install(
  version = NULL,
  hadoop_version = NULL,
  reset = TRUE,
  logging = "INFO",
  verbose = interactive()
)

spark_uninstall(version, hadoop_version)

spark_install_dir()

spark_install_tar(tarfile)

spark_installed_versions()

spark_available_versions(
  show_hadoop = FALSE,
  show_minor = FALSE,
  show_future = FALSE
)
```

### Arguments

|                |  |
|----------------|--|
| version        | Version of Spark to install. See <code>spark_available_versions</code> for a list of supported versions  |
| hadoop_version | Version of Hadoop to install. See <code>spark_available_versions</code> for a list of supported versions                                       |
| reset          | Attempts to reset settings to defaults.  |
| logging        | Logging level to configure install. Supported options: "WARN", "INFO"  |
| verbose        | Report information as Spark is downloaded / installed  |
| tarfile        | Path to TAR file conforming to the pattern <code>spark-###-bin-(hadoop)?###</code> where ### reference spark and hadoop versions respectively. |
| show_hadoop    | Show Hadoop distributions?   |
| show_minor     | Show minor Spark versions?   |
| show_future    | Should future versions which have not been released be shown?  |

**Value**

List with information about the installed version.

---

spark\_integ\_test\_skip *It lets the package know if it should test a particular functionality or not*

---

**Description**

It lets the package know if it should test a particular functionality or not

**Usage**

```
spark_integ_test_skip(sc, test_name)
```

**Arguments**

|           |                      |
|-----------|----------------------|
| sc        | Spark connection     |
| test_name | The name of the test |

**Details**

It expects a boolean to be returned. If TRUE, the corresponding test will be skipped. If FALSE the test will be conducted.

---

spark\_jobj *Retrieve a Spark JVM Object Reference*

---

**Description**

This S3 generic is used for accessing the underlying Java Virtual Machine (JVM) Spark objects associated with R objects. These objects act as references to Spark objects living in the JVM. Methods on these objects can be called with the [invoke](#) family of functions.

**Usage**

```
spark_jobj(x, ...)
```

**Arguments**

|     |  |
|-----|--|
| x   | An R object containing, or wrapping, a spark_jobj. |
| ... | Optional arguments; currently unused.              |

**See Also**

[invoke](#), for calling methods on Java object references.

---

|                  |                         |
|------------------|-------------------------|
| spark_jobj-class | <i>spark_jobj class</i> |
|------------------|-------------------------|

---

**Description**

spark\_jobj class

---

|                  |   |
|------------------|---|
| spark_last_error | <i>Surfaces the last error from Spark captured by internal 'spark_error' function</i> |
|------------------|---|

---

**Description**

Surfaces the last error from Spark captured by internal 'spark\_error' function

**Usage**

spark\_last\_error()

---

|                  |   |
|------------------|---|
| spark_load_table | <i>Reads from a Spark Table into a Spark DataFrame.</i> |
|------------------|---|

---

**Description**

Reads from a Spark Table into a Spark DataFrame.

**Usage**

```
spark_load_table(
  sc,
  name,
  path,
  options = list(),
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE
)
```



**Arguments**

|             |  |
|-------------|--|
| sc          | A spark_connection.  |
| name        | The name to assign to the newly generated table.   |
| path        | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.   |
| options     | A list of strings with additional options. See <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration</a> . |
| repartition | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.  |
| memory      | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)   |
| overwrite   | Boolean; overwrite the table with the given name if it already exists?   |

**See Also**

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

 spark\_log

*View Entries in the Spark Log*


---

**Description**

View the most recent entries in the Spark log. This can be useful when inspecting output / errors produced by Spark during the invocation of various commands.

**Usage**

```
spark_log(sc, n = 100, filter = NULL, ...)
```

**Arguments**

|        |   |
|--------|---|
| sc     | A spark_connection.   |
| n      | The max number of log entries to retrieve. Use NULL to retrieve all entries within the log. |
| filter | Character string to filter log entries.   |
| ...    | Optional arguments; currently unused.   |

---

|            |  |
|------------|--|
| spark_read | <i>Read file(s) into a Spark DataFrame using a custom reader</i> |
|------------|--|

---

### Description

Run a custom R function on Spark workers to ingest data from one or more files into a Spark DataFrame, assuming all files follow the same schema.

### Usage

```
spark_read(sc, paths, reader, columns, packages = TRUE, ...)
```

### Arguments

|          |   |
|----------|---|
| sc       | A spark_connection.   |
| paths    | A character vector of one or more file URIs (e.g., c("hdfs://localhost:9000/file.txt", "hdfs://localhost:9000/file2.txt"))  |
| reader   | A self-contained R function that takes a single file URI as argument and returns the data read from that file as a data frame.  |
| columns  | a named list of column names and column types of the resulting data frame (e.g., list(column_1 = "integer", column_2 = "character")), or a list of column names only if column types should be inferred from the data (e.g., list("column_1", "column_2")), or NULL if column types should be inferred and resulting data frame can have arbitrary column names |
| packages | A list of R packages to distribute to Spark workers   |
| ...      | Optional arguments; currently unused.   |

### See Also

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

### Examples

```
## Not run:

library(sparklyr)
sc <- spark_connect(
  master = "yarn",
  spark_home = "~/spark/spark-2.4.5-bin-hadoop2.7"
)
```

```
# This is a contrived example to show reader tasks will be distributed across
# all Spark worker nodes
spark_read(
  sc,
  rep("/dev/null", 10),
  reader = function(path) system("hostname", intern = TRUE),
  columns = c(hostname = "string")
) %>% sdf_collect()

## End(Not run)
```

---

spark\_read\_avro      *Read Apache Avro data into a Spark DataFrame.*

---

### Description

Notice this functionality requires the Spark connection `sc` to be instantiated with either an explicitly specified Spark version (i.e., `spark_connect(..., version = <version>, packages = c("avro", <other package(s)>), ...)`) or a specific version of Spark avro package to use (e.g., `spark_connect(..., packages = c("org.apache.spark:spark-avro_2.12:3.0.0", <other package(s)>), ...)`).

### Usage

```
spark_read_avro(
  sc,
  name = NULL,
  path = name,
  avro_schema = NULL,
  ignore_extension = TRUE,
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE
)
```

### Arguments

|                               |  |
|-------------------------------|--|
| <code>sc</code>               | A <code>spark_connection</code> .  |
| <code>name</code>             | The name to assign to the newly generated table.   |
| <code>path</code>             | The path to the file. Needs to be accessible from the cluster. Supports the <code>"hdfs://"</code> , <code>"s3a://"</code> and <code>"file://"</code> protocols. |
| <code>avro_schema</code>      | Optional Avro schema in JSON format  |
| <code>ignore_extension</code> | If enabled, all files with and without <code>.avro</code> extension are loaded (default: TRUE)   |
| <code>repartition</code>      | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.  |

|           |  |
|-----------|--|
| memory    | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?) |
| overwrite | Boolean; overwrite the table with the given name if it already exists?                         |

**See Also**

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

|                   |   |
|-------------------|---|
| spark_read_binary | <i>Read binary data into a Spark DataFrame.</i> |
|-------------------|---|

---

**Description**

Read binary files within a directory and convert each file into a record within the resulting Spark dataframe. The output will be a Spark dataframe with the following columns and possibly partition columns:

- path: StringType
- modificationTime: TimestampType
- length: LongType
- content: BinaryType

**Usage**

```
spark_read_binary(
  sc,
  name = NULL,
  dir = name,
  path_glob_filter = "*",
  recursive_file_lookup = FALSE,
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE
)
```

**Arguments**

|      |  |
|------|--|
| sc   | A spark_connection.                              |
| name | The name to assign to the newly generated table. |
| dir  | Directory to read binary files from.             |

|                       |  |
|-----------------------|--|
| path_glob_filter      | Glob pattern of binary files to be loaded (e.g., "*.jpg").   |
| recursive_file_lookup | If FALSE (default), then partition discovery will be enabled (i.e., if a partition naming scheme is present, then partitions specified by subdirectory names such as "date=2019-07-01" will be created and files outside subdirectories following a partition naming scheme will be ignored). If TRUE, then all nested directories will be searched even if their names do not follow a partition naming scheme. |
| repartition           | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.  |
| memory                | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)   |
| overwrite             | Boolean; overwrite the table with the given name if it already exists?   |

**See Also**

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

|                |   |
|----------------|---|
| spark_read_csv | <i>Read a CSV file into a Spark DataFrame</i> |
|----------------|---|

---

**Description**

Read a tabular data file into a Spark DataFrame.

**Usage**

```
spark_read_csv(
  sc,
  name = NULL,
  path = name,
  header = TRUE,
  columns = NULL,
  infer_schema = is.null(columns),
  delimiter = ",",
  quote = "\"",
  escape = "\\\"",
  charset = "UTF-8",
  null_value = NULL,
  options = list(),
  repartition = 0,
```

```

    memory = TRUE,
    overwrite = TRUE,
    ...
)

```

## Arguments

|              |   |
|--------------|---|
| sc           | A spark_connection.   |
| name         | The name to assign to the newly generated table.  |
| path         | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.  |
| header       | Boolean; should the first row of data be used as a header? Defaults to TRUE.  |
| columns      | A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType, "integer" for IntegerType, "integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType. |
| infer_schema | Boolean; should column types be automatically inferred? Requires one extra pass over the data. Defaults to is.null(columns).  |
| delimiter    | The character used to delimit each column. Defaults to ','.   |
| quote        | The character used as a quote. Defaults to '"'.   |
| escape       | The character used to escape other characters. Defaults to '\\'.  |
| charset      | The character set. Defaults to "UTF-8".   |
| null_value   | The character to use for null, or missing, values. Defaults to NULL.  |
| options      | A list of strings with additional options.  |
| repartition  | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.   |
| memory       | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)  |
| overwrite    | Boolean; overwrite the table with the given name if it already exists?  |
| ...          | Optional arguments; currently unused.   |

## Details

You can read data from HDFS (hdfs://), S3 (s3a://), as well as the local file system (file://).

When header is FALSE, the column names are generated with a V prefix; e.g. V1, V2, ....

## See Also

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

|                  |   |
|------------------|---|
| spark_read_delta | <i>Read from Delta Lake into a Spark DataFrame.</i> |
|------------------|---|

---

**Description**

Read from Delta Lake into a Spark DataFrame.

**Usage**

```
spark_read_delta(
  sc,
  path,
  name = NULL,
  version = NULL,
  timestamp = NULL,
  options = list(),
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  ...
)
```

**Arguments**

|             |  |
|-------------|--|
| sc          | A spark_connection.  |
| path        | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols. |
| name        | The name to assign to the newly generated table.   |
| version     | The version of the delta table to read.  |
| timestamp   | The timestamp of the delta table to read. For example, "2019-01-01" or "2019-01-01 T'00:00:00.000".                      |
| options     | A list of strings with additional options.   |
| repartition | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.              |
| memory      | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)                           |
| overwrite   | Boolean; overwrite the table with the given name if it already exists?   |
| ...         | Optional arguments; currently unused.  |

**See Also**

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

|                  |  |
|------------------|--|
| spark_read_image | <i>Read image data into a Spark DataFrame.</i> |
|------------------|--|

---

### Description

Read image files within a directory and convert each file into a record within the resulting Spark dataframe. The output will be a Spark dataframe consisting of struct types containing the following attributes:

- origin: StringType
- height: IntegerType
- width: IntegerType
- nChannels: IntegerType
- mode: IntegerType
- data: BinaryType

### Usage

```
spark_read_image(  
  sc,  
  name = NULL,  
  dir = name,  
  drop_invalid = TRUE,  
  repartition = 0,  
  memory = TRUE,  
  overwrite = TRUE  
)
```

### Arguments

|              |   |
|--------------|---|
| sc           | A spark_connection.   |
| name         | The name to assign to the newly generated table.  |
| dir          | Directory to read binary files from.  |
| drop_invalid | Whether to drop files that are not valid images from the result (default: TRUE).                            |
| repartition  | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning. |
| memory       | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)              |
| overwrite    | Boolean; overwrite the table with the given name if it already exists?                                      |



**See Also**

Other Spark serialization routines: `collect_from_rds()`, `spark_insert_table()`, `spark_load_table()`, `spark_read_avro()`, `spark_read_binary()`, `spark_read_csv()`, `spark_read_delta()`, `spark_read_jdbc()`, `spark_read_json()`, `spark_read_libsvm()`, `spark_read_orc()`, `spark_read_parquet()`, `spark_read_source()`, `spark_read_table()`, `spark_read_text()`, `spark_read()`, `spark_save_table()`, `spark_write_avro()`, `spark_write_csv()`, `spark_write_delta()`, `spark_write_jdbc()`, `spark_write_json()`, `spark_write_orc()`, `spark_write_parquet()`, `spark_write_source()`, `spark_write_table()`, `spark_write_text()`

---

|                 |  |
|-----------------|--|
| spark_read_jdbc | <i>Read from JDBC connection into a Spark DataFrame.</i> |
|-----------------|--|

---

**Description**

Read from JDBC connection into a Spark DataFrame.

**Usage**

```
spark_read_jdbc(
  sc,
  name,
  options = list(),
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  columns = NULL,
  ...
)
```

**Arguments**

|             |   |
|-------------|---|
| sc          | A spark_connection.   |
| name        | The name to assign to the newly generated table.  |
| options     | A list of strings with additional options. See <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration</a> .  |
| repartition | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.   |
| memory      | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)  |
| overwrite   | Boolean; overwrite the table with the given name if it already exists?  |
| columns     | A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType, "integer" for IntegerType, "integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType. |
| ...         | Optional arguments; currently unused.   |

**See Also**

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

**Examples**

```
## Not run:
sc <- spark_connect(
  master = "local",
  config = list(
    `sparklyr.shell.driver-class-path` = "/usr/share/java/mysql-connector-java-8.0.25.jar"
  )
)
spark_read_jdbc(
  sc,
  name = "my_sql_table",
  options = list(
    url = "jdbc:mysql://localhost:3306/my_sql_schema",
    driver = "com.mysql.jdbc.Driver",
    user = "me",
    password = "*****",
    dbtable = "my_sql_table"
  )
)
## End(Not run)
```

---

spark\_read\_json

*Read a JSON file into a Spark DataFrame*


---

**Description**

Read a table serialized in the **JavaScript Object Notation** format into a Spark DataFrame.

**Usage**

```
spark_read_json(
  sc,
  name = NULL,
  path = name,
  options = list(),
  repartition = 0,
  memory = TRUE,
```

```

    overwrite = TRUE,
    columns = NULL,
    ...
)

```

### Arguments

|             |   |
|-------------|---|
| sc          | A spark_connection.   |
| name        | The name to assign to the newly generated table.  |
| path        | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.  |
| options     | A list of strings with additional options.  |
| repartition | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.   |
| memory      | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)  |
| overwrite   | Boolean; overwrite the table with the given name if it already exists?  |
| columns     | A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType, "integer" for IntegerType, "integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType. |
| ...         | Optional arguments; currently unused.   |

### Details

You can read data from HDFS (hdfs://), S3 (s3a://), as well as the local file system (file://).

### See Also

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

spark\_read\_libsvm      *Read libsvm file into a Spark DataFrame.*

---

### Description

Read libsvm file into a Spark DataFrame.

**Usage**

```

spark_read_libsvm(
  sc,
  name = NULL,
  path = name,
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  options = list(),
  ...
)

```

**Arguments**

|             |  |
|-------------|--|
| sc          | A spark_connection.  |
| name        | The name to assign to the newly generated table.   |
| path        | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols. |
| repartition | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.              |
| memory      | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)                           |
| overwrite   | Boolean; overwrite the table with the given name if it already exists?   |
| options     | A list of strings with additional options.   |
| ...         | Optional arguments; currently unused.  |

**See Also**

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

 spark\_read\_orc

*Read a ORC file into a Spark DataFrame*


---

**Description**

Read a **ORC** file into a Spark DataFrame.

**Usage**

```

spark_read_orc(
  sc,
  name = NULL,
  path = name,
  options = list(),
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  columns = NULL,
  schema = NULL,
  ...
)

```

**Arguments**

|             |   |
|-------------|---|
| sc          | A spark_connection.   |
| name        | The name to assign to the newly generated table.  |
| path        | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.  |
| options     | A list of strings with additional options. See <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration</a> .  |
| repartition | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.   |
| memory      | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)  |
| overwrite   | Boolean; overwrite the table with the given name if it already exists?  |
| columns     | A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType, "integer" for IntegerType, "integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType. |
| schema      | A (java) read schema. Useful for optimizing read operation on nested data.  |
| ...         | Optional arguments; currently unused.   |

**Details**

You can read data from HDFS (hdfs://), S3 (s3a://), as well as the local file system (file://).

**See Also**

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

spark\_read\_parquet      *Read a Parquet file into a Spark DataFrame*

---

### Description

Read a **Parquet** file into a Spark DataFrame.

### Usage

```
spark_read_parquet(
  sc,
  name = NULL,
  path = name,
  options = list(),
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  columns = NULL,
  schema = NULL,
  ...
)
```

### Arguments

|             |   |
|-------------|---|
| sc          | A spark_connection.   |
| name        | The name to assign to the newly generated table.  |
| path        | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.  |
| options     | A list of strings with additional options. See <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration</a> .  |
| repartition | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.   |
| memory      | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)  |
| overwrite   | Boolean; overwrite the table with the given name if it already exists?  |
| columns     | A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType, "integer" for IntegerType, "integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType. |
| schema      | A (java) read schema. Useful for optimizing read operation on nested data.  |
| ...         | Optional arguments; currently unused.   |

**Details**

You can read data from HDFS (hdfs://), S3 (s3a://), as well as the local file system (file://).

**See Also**

Other Spark serialization routines: `collect_from_rds()`, `spark_insert_table()`, `spark_load_table()`, `spark_read_avro()`, `spark_read_binary()`, `spark_read_csv()`, `spark_read_delta()`, `spark_read_image()`, `spark_read_jdbc()`, `spark_read_json()`, `spark_read_libsvm()`, `spark_read_orc()`, `spark_read_source()`, `spark_read_table()`, `spark_read_text()`, `spark_read()`, `spark_save_table()`, `spark_write_avro()`, `spark_write_csv()`, `spark_write_delta()`, `spark_write_jdbc()`, `spark_write_json()`, `spark_write_orc()`, `spark_write_parquet()`, `spark_write_source()`, `spark_write_table()`, `spark_write_text()`

---

|                   |   |
|-------------------|---|
| spark_read_source | <i>Read from a generic source into a Spark DataFrame.</i> |
|-------------------|---|

---

**Description**

Read from a generic source into a Spark DataFrame.

**Usage**

```
spark_read_source(
  sc,
  name = NULL,
  path = name,
  source,
  options = list(),
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  columns = NULL,
  ...
)
```

**Arguments**

|             |  |
|-------------|--|
| sc          | A spark_connection.  |
| name        | The name to assign to the newly generated table.   |
| path        | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.   |
| source      | A data source capable of reading data.   |
| options     | A list of strings with additional options. See <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration</a> . |
| repartition | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.  |

|           |   |
|-----------|---|
| memory    | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)  |
| overwrite | Boolean; overwrite the table with the given name if it already exists?  |
| columns   | A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType, "integer" for IntegerType, "integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType. |
| ...       | Optional arguments; currently unused.   |

**See Also**

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

 spark\_read\_table

*Reads from a Spark Table into a Spark DataFrame.*


---

**Description**

Reads from a Spark Table into a Spark DataFrame.

**Usage**

```
spark_read_table(
  sc,
  name,
  options = list(),
  repartition = 0,
  memory = TRUE,
  columns = NULL,
  ...
)
```

**Arguments**

|             |  |
|-------------|--|
| sc          | A spark_connection.  |
| name        | The name to assign to the newly generated table.   |
| options     | A list of strings with additional options. See <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration</a> . |
| repartition | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.  |



|         |   |
|---------|---|
| memory  | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)  |
| columns | A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType, "integer" for IntegerType, "integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType. |
| ...     | Optional arguments; currently unused.   |

**See Also**

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

|                 |  |
|-----------------|--|
| spark_read_text | <i>Read a Text file into a Spark DataFrame</i> |
|-----------------|--|

---

**Description**

Read a Text file into a Spark DataFrame

**Usage**

```
spark_read_text(
  sc,
  name = NULL,
  path = name,
  repartition = 0,
  memory = TRUE,
  overwrite = TRUE,
  options = list(),
  whole = FALSE,
  ...
)
```

**Arguments**

|      |  |
|------|--|
| sc   | A spark_connection.  |
| name | The name to assign to the newly generated table.   |
| path | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols. |

|             |   |
|-------------|---|
| repartition | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning. |
| memory      | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)              |
| overwrite   | Boolean; overwrite the table with the given name if it already exists?                                      |
| options     | A list of strings with additional options.  |
| whole       | Read the entire text file as a single entry? Defaults to FALSE.   |
| ...         | Optional arguments; currently unused.   |

### Details

You can read data from HDFS (`hdfs://`), S3 (`s3a://`), as well as the local file system (`file://`).

### See Also

Other Spark serialization routines: `collect_from_rds()`, `spark_insert_table()`, `spark_load_table()`, `spark_read_avro()`, `spark_read_binary()`, `spark_read_csv()`, `spark_read_delta()`, `spark_read_image()`, `spark_read_jdbc()`, `spark_read_json()`, `spark_read_libsvm()`, `spark_read_orc()`, `spark_read_parquet()`, `spark_read_source()`, `spark_read_table()`, `spark_read()`, `spark_save_table()`, `spark_write_avro()`, `spark_write_csv()`, `spark_write_delta()`, `spark_write_jdbc()`, `spark_write_json()`, `spark_write_orc()`, `spark_write_parquet()`, `spark_write_source()`, `spark_write_table()`, `spark_write_text()`

---

|                  |   |
|------------------|---|
| spark_save_table | <i>Saves a Spark DataFrame as a Spark table</i> |
|------------------|---|

---

### Description

Saves a Spark DataFrame and as a Spark table.

### Usage

```
spark_save_table(x, path, mode = NULL, options = list())
```

### Arguments

|         |  |
|---------|--|
| x       | A Spark DataFrame or dplyr operation   |
| path    | The path to the file. Needs to be accessible from the cluster. Supports the <code>"hdfs://"</code> , <code>"s3a://"</code> and <code>"file://"</code> protocols.   |
| mode    | A character element. Specifies the behavior when data or table already exists. Supported values include: <code>'error'</code> , <code>'append'</code> , <code>'overwrite'</code> and <code>ignore</code> . Notice that <code>'overwrite'</code> will also change the column structure.<br>For more details see also <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes">https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes</a> for your version of Spark. |
| options | A list of strings with additional options.   |

**See Also**

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

spark\_session\_config *Runtime configuration interface for the Spark Session*

---

**Description**

Retrieves or sets runtime configuration entries for the Spark Session

**Usage**

```
spark_session_config(sc, config = TRUE, value = NULL)
```

**Arguments**

|        |   |
|--------|---|
| sc     | A spark_connection.   |
| config | The configuration entry name(s) (e.g., "spark.sql.shuffle.partitions"). Defaults to NULL to retrieve all configuration entries. |
| value  | The configuration value to be set. Defaults to NULL to retrieve configuration entries.  |

**See Also**

Other Spark runtime configuration: [spark\\_adaptive\\_query\\_execution\(\)](#), [spark\\_advisory\\_shuffle\\_partition\\_size\(\)](#), [spark\\_auto\\_broadcast\\_join\\_threshold\(\)](#), [spark\\_coalesce\\_initial\\_num\\_partitions\(\)](#), [spark\\_coalesce\\_min\\_num\\_partitions\(\)](#), [spark\\_coalesce\\_shuffle\\_partitions\(\)](#)

---

spark\_statistical\_routines  
*Generate random samples from some distribution*

---

**Description**

Generator methods for creating single-column Spark dataframes comprised of i.i.d. samples from some distribution.

**Arguments**

|                |  |
|----------------|--|
| sc             | A Spark connection.  |
| n              | Sample Size (default: 1000).   |
| num_partitions | Number of partitions in the resulting Spark dataframe (default: default parallelism of the Spark cluster). |
| seed           | Random seed (default: a random long integer).  |
| output_col     | Name of the output column containing sample values (default: "x").   |

---

|                  |  |
|------------------|--|
| spark_table_name | <i>Generate a Table Name from Expression</i> |
|------------------|--|

---

**Description**

Attempts to generate a table name from an expression; otherwise, assigns an auto-generated generic name with "sparklyr\_" prefix.

**Usage**

```
spark_table_name(expr)
```

**Arguments**

|      |  |
|------|--|
| expr | The expression to attempt to use as name |
|------|--|

---

|               |   |
|---------------|---|
| spark_version | <i>Get the Spark Version Associated with a Spark Connection</i> |
|---------------|---|

---

**Description**

Retrieve the version of Spark associated with a Spark connection.

**Usage**

```
spark_version(sc)
```

**Arguments**

|    |                     |
|----|---------------------|
| sc | A spark_connection. |
|----|---------------------|

**Details**

Suffixes for e.g. preview versions, or snapshotted versions, are trimmed – if you require the full Spark version, you can retrieve it with `invoke(spark_context(sc), "version")`.

**Value**

The Spark version as a [numeric\\_version](#).

---

`spark_version_from_home`*Get the Spark Version Associated with a Spark Installation*

---

**Description**

Retrieve the version of Spark associated with a Spark installation.

**Usage**

```
spark_version_from_home(spark_home, default = NULL)
```

**Arguments**

|                         |  |
|-------------------------|--|
| <code>spark_home</code> | The path to a Spark installation.  |
| <code>default</code>    | The default version to be inferred, in case version lookup failed, e.g. no Spark installation was found at <code>spark_home</code> . |

---

`spark_web`*Open the Spark web interface*

---

**Description**

Open the Spark web interface

**Usage**

```
spark_web(sc, ...)
```

**Arguments**

|                  |                                       |
|------------------|---------------------------------------|
| <code>sc</code>  | A <code>spark_connection</code> .     |
| <code>...</code> | Optional arguments; currently unused. |

---

 spark\_write

*Write Spark DataFrame to file using a custom writer*


---

### Description

Run a custom R function on Spark worker to write a Spark DataFrame into file(s). If Spark's speculative execution feature is enabled (i.e., 'spark.speculation' is true), then each write task may be executed more than once and the user-defined writer function will need to ensure no concurrent writes happen to the same file path (e.g., by appending UUID to each file name).

### Usage

```
spark_write(x, writer, paths, packages = NULL)
```

### Arguments

|          |  |
|----------|--|
| x        | A Spark Dataframe to be saved into file(s)   |
| writer   | A writer function with the signature function(partition, path) where partition is a R dataframe containing all rows from one partition of the original Spark Dataframe x and path is a string specifying the file to write partition to                            |
| paths    | A single destination path or a list of destination paths, each one specifying a location for a partition from x to be written to. If number of partition(s) in x is not equal to length(paths) then x will be re-partitioned to contain length(paths) partition(s) |
| packages | Boolean to distribute .libPaths() packages to each node, a list of packages to distribute, or a package bundle created with  |

### Examples

```
## Not run:

library(sparklyr)

sc <- spark_connect(master = "local[3]")

# copy some test data into a Spark Dataframe
sdf <- sdf_copy_to(sc, iris, overwrite = TRUE)

# create a writer function
writer <- function(df, path) {
  write.csv(df, path)
}

spark_write(
  sdf,
  writer,
  # re-partition sdf into 3 partitions and write them to 3 separate files
  paths = list("file:///tmp/file1", "file:///tmp/file2", "file:///tmp/file3"),
```

```

)

spark_write(
  sdf,
  writer,
  # save all rows into a single file
  paths = list("file:///tmp/all_rows")
)

## End(Not run)

```

---

spark\_write\_avro      *Serialize a Spark DataFrame into Apache Avro format*

---

### Description

Notice this functionality requires the Spark connection `sc` to be instantiated with either an explicitly specified Spark version (i.e., `spark_connect(..., version = <version>, packages = c("avro", <other package(s)>), ...)`) or a specific version of Spark avro package to use (e.g., `spark_connect(..., packages = c("org.apache.spark:spark-avro_2.12:3.0.0", <other package(s)>), ...)`).

### Usage

```

spark_write_avro(
  x,
  path,
  avro_schema = NULL,
  record_name = "topLevelRecord",
  record_namespace = "",
  compression = "snappy",
  partition_by = NULL
)

```

### Arguments

|                               |  |
|-------------------------------|--|
| <code>x</code>                | A Spark DataFrame or dplyr operation   |
| <code>path</code>             | The path to the file. Needs to be accessible from the cluster. Supports the <code>"hdfs://"</code> , <code>"s3a://"</code> and <code>"file://"</code> protocols. |
| <code>avro_schema</code>      | Optional Avro schema in JSON format  |
| <code>record_name</code>      | Optional top level record name in write result (default: "topLevelRecord")   |
| <code>record_namespace</code> | Record namespace in write result (default: "")   |
| <code>compression</code>      | Compression codec to use (default: "snappy")   |
| <code>partition_by</code>     | A character vector. Partitions the output by the given columns on the file system.   |

**See Also**

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

 spark\_write\_csv

 Write a Spark DataFrame to a CSV
 

---

**Description**

Write a Spark DataFrame to a tabular (typically, comma-separated) file.

**Usage**

```
spark_write_csv(
  x,
  path,
  header = TRUE,
  delimiter = ",",
  quote = "\"",
  escape = "\\ ",
  charset = "UTF-8",
  null_value = NULL,
  options = list(),
  mode = NULL,
  partition_by = NULL,
  ...
)
```

**Arguments**

|            |  |
|------------|--|
| x          | A Spark DataFrame or dplyr operation   |
| path       | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols. |
| header     | Should the first row of data be used as a header? Defaults to TRUE.  |
| delimiter  | The character used to delimit each column, defaults to ,.  |
| quote      | The character used as a quote. Defaults to '"'.  |
| escape     | The character used to escape other characters, defaults to \.  |
| charset    | The character set, defaults to "UTF-8".  |
| null_value | The character to use for default values, defaults to NULL.   |
| options    | A list of strings with additional options.   |



|              |  |
|--------------|--|
| mode         | A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure.<br>For more details see also <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes">https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes</a> for your version of Spark. |
| partition_by | A character vector. Partitions the output by the given columns on the file system.   |
| ...          | Optional arguments; currently unused.  |

**See Also**

Other Spark serialization routines: `collect_from_rds()`, `spark_insert_table()`, `spark_load_table()`, `spark_read_avro()`, `spark_read_binary()`, `spark_read_csv()`, `spark_read_delta()`, `spark_read_image()`, `spark_read_jdbc()`, `spark_read_json()`, `spark_read_libsvm()`, `spark_read_orc()`, `spark_read_parquet()`, `spark_read_source()`, `spark_read_table()`, `spark_read_text()`, `spark_read()`, `spark_save_table()`, `spark_write_avro()`, `spark_write_delta()`, `spark_write_jdbc()`, `spark_write_json()`, `spark_write_orc()`, `spark_write_parquet()`, `spark_write_source()`, `spark_write_table()`, `spark_write_text()`

---

spark\_write\_delta      *Writes a Spark DataFrame into Delta Lake*

---

**Description**

Writes a Spark DataFrame into Delta Lake.

**Usage**

```
spark_write_delta(
  x,
  path,
  mode = NULL,
  options = list(),
  partition_by = NULL,
  ...
)
```

**Arguments**

|      |  |
|------|--|
| x    | A Spark DataFrame or dplyr operation   |
| path | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.   |
| mode | A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure.<br>For more details see also <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes">https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes</a> for your version of Spark. |

|              |  |
|--------------|--|
| options      | A list of strings with additional options.   |
| partition_by | A character vector. Partitions the output by the given columns on the file system. |
| ...          | Optional arguments; currently unused.  |

**See Also**

Other Spark serialization routines: `collect_from_rds()`, `spark_insert_table()`, `spark_load_table()`, `spark_read_avro()`, `spark_read_binary()`, `spark_read_csv()`, `spark_read_delta()`, `spark_read_image()`, `spark_read_jdbc()`, `spark_read_json()`, `spark_read_libsvm()`, `spark_read_orc()`, `spark_read_parquet()`, `spark_read_source()`, `spark_read_table()`, `spark_read_text()`, `spark_read()`, `spark_save_table()`, `spark_write_avro()`, `spark_write_csv()`, `spark_write_jdbc()`, `spark_write_json()`, `spark_write_orc()`, `spark_write_parquet()`, `spark_write_source()`, `spark_write_table()`, `spark_write_text()`

---

|                  |   |
|------------------|---|
| spark_write_jdbc | <i>Writes a Spark DataFrame into a JDBC table</i> |
|------------------|---|

---

**Description**

Writes a Spark DataFrame into a JDBC table

**Usage**

```
spark_write_jdbc(
  x,
  name,
  mode = NULL,
  options = list(),
  partition_by = NULL,
  ...
)
```

**Arguments**

|              |  |
|--------------|--|
| x            | A Spark DataFrame or dplyr operation   |
| name         | The name to assign to the newly generated table.   |
| mode         | A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure.<br>For more details see also <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes">https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes</a> for your version of Spark. |
| options      | A list of strings with additional options.   |
| partition_by | A character vector. Partitions the output by the given columns on the file system.   |
| ...          | Optional arguments; currently unused.  |

**See Also**

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

**Examples**

```
## Not run:
sc <- spark_connect(
  master = "local",
  config = list(
    `sparklyr.shell.driver-class-path` = "/usr/share/java/mysql-connector-java-8.0.25.jar"
  )
)
spark_write_jdbc(
  sdf_len(sc, 10),
  name = "my_sql_table",
  options = list(
    url = "jdbc:mysql://localhost:3306/my_sql_schema",
    driver = "com.mysql.jdbc.Driver",
    user = "me",
    password = "*****",
    dbtable = "my_sql_table"
  )
)

## End(Not run)
```

---

 spark\_write\_json

*Write a Spark DataFrame to a JSON file*


---

**Description**

Serialize a Spark DataFrame to the [JavaScript Object Notation](#) format.

**Usage**

```
spark_write_json(
  x,
  path,
  mode = NULL,
  options = list(),
  partition_by = NULL,
  ...
)
```

**Arguments**

|              |  |
|--------------|--|
| x            | A Spark DataFrame or dplyr operation   |
| path         | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.   |
| mode         | A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure.<br>For more details see also <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes">https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes</a> for your version of Spark. |
| options      | A list of strings with additional options.   |
| partition_by | A character vector. Partitions the output by the given columns on the file system.   |
| ...          | Optional arguments; currently unused.  |

**See Also**

Other Spark serialization routines: `collect_from_rds()`, `spark_insert_table()`, `spark_load_table()`, `spark_read_avro()`, `spark_read_binary()`, `spark_read_csv()`, `spark_read_delta()`, `spark_read_image()`, `spark_read_jdbc()`, `spark_read_json()`, `spark_read_libsvm()`, `spark_read_orc()`, `spark_read_parquet()`, `spark_read_source()`, `spark_read_table()`, `spark_read_text()`, `spark_read()`, `spark_save_table()`, `spark_write_avro()`, `spark_write_csv()`, `spark_write_delta()`, `spark_write_jdbc()`, `spark_write_orc()`, `spark_write_parquet()`, `spark_write_source()`, `spark_write_table()`, `spark_write_text()`

---

 spark\_write\_orc

*Write a Spark DataFrame to a ORC file*


---

**Description**

Serialize a Spark DataFrame to the **ORC** format.

**Usage**

```
spark_write_orc(
  x,
  path,
  mode = NULL,
  options = list(),
  partition_by = NULL,
  ...
)
```

## Arguments

|              |  |
|--------------|--|
| x            | A Spark DataFrame or dplyr operation   |
| path         | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.   |
| mode         | A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure.<br>For more details see also <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes">https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes</a> for your version of Spark. |
| options      | A list of strings with additional options. See <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration</a> .   |
| partition_by | A character vector. Partitions the output by the given columns on the file system.   |
| ...          | Optional arguments; currently unused.  |

## See Also

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

spark\_write\_parquet     *Write a Spark DataFrame to a Parquet file*

---

## Description

Serialize a Spark DataFrame to the **Parquet** format.

## Usage

```
spark_write_parquet(  
  x,  
  path,  
  mode = NULL,  
  options = list(),  
  partition_by = NULL,  
  ...  
)
```

**Arguments**

|              |  |
|--------------|--|
| x            | A Spark DataFrame or dplyr operation   |
| path         | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.   |
| mode         | A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure.<br>For more details see also <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes">https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes</a> for your version of Spark. |
| options      | A list of strings with additional options. See <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration</a> .   |
| partition_by | A character vector. Partitions the output by the given columns on the file system.   |
| ...          | Optional arguments; currently unused.  |

**See Also**

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

|                 |   |
|-----------------|---|
| spark_write_rds | <i>Write Spark DataFrame to RDS files</i> |
|-----------------|---|

---

**Description**

Write Spark dataframe to RDS files. Each partition of the dataframe will be exported to a separate RDS file so that all partitions can be processed in parallel.

**Usage**

```
spark_write_rds(x, dest_uri)
```

**Arguments**

|          |   |
|----------|---|
| x        | A Spark DataFrame to be exported  |
| dest_uri | Can be a URI template containing 'partitionId' (e.g., "hdfs://my_data_part_{partitionId}.rds") where 'partitionId' will be substituted with ID of each partition using 'glue', or a list of URIs to be assigned to RDS output from all partitions (e.g., "hdfs://my_data_part_0.rds", "hdfs://my_data_part_1.rds", and so on) If working with a Spark instance running locally, then all URIs should be in "file://<local file path>" form. Otherwise the scheme of the URI should reflect the underlying file system the |

Spark instance is working with (e.g., "hdfs://"). If the resulting list of URI(s) does not contain unique values, then it will be post-processed with 'make.unique()' to ensure uniqueness.

### Value

A tibble containing partition ID and RDS file location for each partition of the input Spark dataframe.

---

|                    |   |
|--------------------|---|
| spark_write_source | <i>Writes a Spark DataFrame into a generic source</i> |
|--------------------|---|

---

### Description

Writes a Spark DataFrame into a generic source.

### Usage

```
spark_write_source(
  x,
  source,
  mode = NULL,
  options = list(),
  partition_by = NULL,
  ...
)
```

### Arguments

|              |   |
|--------------|---|
| x            | A Spark DataFrame or dplyr operation  |
| source       | A data source capable of reading data.  |
| mode         | A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure. For more details see also <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes">https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes</a> for your version of Spark. |
| options      | A list of strings with additional options.  |
| partition_by | A character vector. Partitions the output by the given columns on the file system.  |
| ...          | Optional arguments; currently unused.   |

### See Also

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_table\(\)](#), [spark\\_write\\_text\(\)](#)

---

|                   |  |
|-------------------|--|
| spark_write_table | <i>Writes a Spark DataFrame into a Spark table</i> |
|-------------------|--|

---

### Description

Writes a Spark DataFrame into a Spark table

### Usage

```
spark_write_table(
  x,
  name,
  mode = NULL,
  options = list(),
  partition_by = NULL,
  ...
)
```

### Arguments

|              |  |
|--------------|--|
| x            | A Spark DataFrame or dplyr operation   |
| name         | The name to assign to the newly generated table.   |
| mode         | A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure.<br>For more details see also <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes">https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes</a> for your version of Spark. |
| options      | A list of strings with additional options.   |
| partition_by | A character vector. Partitions the output by the given columns on the file system.   |
| ...          | Optional arguments; currently unused.  |

### See Also

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_text\(\)](#)



---

|                  |   |
|------------------|---|
| spark_write_text | <i>Write a Spark DataFrame to a Text file</i> |
|------------------|---|

---

## Description

Serialize a Spark DataFrame to the plain text format.

## Usage

```
spark_write_text(
  x,
  path,
  mode = NULL,
  options = list(),
  partition_by = NULL,
  ...
)
```

## Arguments

|              |  |
|--------------|--|
| x            | A Spark DataFrame or dplyr operation   |
| path         | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.   |
| mode         | A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure.<br>For more details see also <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes">https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes</a> for your version of Spark. |
| options      | A list of strings with additional options.   |
| partition_by | A character vector. Partitions the output by the given columns on the file system.   |
| ...          | Optional arguments; currently unused.  |

## See Also

Other Spark serialization routines: [collect\\_from\\_rds\(\)](#), [spark\\_insert\\_table\(\)](#), [spark\\_load\\_table\(\)](#), [spark\\_read\\_avro\(\)](#), [spark\\_read\\_binary\(\)](#), [spark\\_read\\_csv\(\)](#), [spark\\_read\\_delta\(\)](#), [spark\\_read\\_image\(\)](#), [spark\\_read\\_jdbc\(\)](#), [spark\\_read\\_json\(\)](#), [spark\\_read\\_libsvm\(\)](#), [spark\\_read\\_orc\(\)](#), [spark\\_read\\_parquet\(\)](#), [spark\\_read\\_source\(\)](#), [spark\\_read\\_table\(\)](#), [spark\\_read\\_text\(\)](#), [spark\\_read\(\)](#), [spark\\_save\\_table\(\)](#), [spark\\_write\\_avro\(\)](#), [spark\\_write\\_csv\(\)](#), [spark\\_write\\_delta\(\)](#), [spark\\_write\\_jdbc\(\)](#), [spark\\_write\\_json\(\)](#), [spark\\_write\\_orc\(\)](#), [spark\\_write\\_parquet\(\)](#), [spark\\_write\\_source\(\)](#), [spark\\_write\\_table\(\)](#)

---

|               |                           |
|---------------|---------------------------|
| src_databases | <i>Show database list</i> |
|---------------|---------------------------|

---

**Description**

Show database list

**Usage**

```
src_databases(sc, ...)
```

**Arguments**

|     |                                       |
|-----|---------------------------------------|
| sc  | A spark_connection.                   |
| ... | Optional arguments; currently unused. |

---

|             |                    |
|-------------|--------------------|
| stream_find | <i>Find Stream</i> |
|-------------|--------------------|

---

**Description**

Finds and returns a stream based on the stream's identifier.

**Usage**

```
stream_find(sc, id)
```

**Arguments**

|    |                                  |
|----|----------------------------------|
| sc | The associated Spark connection. |
| id | The stream identifier to find.   |

**Examples**

```
## Not run:
sc <- spark_connect(master = "local")
sdf_len(sc, 10) %>%
  spark_write_parquet(path = "parquet-in")

stream <- stream_read_parquet(sc, "parquet-in") %>%
  stream_write_parquet("parquet-out")

stream_id <- stream_id(stream)
stream_find(sc, stream_id)

## End(Not run)
```

---

stream\_generate\_test *Generate Test Stream*

---

### Description

Generates a local test stream, useful when testing streams locally.

### Usage

```
stream_generate_test(
  df = rep(1:1000),
  path = "source",
  distribution = floor(10 + 1e+05 * stats::dbinom(1:20, 20, 0.5)),
  iterations = 50,
  interval = 1
)
```

### Arguments

|              |   |
|--------------|---|
| df           | The data frame used as a source of rows to the stream, will be cast to data frame if needed. Defaults to a sequence of one thousand entries.        |
| path         | Path to save stream of files to, defaults to "source".  |
| distribution | The distribution of rows to use over each iteration, defaults to a binomial distribution. The stream will cycle through the distribution if needed. |
| iterations   | Number of iterations to execute before stopping, defaults to fifty.   |
| interval     | The interval in seconds use to write the stream, defaults to one second.  |

### Details

This function requires the callr package to be installed.

---

stream\_id *Spark Stream's Identifier*

---

### Description

Retrieves the identifier of the Spark stream.

### Usage

```
stream_id(stream)
```

### Arguments

|        |                          |
|--------|--------------------------|
| stream | The spark stream object. |
|--------|--------------------------|

stream\_lag

*Apply lag function to columns of a Spark Streaming DataFrame***Description**

Given a streaming Spark dataframe as input, this function will return another streaming dataframe that contains all columns in the input and column(s) that are shifted behind by the offset(s) specified in ‘...’ (see example)

**Usage**

```
stream_lag(x, cols, thresholds = NULL)
```

**Arguments**

|            |   |
|------------|---|
| x          | An object coercable to a Spark Streaming DataFrame.   |
| cols       | A list of expressions for a single or multiple variables to create that will contain the value of a previous entry.   |
| thresholds | Optional named list of timestamp column(s) and corresponding time duration(s) for determining whether a previous record is sufficiently recent relative to the current record. If the any of the time difference(s) between the current and a previous record is greater than the maximal duration allowed, then the previous record is discarded and will not be part of the query result. The durations can be specified with numeric types (which will be interpreted as max difference allowed in number of milliseconds between 2 UNIX timestamps) or time duration strings such as "5s", "5sec", "5min", "5hour", etc. Any timestamp column in ‘x’ that is not of timestamp or date Spark SQL types will be interpreted as number of milliseconds since the UNIX epoch. |

**Examples**

```
## Not run:

library(sparklyr)

sc <- spark_connect(master = "local", version = "2.2.0")

streaming_path <- tempfile("days_df_")
days_df <- dplyr::tibble(
  today = weekdays(as.Date(seq(7), origin = "1970-01-01"))
)
num_iters <- 7
stream_generate_test(
  df = days_df,
  path = streaming_path,
  distribution = rep(nrow(days_df), num_iters),
  iterations = num_iters
)
```

```

stream_read_csv(sc, streaming_path) %>%
  stream_lag(cols = c(yesterday = today ~ 1, two_days_ago = today ~ 2)) %>%
  collect() %>%
  print(n = 10L)

## End(Not run)

```

---

|             |                            |
|-------------|----------------------------|
| stream_name | <i>Spark Stream's Name</i> |
|-------------|----------------------------|

---

### Description

Retrieves the name of the Spark stream if available.

### Usage

```
stream_name(stream)
```

### Arguments

|        |                          |
|--------|--------------------------|
| stream | The spark stream object. |
|--------|--------------------------|

---

|                 |   |
|-----------------|---|
| stream_read_csv | <i>Read files created by the stream</i> |
|-----------------|---|

---

### Description

Read files created by the stream

### Usage

```

stream_read_csv(
  sc,
  path,
  name = NULL,
  header = TRUE,
  columns = NULL,
  delimiter = ",",
  quote = "\"",
  escape = "\\ ",
  charset = "UTF-8",
  null_value = NULL,
  options = list(),
  ...
)

```

```

)

stream_read_text(sc, path, name = NULL, options = list(), ...)

stream_read_json(sc, path, name = NULL, columns = NULL, options = list(), ...)

stream_read_parquet(
  sc,
  path,
  name = NULL,
  columns = NULL,
  options = list(),
  ...
)

stream_read_orc(sc, path, name = NULL, columns = NULL, options = list(), ...)

stream_read_kafka(sc, name = NULL, options = list(), ...)

stream_read_socket(sc, name = NULL, columns = NULL, options = list(), ...)

stream_read_delta(sc, path, name = NULL, options = list(), ...)

```

### Arguments

|            |   |
|------------|---|
| sc         | A spark_connection.   |
| path       | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.  |
| name       | The name to assign to the newly generated stream.   |
| header     | Boolean; should the first row of data be used as a header? Defaults to TRUE.  |
| columns    | A vector of column names or a named vector of column types. If specified, the elements can be "binary" for BinaryType, "boolean" for BooleanType, "byte" for ByteType, "integer" for IntegerType, "integer64" for LongType, "double" for DoubleType, "character" for StringType, "timestamp" for TimestampType and "date" for DateType. |
| delimiter  | The character used to delimit each column. Defaults to ','.   |
| quote      | The character used as a quote. Defaults to '"'.   |
| escape     | The character used to escape other characters. Defaults to '\\'.  |
| charset    | The character set. Defaults to "UTF-8".   |
| null_value | The character to use for null, or missing, values. Defaults to NULL.  |
| options    | A list of strings with additional options.  |
| ...        | Optional arguments; currently unused.   |

**Examples**

```
## Not run:

sc <- spark_connect(master = "local")

dir.create("csv-in")
write.csv(iris, "csv-in/data.csv", row.names = FALSE)

csv_path <- file.path("file://", getwd(), "csv-in")

stream <- stream_read_csv(sc, csv_path) %>% stream_write_csv("csv-out")

stream_stop(stream)

## End(Not run)
```

---

|               |                      |
|---------------|----------------------|
| stream_render | <i>Render Stream</i> |
|---------------|----------------------|

---

**Description**

Collects streaming statistics to render the stream as an 'htmlwidget'.

**Usage**

```
stream_render(stream = NULL, collect = 10, stats = NULL, ...)
```

**Arguments**

|         |  |
|---------|--|
| stream  | The stream to render   |
| collect | The interval in seconds to collect data before rendering the 'htmlwidget'.   |
| stats   | Optional stream statistics collected using <code>stream_stats()</code> , when specified, stream should be omitted. |
| ...     | Additional optional arguments.   |

**Examples**

```
## Not run:
library(sparklyr)
sc <- spark_connect(master = "local")

dir.create("iris-in")
write.csv(iris, "iris-in/iris.csv", row.names = FALSE)

stream <- stream_read_csv(sc, "iris-in/") %>%
  stream_write_csv("iris-out/")
```

```
stream_render(stream)
stream_stop(stream)

## End(Not run)
```

---

stream\_stats

*Stream Statistics*

---

## Description

Collects streaming statistics, usually, to be used with `stream_render()` to render streaming statistics.

## Usage

```
stream_stats(stream, stats = list())
```

## Arguments

|        |  |
|--------|--|
| stream | The stream to collect statistics from.                                 |
| stats  | An optional stats object generated using <code>stream_stats()</code> . |

## Value

A stats object containing streaming statistics that can be passed back to the `stats` parameter to continue aggregating streaming stats.

## Examples

```
## Not run:
sc <- spark_connect(master = "local")
sdf_len(sc, 10) %>%
  spark_write_parquet(path = "parquet-in")

stream <- stream_read_parquet(sc, "parquet-in") %>%
  stream_write_parquet("parquet-out")

stream_stats(stream)

## End(Not run)
```



---

|             |                             |
|-------------|-----------------------------|
| stream_stop | <i>Stops a Spark Stream</i> |
|-------------|-----------------------------|

---

**Description**

Stops processing data from a Spark stream.

**Usage**

```
stream_stop(stream)
```

**Arguments**

|        |  |
|--------|--|
| stream | The spark stream object to be stopped. |
|--------|--|

---

|                           |  |
|---------------------------|--|
| stream_trigger_continuous | <i>Spark Stream Continuous Trigger</i> |
|---------------------------|--|

---

**Description**

Creates a Spark structured streaming trigger to execute continuously. This mode is the most performant but not all operations are supported.

**Usage**

```
stream_trigger_continuous(checkpoint = 5000)
```

**Arguments**

|            |  |
|------------|--|
| checkpoint | The checkpoint interval specified in milliseconds. |
|------------|--|

**See Also**

[stream\\_trigger\\_interval](#)

---

stream\_trigger\_interval  
*Spark Stream Interval Trigger*

---

**Description**

Creates a Spark structured streaming trigger to execute over the specified interval.

**Usage**

```
stream_trigger_interval(interval = 1000)
```

**Arguments**

interval            The execution interval specified in milliseconds.

**See Also**

[stream\\_trigger\\_continuous](#)

---

stream\_view            *View Stream*

---

**Description**

Opens a Shiny gadget to visualize the given stream.

**Usage**

```
stream_view(stream, ...)
```

**Arguments**

stream            The stream to visualize.  
...                Additional optional arguments.

**Examples**

```
## Not run:
library(sparklyr)
sc <- spark_connect(master = "local")

dir.create("iris-in")
write.csv(iris, "iris-in/iris.csv", row.names = FALSE)

stream_read_csv(sc, "iris-in/") %>%
```

```

stream_write_csv("iris-out/") %>%
stream_view() %>%
stream_stop()

## End(Not run)

```

---

|                  |                         |
|------------------|-------------------------|
| stream_watermark | <i>Watermark Stream</i> |
|------------------|-------------------------|

---

### Description

Ensures a stream has a watermark defined, which is required for some operations over streams.

### Usage

```
stream_watermark(x, column = "timestamp", threshold = "10 minutes")
```

### Arguments

|           |   |
|-----------|---|
| x         | An object coercable to a Spark Streaming DataFrame.   |
| column    | The name of the column that contains the event time of the row, if the column is missing, a column with the current time will be added. |
| threshold | The minimum delay to wait to data to arrive late, defaults to ten minutes.  |

---

|                  |                                  |
|------------------|----------------------------------|
| stream_write_csv | <i>Write files to the stream</i> |
|------------------|----------------------------------|

---

### Description

Write files to the stream

### Usage

```

stream_write_csv(
  x,
  path,
  mode = c("append", "complete", "update"),
  trigger = stream_trigger_interval(),
  checkpoint = file.path(path, "checkpoint"),
  header = TRUE,
  delimiter = ",",
  quote = "\"",
  escape = "\\ ",
  charset = "UTF-8",
  null_value = NULL,

```

```
    options = list(),
    partition_by = NULL,
    ...
)

stream_write_text(
  x,
  path,
  mode = c("append", "complete", "update"),
  trigger = stream_trigger_interval(),
  checkpoint = file.path(path, "checkpoints", random_string("")),
  options = list(),
  partition_by = NULL,
  ...
)

stream_write_json(
  x,
  path,
  mode = c("append", "complete", "update"),
  trigger = stream_trigger_interval(),
  checkpoint = file.path(path, "checkpoints", random_string("")),
  options = list(),
  partition_by = NULL,
  ...
)

stream_write_parquet(
  x,
  path,
  mode = c("append", "complete", "update"),
  trigger = stream_trigger_interval(),
  checkpoint = file.path(path, "checkpoints", random_string("")),
  options = list(),
  partition_by = NULL,
  ...
)

stream_write_orc(
  x,
  path,
  mode = c("append", "complete", "update"),
  trigger = stream_trigger_interval(),
  checkpoint = file.path(path, "checkpoints", random_string("")),
  options = list(),
  partition_by = NULL,
  ...
)
```

```

stream_write_kafka(
  x,
  mode = c("append", "complete", "update"),
  trigger = stream_trigger_interval(),
  checkpoint = file.path("checkpoints", random_string("")),
  options = list(),
  partition_by = NULL,
  ...
)

stream_write_console(
  x,
  mode = c("append", "complete", "update"),
  options = list(),
  trigger = stream_trigger_interval(),
  partition_by = NULL,
  ...
)

stream_write_delta(
  x,
  path,
  mode = c("append", "complete", "update"),
  checkpoint = file.path("checkpoints", random_string("")),
  options = list(),
  partition_by = NULL,
  ...
)

```

### Arguments

|            |  |
|------------|--|
| x          | A Spark DataFrame or dplyr operation   |
| path       | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.   |
| mode       | Specifies how data is written to a streaming sink. Valid values are "append", "complete" or "update".  |
| trigger    | The trigger for the stream query, defaults to micro-batches running every 5 seconds. See <a href="#">stream_trigger_interval</a> and <a href="#">stream_trigger_continuous</a> . |
| checkpoint | The location where the system will write all the checkpoint information to guarantee end-to-end fault-tolerance.   |
| header     | Should the first row of data be used as a header? Defaults to TRUE.  |
| delimiter  | The character used to delimit each column, defaults to ,.  |
| quote      | The character used as a quote. Defaults to '' ''.  |
| escape     | The character used to escape other characters, defaults to \.  |
| charset    | The character set, defaults to "UTF-8".  |

|              |  |
|--------------|--|
| null_value   | The character to use for default values, defaults to NULL. |
| options      | A list of strings with additional options.                 |
| partition_by | Partitions the output by the given list of columns.        |
| ...          | Optional arguments; currently unused.                      |

**See Also**

Other Spark stream serialization: [stream\\_write\\_memory\(\)](#)

**Examples**

```
## Not run:

sc <- spark_connect(master = "local")

dir.create("csv-in")
write.csv(iris, "csv-in/data.csv", row.names = FALSE)

csv_path <- file.path("file://", getwd(), "csv-in")

stream <- stream_read_csv(sc, csv_path) %>% stream_write_csv("csv-out")

stream_stop(stream)

## End(Not run)
```

---

stream\_write\_memory     *Write Memory Stream*

---

**Description**

Writes a Spark dataframe stream into a memory stream.

**Usage**

```
stream_write_memory(
  x,
  name = random_string("sparklyr_tmp_"),
  mode = c("append", "complete", "update"),
  trigger = stream_trigger_interval(),
  checkpoint = file.path("checkpoints", name, random_string("")),
  options = list(),
  partition_by = NULL,
  ...
)
```

**Arguments**

|              |  |
|--------------|--|
| x            | A Spark DataFrame or dplyr operation   |
| name         | The name to assign to the newly generated stream.  |
| mode         | Specifies how data is written to a streaming sink. Valid values are "append", "complete" or "update".  |
| trigger      | The trigger for the stream query, defaults to micro-batches running every 5 seconds. See <a href="#">stream_trigger_interval</a> and <a href="#">stream_trigger_continuous</a> . |
| checkpoint   | The location where the system will write all the checkpoint information to guarantee end-to-end fault-tolerance.   |
| options      | A list of strings with additional options.   |
| partition_by | Partitions the output by the given list of columns.  |
| ...          | Optional arguments; currently unused.  |

**See Also**

Other Spark stream serialization: [stream\\_write\\_csv\(\)](#)

---

tbl\_cache

*Cache a Spark Table*


---

**Description**

Force a Spark table with name name to be loaded into memory. Operations on cached tables should normally (although not always) be more performant than the same operation performed on an uncached table.

**Usage**

```
tbl_cache(sc, name, force = TRUE)
```

**Arguments**

|       |   |
|-------|---|
| sc    | A spark_connection.   |
| name  | The table name.   |
| force | Force the data to be loaded into memory? This is accomplished by calling the count API on the associated Spark DataFrame. |

---

|               |                              |
|---------------|------------------------------|
| tbl_change_db | <i>Use specific database</i> |
|---------------|------------------------------|

---

**Description**

Use specific database

**Usage**

```
tbl_change_db(sc, name)
```

**Arguments**

|      |                     |
|------|---------------------|
| sc   | A spark_connection. |
| name | The database name.  |

---

|             |                              |
|-------------|------------------------------|
| tbl_uncache | <i>Uncache a Spark Table</i> |
|-------------|------------------------------|

---

**Description**

Force a Spark table with name name to be unloaded from memory.

**Usage**

```
tbl_uncache(sc, name)
```

**Arguments**

|      |                     |
|------|---------------------|
| sc   | A spark_connection. |
| name | The table name.     |



---

|               |   |
|---------------|---|
| transform_sdf | <i>transform a subset of column(s) in a Spark Dataframe</i> |
|---------------|---|

---

**Description**

transform a subset of column(s) in a Spark Dataframe

**Usage**

```
transform_sdf(x, cols, fn)
```

**Arguments**

|      |  |
|------|--|
| x    | An object coercible to a Spark DataFrame   |
| cols | Subset of columns to apply transformation to   |
| fn   | Transformation function taking column name as the 1st parameter, the corresponding <code>org.apache.spark.sql.Column</code> object as the 2nd parameter, and returning a transformed <code>org.apache.spark.sql.Column</code> object |

---

|       |              |
|-------|--------------|
| unite | <i>Unite</i> |
|-------|--------------|

---

**Description**

See [unite](#) for more details.

---

|        |               |
|--------|---------------|
| unnest | <i>Unnest</i> |
|--------|---------------|

---

**Description**

See [unnest](#) for more details.

---

|              |  |
|--------------|--|
| [. tbl_spark | <i>Subsetting operator for Spark dataframe</i> |
|--------------|--|

---

**Description**

Subsetting operator for Spark dataframe allowing a subset of column(s) to be selected using syntaxes similar to those supported by R dataframes

**Usage**

```
## S3 method for class 'tbl_spark'
x[i]
```

**Arguments**

|   |  |
|---|--|
| x | The Spark dataframe  |
| i | Expression specifying subset of column(s) to include or exclude from the result (e.g., <code>["col1"]</code> , <code>[c("col1", "col2")]</code> , <code>[1:10]</code> , <code>[-1]</code> , <code>[NULL]</code> , or <code>[]</code> ) |

---

|      |   |
|------|---|
| %->% | <i>Infix operator for composing a lambda expression</i> |
|------|---|

---

**Description**

Infix operator that allows a lambda expression to be composed in R and be translated to Spark SQL equivalent using `dbplyr::translate_sql` functionalities

**Usage**

```
params %->% ...
```

**Arguments**

|        |   |
|--------|---|
| params | Parameter(s) of the lambda expression, can be either a single parameter or a comma separated listed of parameters in the form of <code>.(param1, param2, ...)</code> (see examples) |
| ...    | Body of the lambda expression, <i>*must be within parentheses*</i>  |

**Details**

Notice when composing a lambda expression in R, the body of the lambda expression *\*must always be surrounded with parentheses\**, otherwise a parsing error will occur.

### Examples

## Not run:

a %->% (mean(a) + 1) # translates to <SQL> `a` -> (AVG(`a`)) OVER () + 1.0)

.(a, b) %->% (a < 1 && b > 1) # translates to <SQL> `a`,`b` -> (`a` < 1.0 AND `b` > 1.0)

## End(Not run)

# Index

- \* **Spark data frames**
  - sdf\_copy\_to, [166](#)
  - sdf\_distinct, [169](#)
  - sdf\_random\_split, [177](#)
  - sdf\_register, [182](#)
  - sdf\_sample, [192](#)
  - sdf\_sort, [194](#)
  - sdf\_weighted\_sample, [198](#)
- \* **Spark runtime configuration**
  - spark\_adaptive\_query\_execution, [203](#)
  - spark\_advisory\_shuffle\_partition\_size, [204](#)
  - spark\_auto\_broadcast\_join\_threshold, [208](#)
  - spark\_coalesce\_initial\_num\_partitions, [208](#)
  - spark\_coalesce\_min\_num\_partitions, [209](#)
  - spark\_coalesce\_shuffle\_partitions, [209](#)
  - spark\_session\_config, [243](#)
- \* **Spark serialization routines**
  - collect\_from\_rds, [11](#)
  - spark\_insert\_table, [221](#)
  - spark\_load\_table, [224](#)
  - spark\_read, [226](#)
  - spark\_read\_avro, [227](#)
  - spark\_read\_binary, [228](#)
  - spark\_read\_csv, [229](#)
  - spark\_read\_delta, [231](#)
  - spark\_read\_image, [232](#)
  - spark\_read\_jdbc, [233](#)
  - spark\_read\_json, [234](#)
  - spark\_read\_libsvm, [235](#)
  - spark\_read\_orc, [236](#)
  - spark\_read\_parquet, [238](#)
  - spark\_read\_source, [239](#)
  - spark\_read\_table, [240](#)
  - spark\_read\_text, [241](#)
  - spark\_save\_table, [242](#)
  - spark\_write\_avro, [247](#)
  - spark\_write\_csv, [248](#)
  - spark\_write\_delta, [249](#)
  - spark\_write\_jdbc, [250](#)
  - spark\_write\_json, [251](#)
  - spark\_write\_orc, [252](#)
  - spark\_write\_parquet, [253](#)
  - spark\_write\_source, [255](#)
  - spark\_write\_table, [256](#)
  - spark\_write\_text, [257](#)
- \* **Spark statistical routines**
  - sdf\_rbeta, [178](#)
  - sdf\_rbinom, [179](#)
  - sdf\_rcauchy, [180](#)
  - sdf\_rchisq, [181](#)
  - sdf\_rexp, [183](#)
  - sdf\_rgamma, [184](#)
  - sdf\_rgeom, [185](#)
  - sdf\_rhyper, [186](#)
  - sdf\_rlnorm, [187](#)
  - sdf\_rnorm, [188](#)
  - sdf\_rpois, [189](#)
  - sdf\_rt, [189](#)
  - sdf\_runif, [190](#)
  - sdf\_rweibull, [191](#)
- \* **Spark stream serialization**
  - stream\_write\_csv, [267](#)
  - stream\_write\_memory, [270](#)
- \* **feature transformers**
  - ft\_binarizer, [16](#)
  - ft\_bucketizer, [17](#)
  - ft\_chisq\_selector, [19](#)
  - ft\_count\_vectorizer, [20](#)
  - ft\_dct, [22](#)
  - ft\_elementwise\_product, [23](#)
  - ft\_feature\_hasher, [24](#)
  - ft\_hashing\_tf, [26](#)

- ft\_idf, 27
- ft\_imputer, 28
- ft\_index\_to\_string, 29
- ft\_interaction, 30
- ft\_lsh, 31
- ft\_max\_abs\_scaler, 34
- ft\_min\_max\_scaler, 35
- ft\_ngram, 37
- ft\_normalizer, 38
- ft\_one\_hot\_encoder, 39
- ft\_one\_hot\_encoder\_estimator, 40
- ft\_pca, 41
- ft\_polynomial\_expansion, 43
- ft\_quantile\_discretizer, 44
- ft\_r\_formula, 48
- ft\_regex\_tokenizer, 46
- ft\_robust\_scaler, 47
- ft\_sql\_transformer, 50
- ft\_standard\_scaler, 51
- ft\_stop\_words\_remover, 53
- ft\_string\_indexer, 54
- ft\_tokenizer, 56
- ft\_vector\_assembler, 57
- ft\_vector\_indexer, 58
- ft\_vector\_slicer, 59
- ft\_word2vec, 60
- \* ml algorithms**
  - ml\_aft\_survival\_regression, 85
  - ml\_decision\_tree\_classifier, 96
  - ml\_gbt\_classifier, 107
  - ml\_generalized\_linear\_regression, 112
  - ml\_isotonic\_regression, 116
  - ml\_linear\_regression, 125
  - ml\_linear\_svc, 127
  - ml\_logistic\_regression, 129
  - ml\_multilayer\_perceptron\_classifier, 137
  - ml\_naive\_bayes, 140
  - ml\_one\_vs\_rest, 143
  - ml\_random\_forest\_classifier, 149
- ?join\_by, 77
- [.tbl\_spark, 274
- %->%, 274
- augment.\_ml\_model\_decision\_tree\_classification (ml\_tree\_tidiers), 155
- augment.\_ml\_model\_decision\_tree\_regression (ml\_tree\_tidiers), 155
- augment.\_ml\_model\_gbt\_classification (ml\_tree\_tidiers), 155
- augment.\_ml\_model\_gbt\_regression (ml\_tree\_tidiers), 155
- augment.\_ml\_model\_linear\_regression (ml\_glm\_tidiers), 114
- augment.\_ml\_model\_logistic\_regression (ml\_logistic\_regression\_tidiers), 132
- augment.\_ml\_model\_random\_forest\_classification (ml\_tree\_tidiers), 155
- augment.\_ml\_model\_random\_forest\_regression (ml\_tree\_tidiers), 155
- augment.ml\_model\_aft\_survival\_regression (ml\_survival\_regression\_tidiers), 154
- augment.ml\_model\_als (ml\_als\_tidiers), 91
- augment.ml\_model\_bisecting\_kmeans (ml\_unsupervised\_tidiers), 157
- augment.ml\_model\_decision\_tree\_classification (ml\_tree\_tidiers), 155
- augment.ml\_model\_decision\_tree\_regression (ml\_tree\_tidiers), 155
- augment.ml\_model\_gaussian\_mixture (ml\_unsupervised\_tidiers), 157
- augment.ml\_model\_gbt\_classification (ml\_tree\_tidiers), 155
- augment.ml\_model\_gbt\_regression (ml\_tree\_tidiers), 155
- augment.ml\_model\_generalized\_linear\_regression (ml\_glm\_tidiers), 114
- augment.ml\_model\_isotonic\_regression (ml\_isotonic\_regression\_tidiers), 117
- augment.ml\_model\_kmeans (ml\_unsupervised\_tidiers), 157
- augment.ml\_model\_lda (ml\_lda\_tidiers), 124
- augment.ml\_model\_linear\_regression (ml\_glm\_tidiers), 114
- augment.ml\_model\_linear\_svc (ml\_linear\_svc\_tidiers), 129
- augment.ml\_model\_logistic\_regression (ml\_logistic\_regression\_tidiers), 132
- augment.ml\_model\_multilayer\_perceptron\_classification (ml\_multilayer\_perceptron\_tidiers),

- 140
- augment.ml\_model\_naive\_bayes  
(ml\_naive\_bayes\_tidiers), 142
- augment.ml\_model\_pca(ml\_pca\_tidiers),  
144
- augment.ml\_model\_random\_forest\_classification  
(ml\_tree\_tidiers), 155
- augment.ml\_model\_random\_forest\_regression  
(ml\_tree\_tidiers), 155
  
- checkpoint\_directory, 10
- collect\_from\_rds, 11, 221, 225, 226,  
228–231, 233–237, 239–243,  
248–257
- compile\_package\_jars, 11, 210, 216
- config, 211
- connection\_config, 12
- copy\_to.spark\_connection, 12
- cross\_join(), 77
- cut, 17
  
- distinct, 13, 13
- download\_scalac, 14
- dplyr\_hof, 14
  
- ensure, 14
  
- fill, 15, 15
- filter, 15, 15
- find\_scalac, 15
- ft\_binarizer, 16, 18, 20, 22–24, 26–31, 33,  
35, 36, 38–42, 44, 45, 47, 48, 50–52,  
54–57, 59–61
- ft\_bucketed\_random\_projection\_lsh  
(ft\_lsh), 31
- ft\_bucketizer, 16, 17, 20, 22–24, 26–31, 33,  
35, 36, 38–42, 44, 45, 47, 48, 50–52,  
54–57, 59–61
- ft\_chisq\_selector, 16, 18, 19, 22–24,  
26–31, 33, 35, 36, 38–42, 44, 45, 47,  
48, 50–52, 54–57, 59–61
- ft\_count\_vectorizer, 16, 18, 20, 20, 23, 24,  
26–31, 33, 35, 36, 38–42, 44, 45, 47,  
48, 50–52, 54–57, 59–61, 122
- ft\_dct, 16, 18, 20, 22, 22, 24, 26–31, 33, 35,  
36, 38–42, 44, 45, 47, 48, 50–52,  
54–57, 59–61
- ft\_discrete\_cosine\_transform(ft\_dct),  
22
- ft\_dplyr\_transformer  
(ft\_sql\_transformer), 50
- ft\_elementwise\_product, 16, 18, 20, 22, 23,  
23, 26–31, 33, 35, 36, 38–42, 44, 45,  
47, 48, 50–52, 54–57, 59–61
- ft\_feature\_hasher, 16, 18, 20, 22–24, 24,  
27–31, 33, 35, 36, 38–42, 44, 45, 47,  
48, 50–52, 54–57, 59–61
- ft\_hashing\_tf, 16, 18, 20, 22–24, 26, 26,  
28–31, 33, 35, 36, 38–42, 44, 45, 47,  
48, 50–52, 54–57, 59–61
- ft\_idf, 16, 18, 20, 22–24, 26, 27, 27, 29–31,  
33, 35, 36, 38–42, 44, 45, 47, 48,  
50–52, 54–57, 59–61
- ft\_imputer, 16, 18, 20, 22–24, 26–28, 28, 30,  
31, 33, 35, 36, 38–42, 44, 46–48,  
50–52, 54–57, 59–61
- ft\_index\_to\_string, 16, 18, 20, 22–24,  
26–29, 29, 31, 33, 35, 36, 38–42, 44,  
46–48, 50–52, 54–57, 59–61
- ft\_interaction, 16, 18, 20, 22–24, 26–30,  
30, 33, 35, 36, 38–42, 44, 46–48,  
50–52, 54–57, 59–61
- ft\_lsh, 16, 18, 20, 22–24, 26–31, 31, 35, 36,  
38–42, 44, 46–48, 50–52, 54–57,  
59–61
- ft\_lsh\_utils, 33
- ft\_max\_abs\_scaler, 16, 18, 20, 22–24,  
26–31, 33, 34, 36, 38–42, 44, 46–48,  
50–52, 54–57, 59–61
- ft\_min\_max\_scaler, 16, 18, 20, 22–24,  
26–31, 33, 35, 35, 38–42, 44, 46–48,  
50–52, 54–57, 59–61
- ft\_minhash\_lsh(ft\_lsh), 31
- ft\_ngram, 16, 18, 20, 22–24, 26–31, 33, 35,  
36, 37, 39–42, 44, 46–48, 50–52,  
54–57, 59–61
- ft\_normalizer, 16, 18, 20, 22–24, 26–31, 33,  
35, 36, 38, 38, 40–42, 44, 46–48,  
50–52, 54–57, 59–61
- ft\_one\_hot\_encoder, 16, 18, 20, 22–24,  
26–31, 33, 35, 36, 38, 39, 39, 41, 42,  
44, 46–48, 50–52, 54–57, 59–61
- ft\_one\_hot\_encoder\_estimator, 16, 18, 20,  
22–24, 26–31, 33, 35, 36, 38–40, 40,  
42, 44, 46–48, 50–52, 54–57, 59–61
- ft\_pca, 16, 18, 20, 22–24, 26–31, 33, 35, 36,  
38–41, 41, 44, 46–48, 50–52, 54–57,

- 59–61
- ft\_polynomial\_expansion, 16, 18, 20, 22–24, 26–31, 33, 35, 36, 38–42, 43, 46–48, 50–52, 54–57, 59–61
- ft\_quantile\_discretizer, 16, 18, 20, 22–24, 26–31, 33, 35, 36, 38–42, 44, 44, 47, 48, 50–52, 54–57, 59–61
- ft\_r\_formula, 16, 18–20, 22–24, 26–31, 33, 35, 36, 38–42, 44, 46–48, 48, 49, 51, 52, 54–57, 59–61, 86, 87, 89, 92, 97, 98, 106, 107, 109, 110, 112, 113, 116, 119, 121, 122, 125–128, 130, 131, 138, 141, 143, 151, 152
- ft\_regex\_tokenizer, 16, 18, 20, 22–24, 26–31, 33, 35, 36, 38–42, 44, 46, 46, 48, 50–52, 54–57, 59–61
- ft\_robust\_scaler, 16, 18, 20, 22–24, 26–31, 33, 35, 36, 38–42, 44, 46, 47, 47, 50–52, 54, 56, 57, 59–61
- ft\_sql\_transformer, 16, 18, 20, 22–24, 26–31, 33, 35, 36, 38–42, 44, 46–48, 50, 50, 52, 54, 56, 57, 59–61
- ft\_standard\_scaler, 16, 18, 20, 22–24, 26–31, 33, 35, 36, 38–42, 44, 46–48, 50, 51, 51, 54, 56, 57, 59–61
- ft\_stop\_words\_remover, 16, 18, 20, 22–24, 26–31, 33, 35, 36, 38–42, 44, 46–48, 50–52, 53, 56, 57, 59–61, 100
- ft\_string\_indexer, 16, 18, 20, 22–24, 26–31, 33, 35, 36, 38–42, 44, 46–48, 50–52, 54, 54, 56, 57, 59–61
- ft\_string\_indexer\_model (ft\_string\_indexer), 54
- ft\_tokenizer, 16, 18, 20, 22–24, 26–31, 33, 35, 36, 38–42, 44, 46–48, 50–52, 54, 56, 56, 57, 59–61, 122
- ft\_vector\_assembler, 16, 18, 20, 22–24, 26–31, 33, 35, 36, 38–42, 44, 46–48, 50–52, 54, 56, 57, 59–61
- ft\_vector\_indexer, 16, 18, 20, 22–24, 26–31, 33, 35, 36, 38–42, 44, 46–48, 50–52, 54, 56, 57, 58, 60, 61
- ft\_vector\_slicer, 16, 18, 20, 22–24, 26–31, 33, 35, 36, 38–42, 44, 46–48, 50–52, 54, 56, 57, 59, 59, 61
- ft\_word2vec, 16, 18, 20, 22–24, 26–31, 33, 35, 36, 38–42, 44, 46–48, 50–52, 54, 56, 57, 59, 60, 60
- full\_join, 61, 61
- full\_join.tbl\_spark (join.tbl\_spark), 75
- generic\_call\_interface, 62
- get\_spark\_sql\_catalog\_implementation, 62
- glance.ml\_model\_aft\_survival\_regression (ml\_survival\_regression\_tidiers), 154
- glance.ml\_model\_als (ml\_als\_tidiers), 91
- glance.ml\_model\_bisecting\_kmeans (ml\_unsupervised\_tidiers), 157
- glance.ml\_model\_decision\_tree\_classification (ml\_tree\_tidiers), 155
- glance.ml\_model\_decision\_tree\_regression (ml\_tree\_tidiers), 155
- glance.ml\_model\_gaussian\_mixture (ml\_unsupervised\_tidiers), 157
- glance.ml\_model\_gbt\_classification (ml\_tree\_tidiers), 155
- glance.ml\_model\_gbt\_regression (ml\_tree\_tidiers), 155
- glance.ml\_model\_generalized\_linear\_regression (ml\_glm\_tidiers), 114
- glance.ml\_model\_isotonic\_regression (ml\_isotonic\_regression\_tidiers), 117
- glance.ml\_model\_kmeans (ml\_unsupervised\_tidiers), 157
- glance.ml\_model\_lda (ml\_lda\_tidiers), 124
- glance.ml\_model\_linear\_regression (ml\_glm\_tidiers), 114
- glance.ml\_model\_linear\_svc (ml\_linear\_svc\_tidiers), 129
- glance.ml\_model\_logistic\_regression (ml\_logistic\_regression\_tidiers), 132
- glance.ml\_model\_multilayer\_perceptron\_classification (ml\_multilayer\_perceptron\_tidiers), 140
- glance.ml\_model\_naive\_bayes (ml\_naive\_bayes\_tidiers), 142
- glance.ml\_model\_pca (ml\_pca\_tidiers), 144
- glance.ml\_model\_random\_forest\_classification (ml\_tree\_tidiers), 155
- glance.ml\_model\_random\_forest\_regression (ml\_tree\_tidiers), 155

- hive\_context (spark-api), 200
- hive\_context\_config, 63
- hof\_aggregate, 63
- hof\_array\_sort, 64
- hof\_exists, 65
- hof\_filter, 65
- hof\_forall, 66
- hof\_map\_filter, 67
- hof\_map\_zip\_with, 68
- hof\_transform, 69
- hof\_transform\_keys, 70
- hof\_transform\_values, 71
- hof\_zip\_with, 71
  
- inner\_join, 72, 72
- inner\_join.tbl\_spark (join.tbl\_spark), 75
- invoke, 73, 200, 223
- invoke\_new (invoke), 73
- invoke\_static (invoke), 73
- is\_ml\_estimator (ml-transform-methods), 82
- is\_ml\_transformer (ml-transform-methods), 82
  
- j\_invoke, 77
- j\_invoke\_new (j\_invoke), 77
- j\_invoke\_static (j\_invoke), 77
- jarray, 73
- java\_context (spark-api), 200
- jfloat, 74
- jfloat\_array, 75
- join.tbl\_spark, 75
- join\_by(), 76, 77
  
- left\_join, 78, 78
- left\_join.tbl\_spark (join.tbl\_spark), 75
- list\_sparklyr\_jars, 78
- livy\_config, 78
- livy\_service\_start, 80
- livy\_service\_stop (livy\_service\_start), 80
  
- ml-params, 80
- ml-persistence, 81
- ml-transform-methods, 82, 162
- ml-tuning, 83
- ml\_aft\_survival\_regression, 85, 99, 111, 114, 117, 126, 128, 131, 139, 142, 144, 153
- ml\_als, 88
- ml\_als\_tidiers, 91
- ml\_approx\_nearest\_neighbors (ft\_lsh\_utils), 33
- ml\_approx\_similarity\_join (ft\_lsh\_utils), 33
- ml\_association\_rules (ml\_fpgrowth), 105
- ml\_binary\_classification\_eval (ml\_evaluator), 102
- ml\_binary\_classification\_evaluator (ml\_evaluator), 102
- ml\_bisecting\_kmeans, 91
- ml\_chisquare\_test, 93
- ml\_classification\_eval (ml\_evaluator), 102
- ml\_clustering\_evaluator, 94
- ml\_compute\_cost (ml\_kmeans), 118
- ml\_compute\_silhouette\_measure (ml\_kmeans), 118
- ml\_corr, 95
- ml\_cross\_validator (ml-tuning), 83
- ml\_decision\_tree (ml\_decision\_tree\_classifier), 96
- ml\_decision\_tree\_classifier, 87, 96, 111, 114, 117, 126, 128, 131, 139, 142, 144, 153
- ml\_decision\_tree\_regressor (ml\_decision\_tree\_classifier), 96
- ml\_default\_stop\_words, 54, 100
- ml\_describe\_topics (ml\_lda), 120
- ml\_evaluate, 101
- ml\_evaluator, 84, 102
- ml\_feature\_importances, 104
- ml\_find\_synonyms (ft\_word2vec), 60
- ml\_fit (ml-transform-methods), 82
- ml\_fit\_and\_transform (ml-transform-methods), 82
- ml\_fpgrowth, 105
- ml\_freq\_itemsets (ml\_fpgrowth), 105
- ml\_freq\_seq\_patterns (ml\_prefixspan), 148
- ml\_gaussian\_mixture, 106
- ml\_gbt\_classifier, 87, 99, 107, 114, 117, 126, 128, 131, 139, 142, 144, 153
- ml\_gbt\_regressor (ml\_gbt\_classifier), 107



- ml\_generalized\_linear\_regression, [87](#),  
[99](#), [111](#), [112](#), [117](#), [126](#), [128](#), [131](#),  
[139](#), [142](#), [144](#), [153](#)
- ml\_glm\_tidiers, [114](#)
- ml\_gradient\_boosted\_trees  
(ml\_gbt\_classifier), [107](#)
- ml\_is\_set (ml-params), [80](#)
- ml\_isotonic\_regression, [87](#), [99](#), [111](#), [114](#),  
[116](#), [126](#), [128](#), [131](#), [139](#), [142](#), [144](#),  
[153](#)
- ml\_isotonic\_regression\_tidiers, [117](#)
- ml\_kmeans, [118](#)
- ml\_kmeans\_cluster\_eval, [120](#)
- ml\_labels (ft\_string\_indexer), [54](#)
- ml\_lda, [120](#)
- ml\_lda\_tidiers, [124](#)
- ml\_linear\_regression, [87](#), [99](#), [111](#), [114](#),  
[117](#), [125](#), [128](#), [131](#), [139](#), [142](#), [144](#),  
[153](#)
- ml\_linear\_svc, [87](#), [99](#), [111](#), [114](#), [117](#), [126](#),  
[127](#), [131](#), [139](#), [142](#), [144](#), [153](#)
- ml\_linear\_svc\_tidiers, [129](#)
- ml\_load (ml-persistence), [81](#)
- ml\_log\_likelihood (ml\_lda), [120](#)
- ml\_log\_perplexity (ml\_lda), [120](#)
- ml\_logistic\_regression, [87](#), [99](#), [111](#), [114](#),  
[117](#), [126](#), [128](#), [129](#), [139](#), [142](#), [144](#),  
[153](#)
- ml\_logistic\_regression\_tidiers, [132](#)
- ml\_metrics\_binary, [133](#)
- ml\_metrics\_multiclass, [134](#)
- ml\_metrics\_regression, [135](#)
- ml\_model\_data, [136](#)
- ml\_multiclass\_classification\_evaluator  
(ml\_evaluator), [102](#)
- ml\_multilayer\_perceptron  
(ml\_multilayer\_perceptron\_classifier),  
[137](#)
- ml\_multilayer\_perceptron\_classifier,  
[87](#), [99](#), [111](#), [114](#), [117](#), [126](#), [128](#), [131](#),  
[137](#), [142](#), [144](#), [153](#)
- ml\_multilayer\_perceptron\_tidiers, [140](#)
- ml\_naive\_bayes, [87](#), [99](#), [111](#), [114](#), [117](#), [126](#),  
[128](#), [131](#), [139](#), [140](#), [144](#), [153](#)
- ml\_naive\_bayes\_tidiers, [142](#)
- ml\_one\_vs\_rest, [87](#), [99](#), [111](#), [114](#), [117](#), [126](#),  
[128](#), [131](#), [139](#), [142](#), [143](#), [153](#)
- ml\_param (ml-params), [80](#)
- ml\_param\_map (ml-params), [80](#)
- ml\_params (ml-params), [80](#)
- ml\_pca (ft\_pca), [41](#)
- ml\_pca\_tidiers, [144](#)
- ml\_pipeline, [145](#)
- ml\_power\_iteration, [145](#)
- ml\_predict (ml-transform-methods), [82](#)
- ml\_prefixspan, [148](#)
- ml\_random\_forest  
(ml\_random\_forest\_classifier),  
[149](#)
- ml\_random\_forest\_classifier, [87](#), [99](#), [111](#),  
[114](#), [117](#), [126](#), [128](#), [131](#), [139](#), [142](#),  
[144](#), [149](#)
- ml\_random\_forest\_regressor  
(ml\_random\_forest\_classifier),  
[149](#)
- ml\_recommend (ml\_als), [88](#)
- ml\_regression\_evaluator (ml\_evaluator),  
[102](#)
- ml\_save (ml-persistence), [81](#)
- ml\_stage, [153](#)
- ml\_stages (ml\_stage), [153](#)
- ml\_sub\_models (ml-tuning), [83](#)
- ml\_summary, [154](#)
- ml\_survival\_regression  
(ml\_aft\_survival\_regression),  
[85](#)
- ml\_survival\_regression\_tidiers, [154](#)
- ml\_topics\_matrix (ml\_lda), [120](#)
- ml\_train\_validation\_split (ml-tuning),  
[83](#)
- ml\_transform (ml-transform-methods), [82](#)
- ml\_tree\_feature\_importance  
(ml\_feature\_importances), [104](#)
- ml\_tree\_tidiers, [155](#)
- ml\_uid, [157](#)
- ml\_unsupervised\_tidiers, [157](#)
- ml\_validation\_metrics (ml-tuning), [83](#)
- ml\_vocabulary (ft\_count\_vectorizer), [20](#)
- mutate, [158](#), [158](#)
- NA, [158](#)
- na.replace, [158](#)
- nest, [158](#), [158](#)
- numeric\_version, [244](#)
- pivot\_longer, [159](#), [159](#)
- pivot\_wider, [159](#), [159](#)

- random\_string, 159
- reactiveSpark, 160
- register\_extension, 161
- registerDoSpark, 160
- registered\_extensions
  - (register\_extension), 161
- replace\_na, 161, 161
- right\_join, 161, 161
- right\_join.tbl\_spark(join.tbl\_spark), 75
  
- sdf-saveload, 162
- sdf-transform-methods, 82, 162
- sdf\_along, 163
- sdf\_bind, 163
- sdf\_bind\_cols(sdf\_bind), 163
- sdf\_bind\_rows(sdf\_bind), 163
- sdf\_broadcast, 164
- sdf\_checkpoint, 165
- sdf\_coalesce, 165
- sdf\_collect, 166
- sdf\_copy\_to, 166, 169, 178, 182, 192, 194, 198
- sdf\_crosstab, 167
- sdf\_debug\_string, 168
- sdf\_describe, 168
- sdf\_dim, 169
- sdf\_distinct, 167, 169, 178, 182, 192, 194, 198
- sdf\_drop\_duplicates, 170
- sdf\_expand\_grid, 170
- sdf\_fit(sdf-transform-methods), 162
- sdf\_fit\_and\_transform
  - (sdf-transform-methods), 162
- sdf\_from\_avro, 171
- sdf\_import(sdf\_copy\_to), 166
- sdf\_is\_streaming, 171
- sdf\_last\_index, 172
- sdf\_len, 172
- sdf\_load\_parquet(sdf-saveload), 162
- sdf\_load\_table(sdf-saveload), 162
- sdf\_ncol(sdf\_dim), 169
- sdf\_nrow(sdf\_dim), 169
- sdf\_num\_partitions, 173
- sdf\_partition(sdf\_random\_split), 177
- sdf\_partition\_sizes, 173
- sdf\_persist, 174
- sdf\_pivot, 174
- sdf\_predict, 94, 103
- sdf\_predict(sdf-transform-methods), 162
- sdf\_project, 175
- sdf\_quantile, 176
- sdf\_random\_split, 167, 169, 177, 182, 192, 194, 198
- sdf\_rbeta, 178, 180, 181, 184–191
- sdf\_rbinom, 179, 179, 180, 181, 184–191
- sdf\_rcauchy, 179, 180, 180, 181, 184–191
- sdf\_rchisq, 179, 180, 181, 184–191
- sdf\_read\_column, 181
- sdf\_register, 167, 169, 178, 182, 192, 194, 198
- sdf\_repartition, 182
- sdf\_residuals
  - (sdf\_residuals.ml\_model\_generalized\_linear\_regression), 183
- sdf\_residuals.ml\_model\_generalized\_linear\_regression, 183
- sdf\_rexp, 179–181, 183, 185–191
- sdf\_rgamma, 179–181, 184, 184, 185–191
- sdf\_rgeom, 179–181, 184, 185, 185, 186–191
- sdf\_rhyper, 179–181, 184, 185, 186, 187–191
- sdf\_rlnorm, 179–181, 184–186, 187, 188–191
- sdf\_rnorm, 179–181, 184–187, 188, 189–191
- sdf\_rpois, 179–181, 184–188, 189, 190, 191
- sdf\_rt, 179–181, 184–189, 189, 191
- sdf\_runif, 179–181, 184–190, 190, 191
- sdf\_rweibull, 179–181, 184–191, 191
- sdf\_sample, 167, 169, 178, 182, 192, 194, 198
- sdf\_save\_parquet(sdf-saveload), 162
- sdf\_save\_table(sdf-saveload), 162
- sdf\_schema, 192
- sdf\_separate\_column, 193
- sdf\_seq, 193
- sdf\_sort, 167, 169, 178, 182, 192, 194, 198
- sdf\_sql, 194
- sdf\_to\_avro, 195
- sdf\_transform(sdf-transform-methods), 162
- sdf\_unnest\_longer, 195
- sdf\_unnest\_wider, 196
- sdf\_weighted\_sample, 167, 169, 178, 182, 192, 194, 198
- sdf\_with\_sequential\_id, 198
- sdf\_with\_unique\_id, 199
- select, 199, 199

- separate, [199](#), [199](#)
- spark-api, [200](#)
- spark-connections, [201](#)
- spark\_adaptive\_query\_execution, [203](#),  
[204](#), [208–210](#), [243](#)
- spark\_advisory\_shuffle\_partition\_size,  
[204](#), [204](#), [208–210](#), [243](#)
- spark\_apply, [204](#)
- spark\_apply\_bundle, [207](#)
- spark\_apply\_log, [207](#)
- spark\_auto\_broadcast\_join\_threshold,  
[204](#), [208](#), [209](#), [210](#), [243](#)
- spark\_available\_versions  
(spark\_install), [222](#)
- spark\_coalesce\_initial\_num\_partitions,  
[204](#), [208](#), [208](#), [209](#), [210](#), [243](#)
- spark\_coalesce\_min\_num\_partitions, [204](#),  
[208](#), [209](#), [209](#), [210](#), [243](#)
- spark\_coalesce\_shuffle\_partitions, [204](#),  
[208](#), [209](#), [209](#), [243](#)
- spark\_compilation\_spec, [210](#)
- spark\_config, [202](#), [211](#), [215](#)
- spark\_config\_kubernetes, [212](#)
- spark\_config\_settings, [213](#)
- spark\_connect (spark-connections), [201](#)
- spark\_connect\_method, [214](#)
- spark\_connection, [213](#)
- spark\_connection\_class, [213](#)
- spark\_connection\_find, [214](#)
- spark\_connection\_is\_open  
(spark-connections), [201](#)
- spark\_context (spark-api), [200](#)
- spark\_context\_config, [215](#)
- spark\_dataframe, [216](#)
- spark\_default\_compilation\_spec, [216](#)
- spark\_dependency, [217](#)
- spark\_dependency\_fallback, [218](#)
- spark\_disconnect (spark-connections),  
[201](#)
- spark\_disconnect\_all  
(spark-connections), [201](#)
- spark\_extension, [218](#)
- spark\_get\_checkpoint\_dir  
(checkpoint\_directory), [10](#)
- spark\_home\_dir, [210](#)
- spark\_home\_set, [219](#)
- spark\_ide\_columns  
(spark\_ide\_connection\_open),  
[219](#)
- spark\_ide\_connection\_actions  
(spark\_ide\_connection\_open),  
[219](#)
- spark\_ide\_connection\_closed  
(spark\_ide\_connection\_open),  
[219](#)
- spark\_ide\_connection\_open, [219](#)
- spark\_ide\_connection\_updated  
(spark\_ide\_connection\_open),  
[219](#)
- spark\_ide\_objects  
(spark\_ide\_connection\_open),  
[219](#)
- spark\_ide\_preview  
(spark\_ide\_connection\_open),  
[219](#)
- spark\_insert\_table, [11](#), [221](#), [225](#), [226](#),  
[228–231](#), [233–237](#), [239–243](#),  
[248–257](#)
- spark\_install, [202](#), [215](#), [222](#)
- spark\_install\_dir (spark\_install), [222](#)
- spark\_install\_tar (spark\_install), [222](#)
- spark\_installed\_versions  
(spark\_install), [222](#)
- spark\_integ\_test\_skip, [223](#)
- spark\_jobj, [216](#), [223](#)
- spark\_jobj\_class, [224](#)
- spark\_last\_error, [224](#)
- spark\_load\_table, [11](#), [221](#), [224](#), [226](#),  
[228–231](#), [233–237](#), [239–243](#),  
[248–257](#)
- spark\_log, [225](#)
- spark\_read, [11](#), [221](#), [225](#), [226](#), [228–231](#),  
[233–237](#), [239–243](#), [248–257](#)
- spark\_read\_avro, [11](#), [221](#), [225](#), [226](#), [227](#),  
[229–231](#), [233–237](#), [239–243](#),  
[248–257](#)
- spark\_read\_binary, [11](#), [221](#), [225](#), [226](#), [228](#),  
[228](#), [230](#), [231](#), [233–237](#), [239–243](#),  
[248–257](#)
- spark\_read\_csv, [11](#), [221](#), [225](#), [226](#), [228](#), [229](#),  
[229](#), [231](#), [233–237](#), [239–243](#),  
[248–257](#)
- spark\_read\_delta, [11](#), [221](#), [225](#), [226](#),  
[228–230](#), [231](#), [233–237](#), [239–243](#),  
[248–257](#)
- spark\_read\_image, [11](#), [221](#), [225](#), [226](#),

- 228–231, 232, 234–237, 239–243, 248–257
- spark\_read\_jdbc, 11, 221, 225, 226, 228–231, 233, 233, 235–237, 239–243, 248–257
- spark\_read\_json, 11, 221, 225, 226, 228–231, 233, 234, 234, 236, 237, 239–243, 248–257
- spark\_read\_libsvm, 11, 221, 225, 226, 228–231, 233–235, 235, 237, 239–243, 248–257
- spark\_read\_orc, 11, 221, 225, 226, 228–231, 233–236, 236, 239–243, 248–257
- spark\_read\_parquet, 11, 221, 225, 226, 228–231, 233–237, 238, 240–243, 248–257
- spark\_read\_source, 11, 221, 225, 226, 228–231, 233–237, 239, 239, 241–243, 248–257
- spark\_read\_table, 11, 221, 225, 226, 228–231, 233–237, 239, 240, 240, 242, 243, 248–257
- spark\_read\_text, 11, 221, 225, 226, 228–231, 233–237, 239–241, 241, 243, 248–257
- spark\_save\_table, 11, 221, 225, 226, 228–231, 233–237, 239–242, 242, 248–257
- spark\_session (spark-api), 200
- spark\_session\_config, 204, 208–210, 243
- spark\_set\_checkpoint\_dir (checkpoint\_directory), 10
- spark\_statistical\_routines, 243
- spark\_submit (spark-connections), 201
- spark\_table\_name, 244
- spark\_uninstall (spark\_install), 222
- spark\_version, 244
- spark\_version\_from\_home, 245
- spark\_web, 245
- spark\_write, 246
- spark\_write\_avro, 11, 221, 225, 226, 228–231, 233–237, 239–243, 247, 249–257
- spark\_write\_csv, 11, 221, 225, 226, 228–231, 233–237, 239–243, 248, 248, 250–257
- spark\_write\_delta, 11, 221, 225, 226, 228–231, 233–237, 239–243, 248, 249, 249, 251–257
- spark\_write\_jdbc, 11, 221, 225, 226, 228–231, 233–237, 239–243, 248–250, 250, 252–257
- spark\_write\_json, 11, 221, 225, 226, 228–231, 233–237, 239–243, 248–251, 251, 253–257
- spark\_write\_orc, 11, 221, 225, 226, 228–231, 233–237, 239–243, 248–252, 252, 254–257
- spark\_write\_parquet, 11, 221, 225, 226, 228–231, 233–237, 239–243, 248–253, 253, 255–257
- spark\_write\_rds, 254
- spark\_write\_source, 11, 221, 225, 226, 228–231, 233–237, 239–243, 248–254, 255, 256, 257
- spark\_write\_table, 11, 221, 225, 226, 228–231, 233–237, 239–243, 248–255, 256, 257
- spark\_write\_text, 11, 221, 225, 226, 228–231, 233–237, 239–243, 248–256, 257
- sparklyr::register\_extension, 202, 215
- sparklyr\_get\_backend\_port, 203
- sql\_variant, 217
- src\_databases, 258
- stream\_find, 258
- stream\_generate\_test, 259
- stream\_id, 259
- stream\_lag, 260
- stream\_name, 261
- stream\_read\_csv, 261
- stream\_read\_delta (stream\_read\_csv), 261
- stream\_read\_json (stream\_read\_csv), 261
- stream\_read\_kafka (stream\_read\_csv), 261
- stream\_read\_orc (stream\_read\_csv), 261
- stream\_read\_parquet (stream\_read\_csv), 261
- stream\_read\_socket (stream\_read\_csv), 261
- stream\_read\_text (stream\_read\_csv), 261
- stream\_render, 263
- stream\_stats, 264
- stream\_stop, 265
- stream\_trigger\_continuous, 265, 266, 269, 271
- stream\_trigger\_interval, 265, 266, 269,

- [271](#)
- [stream\\_view](#), [266](#)
- [stream\\_watermark](#), [267](#)
- [stream\\_write\\_console](#)  
([stream\\_write\\_csv](#)), [267](#)
- [stream\\_write\\_csv](#), [267](#), [271](#)
- [stream\\_write\\_delta](#) ([stream\\_write\\_csv](#)),  
[267](#)
- [stream\\_write\\_json](#) ([stream\\_write\\_csv](#)),  
[267](#)
- [stream\\_write\\_kafka](#) ([stream\\_write\\_csv](#)),  
[267](#)
- [stream\\_write\\_memory](#), [270](#), [270](#)
- [stream\\_write\\_orc](#) ([stream\\_write\\_csv](#)), [267](#)
- [stream\\_write\\_parquet](#)  
([stream\\_write\\_csv](#)), [267](#)
- [stream\\_write\\_text](#) ([stream\\_write\\_csv](#)),  
[267](#)
  
- [tbl\\_cache](#), [271](#)
- [tbl\\_change\\_db](#), [272](#)
- [tbl\\_uncache](#), [272](#)
- [tibble](#), [196](#), [197](#)
- [tidy.ml\\_model\\_aft\\_survival\\_regression](#)  
([ml\\_survival\\_regression\\_tidiers](#)),  
[154](#)
- [tidy.ml\\_model\\_als](#) ([ml\\_als\\_tidiers](#)), [91](#)
- [tidy.ml\\_model\\_bisecting\\_kmeans](#)  
([ml\\_unsupervised\\_tidiers](#)), [157](#)
- [tidy.ml\\_model\\_decision\\_tree\\_classification](#)  
([ml\\_tree\\_tidiers](#)), [155](#)
- [tidy.ml\\_model\\_decision\\_tree\\_regression](#)  
([ml\\_tree\\_tidiers](#)), [155](#)
- [tidy.ml\\_model\\_gaussian\\_mixture](#)  
([ml\\_unsupervised\\_tidiers](#)), [157](#)
- [tidy.ml\\_model\\_gbt\\_classification](#)  
([ml\\_tree\\_tidiers](#)), [155](#)
- [tidy.ml\\_model\\_gbt\\_regression](#)  
([ml\\_tree\\_tidiers](#)), [155](#)
- [tidy.ml\\_model\\_generalized\\_linear\\_regression](#)  
([ml\\_glm\\_tidiers](#)), [114](#)
- [tidy.ml\\_model\\_isotonic\\_regression](#)  
([ml\\_isotonic\\_regression\\_tidiers](#)),  
[117](#)
- [tidy.ml\\_model\\_kmeans](#)  
([ml\\_unsupervised\\_tidiers](#)), [157](#)
- [tidy.ml\\_model\\_lda](#) ([ml\\_lda\\_tidiers](#)), [124](#)
- [tidy.ml\\_model\\_linear\\_regression](#)  
([ml\\_glm\\_tidiers](#)), [114](#)
  
- [tidy.ml\\_model\\_linear\\_svc](#)  
([ml\\_linear\\_svc\\_tidiers](#)), [129](#)
- [tidy.ml\\_model\\_logistic\\_regression](#)  
([ml\\_logistic\\_regression\\_tidiers](#)),  
[132](#)
- [tidy.ml\\_model\\_multilayer\\_perceptron\\_classification](#)  
([ml\\_multilayer\\_perceptron\\_tidiers](#)),  
[140](#)
- [tidy.ml\\_model\\_naive\\_bayes](#)  
([ml\\_naive\\_bayes\\_tidiers](#)), [142](#)
- [tidy.ml\\_model\\_pca](#) ([ml\\_pca\\_tidiers](#)), [144](#)
- [tidy.ml\\_model\\_random\\_forest\\_classification](#)  
([ml\\_tree\\_tidiers](#)), [155](#)
- [tidy.ml\\_model\\_random\\_forest\\_regression](#)  
([ml\\_tree\\_tidiers](#)), [155](#)
- [transform\\_sdf](#), [273](#)
  
- [unite](#), [273](#), [273](#)
- [unnest](#), [273](#), [273](#)