

# Package ‘sparklyr’

November 2, 2017

**Type** Package

**Title** R Interface to Apache Spark

**Version** 0.6.4

**Maintainer** Javier Luraschi <javier@rstudio.com>

**Description** R interface to Apache Spark, a fast and general engine for big data processing, see <<http://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

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

**URL** <http://spark.rstudio.com>

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

**LazyData** TRUE

**RoxygenNote** 6.0.1.9000

**Depends** R (>= 3.1.2)

**Imports** assertthat, base64enc, broom, config (>= 0.2), DBI (>= 0.6-1), dplyr (>= 0.7.2), dbplyr (>= 1.1.0), digest, httr (>= 1.2.1), jsonlite (>= 1.4), lazyeval (>= 0.2.0), methods, openssl (>= 0.8), rappdirs, readr (>= 1.1.0), rlang (>= 0.1), rprojroot, rstudioapi, shiny (>= 1.0.1), withr, xml2

**Suggests** ggplot2, janeaustenr, nycflights13, testthat, RCurl

**NeedsCompilation** no

**Author** Javier Luraschi [aut, cre],  
Kevin Ushey [aut],  
JJ Allaire [aut],  
RStudio [cph],  
The Apache Software Foundation [aut, cph]

**Repository** CRAN

**Date/Publication** 2017-11-02 00:57:51 UTC

**R topics documented:**

checkpoint_directory . . . . .	4
compile_package_jars . . . . .	5
connection_config . . . . .	5
copy_to.spark_connection . . . . .	6
download_scalac . . . . .	6
ensure . . . . .	7
find_scalac . . . . .	8
ft_binarizer . . . . .	8
ft_bucketizer . . . . .	9
ft_count_vectorizer . . . . .	10
ft_discrete_cosine_transform . . . . .	11
ft_elementwise_product . . . . .	11
ft_index_to_string . . . . .	12
ft_one_hot_encoder . . . . .	13
ft_quantile_discretizer . . . . .	13
ft_regex_tokenizer . . . . .	14
ft_sql_transformer . . . . .	15
ft_stop_words_remover . . . . .	15
ft_string_indexer . . . . .	16
ft_tokenizer . . . . .	17
ft_vector_assembler . . . . .	17
hive_context_config . . . . .	18
invoke . . . . .	18
livy_config . . . . .	19
livy_service_start . . . . .	20
ml_als_factorization . . . . .	21
ml_binary_classification_eval . . . . .	22
ml_classification_eval . . . . .	22
ml_create_dummy_variables . . . . .	23
ml_decision_tree . . . . .	24
ml_generalized_linear_regression . . . . .	26
ml_glm_tidiers . . . . .	27
ml_gradient_boosted_trees . . . . .	28
ml_kmeans . . . . .	29
ml_lda . . . . .	30
ml_linear_regression . . . . .	32
ml_logistic_regression . . . . .	33
ml_model . . . . .	34
ml_model_data . . . . .	35
ml_multilayer_perceptron . . . . .	35
ml_naive_bayes . . . . .	36
ml_one_vs_rest . . . . .	37
ml_options . . . . .	38
ml_pca . . . . .	38
ml_prepare_dataframe . . . . .	39
ml_prepare_response_features_intercept . . . . .	40

ml_random_forest . . . . .	42
ml_saveload . . . . .	44
ml_survival_regression . . . . .	44
ml_tree_feature_importance . . . . .	45
na.replace . . . . .	46
register_extension . . . . .	46
sdf-saveload . . . . .	47
sdf_along . . . . .	47
sdf_bind . . . . .	48
sdf_broadcast . . . . .	49
sdf_checkpoint . . . . .	49
sdf_coalesce . . . . .	49
sdf_copy_to . . . . .	50
sdf_dim . . . . .	51
sdf_last_index . . . . .	51
sdf_len . . . . .	52
sdf_mutate . . . . .	52
sdf_num_partitions . . . . .	53
sdf_partition . . . . .	54
sdf_persist . . . . .	55
sdf_pivot . . . . .	56
sdf_predict . . . . .	56
sdf_project . . . . .	57
sdf_quantile . . . . .	57
sdf_read_column . . . . .	58
sdf_register . . . . .	58
sdf_repartition . . . . .	59
sdf_residuals.ml_model_generalized_linear_regression . . . . .	59
sdf_sample . . . . .	60
sdf_schema . . . . .	61
sdf_separate_column . . . . .	61
sdf_seq . . . . .	62
sdf_sort . . . . .	62
sdf_with_sequential_id . . . . .	63
sdf_with_unique_id . . . . .	63
spark-api . . . . .	64
spark-connections . . . . .	65
spark_apply . . . . .	66
spark_apply_bundle . . . . .	67
spark_apply_log . . . . .	67
spark_compilation_spec . . . . .	68
spark_config . . . . .	69
spark_connection . . . . .	69
spark_context_config . . . . .	70
spark_dataframe . . . . .	70
spark_default_compilation_spec . . . . .	71
spark_dependency . . . . .	71
spark_home_set . . . . .	72

spark_install_sync . . . . .	72
spark_jobj . . . . .	73
spark_load_table . . . . .	73
spark_log . . . . .	74
spark_read_csv . . . . .	74
spark_read_jdbc . . . . .	76
spark_read_json . . . . .	76
spark_read_parquet . . . . .	77
spark_read_source . . . . .	78
spark_read_table . . . . .	79
spark_read_text . . . . .	80
spark_save_table . . . . .	81
spark_table_name . . . . .	81
spark_version . . . . .	82
spark_version_from_home . . . . .	82
spark_web . . . . .	83
spark_write_csv . . . . .	83
spark_write_jdbc . . . . .	84
spark_write_json . . . . .	85
spark_write_parquet . . . . .	85
spark_write_source . . . . .	86
spark_write_table . . . . .	87
spark_write_text . . . . .	87
src_databases . . . . .	88
tbl_cache . . . . .	88
tbl_change_db . . . . .	89
tbl_uncache . . . . .	89

## **Index** **90**

---

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

---

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

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

### Value

A tbl\_spark, representing a dplyr-compatible interface to a Spark DataFrame.

---

download\_scalac

*Downloads default Scala Compilers*

---

### 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.

---

ensure                      *Enforce Specific Structure for R Objects*

---

**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.

**Usage**

```
ensure_scalar_integer(object, allow.na = FALSE, allow.null = FALSE,
  default = NULL)
```

```
ensure_scalar_double(object, allow.na = FALSE, allow.null = FALSE,
  default = NULL)
```

```
ensure_scalar_boolean(object, allow.na = FALSE, allow.null = FALSE,
  default = NULL)
```

```
ensure_scalar_character(object, allow.na = FALSE, allow.null = FALSE,
  default = NULL)
```

**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).

---

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	<i>Feature Transformation – Binarizer</i>
--------------	---

---

### 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 = 0.5, ...)
```

### Arguments

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
threshold	The numeric threshold.
...	Optional arguments; currently unused.

**See Also**

See <http://spark.apache.org/docs/latest/ml-features.html> for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: [ft\\_bucketizer](#), [ft\\_count\\_vectorizer](#), [ft\\_discrete\\_cosine\\_transform](#), [ft\\_elementwise\\_product](#), [ft\\_index\\_to\\_string](#), [ft\\_one\\_hot\\_encoder](#), [ft\\_quantile\\_discretizer](#), [ft\\_regex\\_tokenizer](#), [ft\\_stop\\_words\\_remover](#), [ft\\_string\\_indexer](#), [ft\\_tokenizer](#), [ft\\_vector\\_assembler](#), [sdf\\_mutate](#)

---

`ft_bucketizer`*Feature Transformation – Bucketizer*

---

**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, output.col, splits, ...)
```

**Arguments**

<code>x</code>	An object (usually a <code>spark_tbl</code> ) coercable to a Spark DataFrame.
<code>input.col</code>	The name of the input column(s).
<code>output.col</code>	The name of the output column.
<code>splits</code>	A numeric vector of cutpoints, indicating the bucket boundaries.
<code>...</code>	Optional arguments; currently unused.

**See Also**

See <http://spark.apache.org/docs/latest/ml-features.html> for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: [ft\\_binarizer](#), [ft\\_count\\_vectorizer](#), [ft\\_discrete\\_cosine\\_transform](#), [ft\\_elementwise\\_product](#), [ft\\_index\\_to\\_string](#), [ft\\_one\\_hot\\_encoder](#), [ft\\_quantile\\_discretizer](#), [ft\\_regex\\_tokenizer](#), [ft\\_stop\\_words\\_remover](#), [ft\\_string\\_indexer](#), [ft\\_tokenizer](#), [ft\\_vector\\_assembler](#), [sdf\\_mutate](#)

---

 ft\_count\_vectorizer     *Feature Transformation – CountVectorizer*


---

### Description

Extracts a vocabulary from document collections.

### Usage

```
ft_count_vectorizer(x, input.col, output.col, min.df = NULL, min.tf = NULL,
  vocab.size = NULL, vocabulary.only = FALSE, ...)
```

### Arguments

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
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
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).
vocab.size	Build a vocabulary that only considers the top vocab.size terms ordered by term frequency across the corpus.
vocabulary.only	Boolean; should the vocabulary only be returned?
...	Optional arguments; currently unused.

### See Also

See <http://spark.apache.org/docs/latest/ml-features.html> for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: [ft\\_binarizer](#), [ft\\_bucketizer](#), [ft\\_discrete\\_cosine\\_transform](#), [ft\\_elementwise\\_product](#), [ft\\_index\\_to\\_string](#), [ft\\_one\\_hot\\_encoder](#), [ft\\_quantile\\_discretizer](#), [ft\\_regex\\_tokenizer](#), [ft\\_stop\\_words\\_remover](#), [ft\\_string\\_indexer](#), [ft\\_tokenizer](#), [ft\\_vector\\_assembler](#), [sdf\\_mutate](#)

---

`ft_discrete_cosine_transform`*Feature Transformation – Discrete Cosine Transform (DCT)*

---

**Description**

Transform a column in the time domain into another column in the frequency domain.

**Usage**

```
ft_discrete_cosine_transform(x, input.col, output.col, inverse = FALSE, ...)
```

**Arguments**

<code>x</code>	An object (usually a <code>spark_tbl</code> ) coercable to a Spark DataFrame.
<code>input.col</code>	The name of the input column(s).
<code>output.col</code>	The name of the output column.
<code>inverse</code>	Perform inverse DCT?
<code>...</code>	Optional arguments; currently unused.

**See Also**

See <http://spark.apache.org/docs/latest/ml-features.html> for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: [ft\\_binarizer](#), [ft\\_bucketizer](#), [ft\\_count\\_vectorizer](#), [ft\\_elementwise\\_product](#), [ft\\_index\\_to\\_string](#), [ft\\_one\\_hot\\_encoder](#), [ft\\_quantile\\_discretizer](#), [ft\\_regex\\_tokenizer](#), [ft\\_stop\\_words\\_remover](#), [ft\\_string\\_indexer](#), [ft\\_tokenizer](#), [ft\\_vector\\_assembler](#), [sdf\\_mutate](#)

---

`ft_elementwise_product`*Feature Transformation – ElementwiseProduct*

---

**Description**

Computes the element-wise product between two columns. Generally, this is intended as a scaling transformation, where an input vector is scaled by another vector, but this should apply for all element-wise product transformations.

**Usage**

```
ft_elementwise_product(x, input.col, output.col, scaling.col, ...)
```

**Arguments**

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
scaling.col	The column used to scale input.col.
...	Optional arguments; currently unused.

**See Also**

See <http://spark.apache.org/docs/latest/ml-features.html> for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: [ft\\_binarizer](#), [ft\\_bucketizer](#), [ft\\_count\\_vectorizer](#), [ft\\_discrete\\_cosine\\_transform](#), [ft\\_index\\_to\\_string](#), [ft\\_one\\_hot\\_encoder](#), [ft\\_quantile\\_discretizer](#), [ft\\_regex\\_tokenizer](#), [ft\\_stop\\_words\\_remover](#), [ft\\_string\\_indexer](#), [ft\\_tokenizer](#), [ft\\_vector\\_assembler](#), [sdf\\_mutate](#)

---

ft\_index\_to\_string      *Feature Transformation – IndexToString*

---

**Description**

Symmetrically to [ft\\_string\\_indexer](#), `ft_index_to_string` maps a column of label indices back to a column containing the original labels as strings.

**Usage**

```
ft_index_to_string(x, input.col, output.col, ...)
```

**Arguments**

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
...	Optional arguments; currently unused.

**See Also**

See <http://spark.apache.org/docs/latest/ml-features.html> for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: [ft\\_binarizer](#), [ft\\_bucketizer](#), [ft\\_count\\_vectorizer](#), [ft\\_discrete\\_cosine\\_transform](#), [ft\\_elementwise\\_product](#), [ft\\_one\\_hot\\_encoder](#), [ft\\_quantile\\_discretizer](#), [ft\\_regex\\_tokenizer](#), [ft\\_stop\\_words\\_remover](#), [ft\\_string\\_indexer](#), [ft\\_tokenizer](#), [ft\\_vector\\_assembler](#), [sdf\\_mutate](#)

---

 ft\_one\_hot\_encoder      *Feature Transformation – OneHotEncoder*


---

**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.col, output.col, drop.last = TRUE, ...)
```

**Arguments**

<code>x</code>	An object (usually a <code>spark_tbl</code> ) coercable to a Spark DataFrame.
<code>input.col</code>	The name of the input column(s).
<code>output.col</code>	The name of the output column.
<code>drop.last</code>	Boolean; drop the last category?
<code>...</code>	Optional arguments; currently unused.

**See Also**

See <http://spark.apache.org/docs/latest/ml-features.html> for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: [ft\\_binarizer](#), [ft\\_bucketizer](#), [ft\\_count\\_vectorizer](#), [ft\\_discrete\\_cosine\\_transform](#), [ft\\_elementwise\\_product](#), [ft\\_index\\_to\\_string](#), [ft\\_quantile\\_discretizer](#), [ft\\_regex\\_tokenizer](#), [ft\\_stop\\_words\\_remover](#), [ft\\_string\\_indexer](#), [ft\\_tokenizer](#), [ft\\_vector\\_assembler](#), [sdf\\_mutate](#)

---

 ft\_quantile\_discretizer      *Feature Transformation – QuantileDiscretizer*


---

**Description**

Takes a column with continuous features and outputs a column with binned categorical features. The bin ranges are chosen by taking a sample of the data and dividing it into roughly equal parts. The lower and upper bin bounds will be `-Infinity` and `+Infinity`, covering all real values. This attempts to find `numBuckets` partitions based on a sample of the given input data, but it may find fewer depending on the data sample values.

**Usage**

```
ft_quantile_discretizer(x, input.col, output.col, n.buckets = 5L, ...)
```

**Arguments**

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
n.buckets	The number of buckets to use.
...	Optional arguments; currently unused.

**Details**

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

**See Also**

See <http://spark.apache.org/docs/latest/ml-features.html> for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: [ft\\_binarizer](#), [ft\\_bucketizer](#), [ft\\_count\\_vectorizer](#), [ft\\_discrete\\_cosine\\_transform](#), [ft\\_elementwise\\_product](#), [ft\\_index\\_to\\_string](#), [ft\\_one\\_hot\\_encoder](#), [ft\\_regex\\_tokenizer](#), [ft\\_stop\\_words\\_remover](#), [ft\\_string\\_indexer](#), [ft\\_tokenizer](#), [ft\\_vector\\_assembler](#), [sdf\\_mutate](#)

---

ft\_regex\_tokenizer      *Feature Transformation – RegexTokenizer*

---

**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, output.col, pattern, ...)
```

**Arguments**

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
pattern	The regular expression pattern to be used.
...	Optional arguments; currently unused.

**See Also**

See <http://spark.apache.org/docs/latest/ml-features.html> for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: [ft\\_binarizer](#), [ft\\_bucketizer](#), [ft\\_count\\_vectorizer](#), [ft\\_discrete\\_cosine\\_transform](#), [ft\\_elementwise\\_product](#), [ft\\_index\\_to\\_string](#), [ft\\_one\\_hot\\_encoder](#), [ft\\_quantile\\_discretizer](#), [ft\\_stop\\_words\\_remover](#), [ft\\_string\\_indexer](#), [ft\\_tokenizer](#), [ft\\_vector\\_assembler](#), [sdf\\_mutate](#)

---

ft\_sql\_transformer      *Feature Transformation – SQLTransformer*

---

**Description**

Transform a data set using SQL. Use the `__THIS__` placeholder as a proxy for the active table.

**Usage**

```
ft_sql_transformer(x, sql, ...)
```

**Arguments**

x	An object (usually a <code>spark_tbl</code> ) coercable to a Spark DataFrame.
sql	A SQL statement.
...	Optional arguments; currently unused.

---

ft\_stop\_words\_remover      *Feature Transformation – StopWordsRemover*

---

**Description**

A feature transformer that drops all the stop words from the input sequence.

**Usage**

```
ft_stop_words_remover(x, input.col, output.col, ...)
```

**Arguments**

x	An object (usually a <code>spark_tbl</code> ) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
...	Optional arguments; currently unused.

**See Also**

See <http://spark.apache.org/docs/latest/ml-features.html> for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: [ft\\_binarizer](#), [ft\\_bucketizer](#), [ft\\_count\\_vectorizer](#), [ft\\_discrete\\_cosine\\_transform](#), [ft\\_elementwise\\_product](#), [ft\\_index\\_to\\_string](#), [ft\\_one\\_hot\\_encoder](#), [ft\\_quantile\\_discretizer](#), [ft\\_regex\\_tokenizer](#), [ft\\_string\\_indexer](#), [ft\\_tokenizer](#), [ft\\_vector\\_assembler](#), [sdf\\_mutate](#)

---

ft_string_indexer	<i>Feature Transformation – StringIndexer</i>
-------------------	---

---

**Description**

Encode a column of labels into a column of label indices. The indices are in  $[0, \text{numLabels})$ , ordered by label frequencies, with the most frequent label assigned index 0. The transformation can be reversed with [ft\\_index\\_to\\_string](#).

**Usage**

```
ft_string_indexer(x, input.col, output.col, params = NULL, ...)
```

**Arguments**

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
params	An (optional) R environment – when available, the index <-> label mapping generated by the string indexer will be injected into this environment under the labels key.
...	Optional arguments; currently unused.

**See Also**

See <http://spark.apache.org/docs/latest/ml-features.html> for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: [ft\\_binarizer](#), [ft\\_bucketizer](#), [ft\\_count\\_vectorizer](#), [ft\\_discrete\\_cosine\\_transform](#), [ft\\_elementwise\\_product](#), [ft\\_index\\_to\\_string](#), [ft\\_one\\_hot\\_encoder](#), [ft\\_quantile\\_discretizer](#), [ft\\_regex\\_tokenizer](#), [ft\\_stop\\_words\\_remover](#), [ft\\_tokenizer](#), [ft\\_vector\\_assembler](#), [sdf\\_mutate](#)

---

ft_tokenizer	<i>Feature Transformation – Tokenizer</i>
--------------	---

---

**Description**

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

**Usage**

```
ft_tokenizer(x, input.col, output.col, ...)
```

**Arguments**

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
...	Optional arguments; currently unused.

**See Also**

See <http://spark.apache.org/docs/latest/ml-features.html> for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: [ft\\_binarizer](#), [ft\\_bucketizer](#), [ft\\_count\\_vectorizer](#), [ft\\_discrete\\_cosine\\_transform](#), [ft\\_elementwise\\_product](#), [ft\\_index\\_to\\_string](#), [ft\\_one\\_hot\\_encoder](#), [ft\\_quantile\\_discretizer](#), [ft\\_regex\\_tokenizer](#), [ft\\_stop\\_words\\_remover](#), [ft\\_string\\_indexer](#), [ft\\_vector\\_assembler](#), [sdf\\_mutate](#)

---

ft_vector_assembler	<i>Feature Transformation – VectorAssembler</i>
---------------------	---

---

**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.col, output.col, ...)
```

**Arguments**

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
input.col	The name of the input column(s).
output.col	The name of the output column.
...	Optional arguments; currently unused.

**See Also**

See <http://spark.apache.org/docs/latest/ml-features.html> for more information on the set of transformations available for DataFrame columns in Spark.

Other feature transformation routines: `ft_binarizer`, `ft_bucketizer`, `ft_count_vectorizer`, `ft_discrete_cosine_transform`, `ft_elementwise_product`, `ft_index_to_string`, `ft_one_hot_encoder`, `ft_quantile_discretizer`, `ft_regex_tokenizer`, `ft_stop_words_remover`, `ft_string_indexer`, `ft_tokenizer`, `sdf_mutate`

---

`hive_context_config`     *Runtime configuration interface for Hive*

---

**Description**

Retrieves the runtime configuration interface for Hive.

**Usage**

```
hive_context_config(sc)
```

**Arguments**

`sc`                    A `spark_connection`.

---

`invoke`                *Invoke a Method on a JVM Object*

---

**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")
```

---

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,
  custom_headers = list(`X-Requested-By` = "sparklyr"), ...)
```

**Arguments**

config	Optional base configuration
username	The username to use in the Authorization header
password	The password to use in the Authorization header
custom_headers	List of custom headers to append to http requests. Defaults to list("X-Requested-By" = "sparklyr").
...	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:

proxy\_user User to impersonate when starting the session

jars jars to be used in this session  
 py\_files Python files to be used in this session  
 files files to be used in this session  
 driver\_memory Amount of memory to use for the driver process  
 driver\_cores Number of cores to use for the driver process  
 executor\_memory Amount of memory to use per executor process  
 executor\_cores Number of cores to use for each executor  
 num\_executors Number of executors to launch for this session  
 archives Archives to be used in this session  
 queue The name of the YARN queue to which submitted  
 queue The name of this session  
 heartbeat\_timeout Timeout in seconds to which session be orphaned

**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\_als\_factorization *Spark ML – Alternating Least Squares (ALS) matrix factorization.*

---

## Description

Perform alternating least squares matrix factorization on a Spark DataFrame.

## Usage

```
ml_als_factorization(x, rating.column = "rating", user.column = "user",
  item.column = "item", rank = 10L, regularization.parameter = 0.1,
  implicit.preferences = FALSE, alpha = 1, nonnegative = FALSE,
  iter.max = 10L, ml.options = ml_options(), ...)
```

## Arguments

x	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
rating.column	The name of the column containing ratings.
user.column	The name of the column containing user IDs.
item.column	The name of the column containing item IDs.
rank	Rank of the factorization.
regularization.parameter	The regularization parameter.
implicit.preferences	Use implicit preference.
alpha	The parameter in the implicit preference formulation.
nonnegative	Use nonnegative constraints for least squares.
iter.max	The maximum number of iterations to use.
ml.options	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .

## See Also

Other Spark ML routines: [ml\\_decision\\_tree](#), [ml\\_generalized\\_linear\\_regression](#), [ml\\_gradient\\_boosted\\_trees](#), [ml\\_kmeans](#), [ml\\_lda](#), [ml\\_linear\\_regression](#), [ml\\_logistic\\_regression](#), [ml\\_multilayer\\_perceptron](#), [ml\\_naive\\_bayes](#), [ml\\_one\\_vs\\_rest](#), [ml\\_pca](#), [ml\\_random\\_forest](#), [ml\\_survival\\_regression](#)

---

`ml_binary_classification_eval`*Spark ML - Binary Classification Evaluator*

---

**Description**

See the Spark ML Documentation [BinaryClassificationEvaluator](#)

**Usage**

```
ml_binary_classification_eval(predicted_tbl_spark, label, score,  
metric = "areaUnderROC")
```

**Arguments**

<code>predicted_tbl_spark</code>	The result of running <code>sdf_predict</code>
<code>label</code>	Name of column string specifying which column contains the true, indexed labels (ie 0 / 1)
<code>score</code>	Name of column contains the scored probability of a success (ie 1)
<code>metric</code>	The classification metric - one of: <code>areaUnderRoc</code> (default) or <code>areaUnderPR</code> (not available in Spark 2.X)

**Value**

area under the specified curve

---

`ml_classification_eval`*Spark ML - Classification Evaluator*

---

**Description**

See the Spark ML Documentation [MulticlassClassificationEvaluator](#)

**Usage**

```
ml_classification_eval(predicted_tbl_spark, label, predicted_lbl,  
metric = "f1")
```

**Arguments**

predicted_tbl_spark	A tbl_spark object that contains a columns with predicted labels
label	Name of the column that contains the true, indexed label. Support for binary and multi-class labels, column should be of double type (use as.double)
predicted_lbl	Name of the column that contains the predicted label NOT the scored probability. Support for binary and multi-class labels, column should be of double type (use as.double)
metric	A classification metric, for Spark 1.6: f1 (default), precision, recall, weighted-Precision, weightedRecall or accuracy; for Spark 2.X: f1 (default), weighted-Precision, weightedRecall or accuracy

**Value**

see metric

---

ml\_create\_dummy\_variables

*Create Dummy Variables*

---

**Description**

Given a column in a Spark DataFrame, generate a new Spark DataFrame containing dummy variable columns.

**Usage**

```
ml_create_dummy_variables(x, input, reference = NULL, levels = NULL,
  labels = NULL, envir = new.env(parent = emptyenv()))
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
input	The name of the input column.
reference	The reference label. This variable is omitted when generating dummy variables (to avoid perfect multi-collinearity if all dummy variables were to be used in the model fit); to generate dummy variables for all columns this can be explicitly set as NULL.
levels	The set of levels for which dummy variables should be generated. By default, constructs one variable for each unique value occurring in the column specified by input.
labels	An optional R list, mapping values in the input column to column names to be assigned to the associated dummy variable.
envir	An optional R environment; when provided, it will be filled with useful auxiliary information. See <b>Auxiliary Information</b> for more information.

**Details**

The dummy variables are generated in a similar mechanism to `model.matrix`, where categorical variables are expanded into a set of binary (dummy) variables. These dummy variables can be used for regression of categorical variables within the various regression routines provided by sparklyr.

**Auxiliary Information**

The `envir` argument can be used as a mechanism for returning optional information. Currently, the following pieces are returned:

levels: The set of unique values discovered within the input column.  
 columns: The column names generated.

If the `envir` argument is supplied, the names of any dummy variables generated will be included, under the `labels` key.

---

ml\_decision\_tree      *Spark ML – Decision Trees*

---

**Description**

Perform regression or classification using decision trees.

**Usage**

```
ml_decision_tree(x, response, features, impurity = c("auto", "gini",
  "entropy", "variance"), max.bins = 32L, max.depth = 5L,
  min.info.gain = 0, min.rows = 1L, type = c("auto", "regression",
  "classification"), thresholds = NULL, seed = NULL,
  checkpoint.interval = 10L, cache.node.ids = FALSE, max.memory = 256L,
  ml.options = ml_options(), ...)
```

**Arguments**

<code>x</code>	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
<code>response</code>	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When <code>response</code> is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. <code>response ~ feature1 + feature2 + ...</code> . The intercept term can be omitted by using <code>- 1</code> in the model fit.
<code>features</code>	The name of features (terms) to use for the model fit.
<code>impurity</code>	Criterion used for information gain calculation One of <code>'auto'</code> , <code>'gini'</code> , <code>'entropy'</code> , or <code>'variance'</code> . <code>'auto'</code> defaults to <code>'gini'</code> for classification and <code>'variance'</code> for regression.

<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>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>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>min.rows</code>	Minimum number of instances each child must have after split.
<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.
<code>thresholds</code>	Thresholds in multi-class classification to adjust the probability of predicting each class. Vector 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>seed</code>	Seed for random numbers.
<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>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>max.memory</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>ml.options</code>	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
<code>...</code>	Optional arguments. The data argument can be used to specify the data to be used when $x$ is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .

### See Also

Other Spark ML routines: [ml\\_als\\_factorization](#), [ml\\_generalized\\_linear\\_regression](#), [ml\\_gradient\\_boosted\\_trees](#), [ml\\_kmeans](#), [ml\\_lda](#), [ml\\_linear\\_regression](#), [ml\\_logistic\\_regression](#), [ml\\_multilayer\\_perceptron](#), [ml\\_naive\\_bayes](#), [ml\\_one\\_vs\\_rest](#), [ml\\_pca](#), [ml\\_random\\_forest](#), [ml\\_survival\\_regression](#)

---

 ml\_generalized\_linear\_regression

*Spark ML – Generalized Linear Regression*


---

## Description

Perform generalized linear regression on a Spark DataFrame.

## Usage

```
ml_generalized_linear_regression(x, response, features, intercept = TRUE,
  family = gaussian(link = "identity"), weights.column = NULL,
  iter.max = 100L, ml.options = ml_options(), ...)
```

## Arguments

x	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. <code>response ~ feature1 + feature2 + ...</code> . The intercept term can be omitted by using <code>- 1</code> in the model fit.
features	The name of features (terms) to use for the model fit.
intercept	Boolean; should the model be fit with an intercept term?
family	The family / link function to use; analogous to those normally passed in to calls to R's own <code>glm</code> .
weights.column	The name of the column to use as weights for the model fit.
iter.max	The maximum number of iterations to use.
ml.options	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .

## Details

In contrast to `ml_linear_regression()` and `ml_logistic_regression()`, these routines do not allow you to tweak the loss function (e.g. for elastic net regression); however, the model fits returned by this routine are generally richer in regards to information provided for assessing the quality of fit.

**See Also**

Other Spark ML routines: [ml\\_als\\_factorization](#), [ml\\_decision\\_tree](#), [ml\\_gradient\\_boosted\\_trees](#), [ml\\_kmeans](#), [ml\\_lda](#), [ml\\_linear\\_regression](#), [ml\\_logistic\\_regression](#), [ml\\_multilayer\\_perceptron](#), [ml\\_naive\\_bayes](#), [ml\\_one\\_vs\\_rest](#), [ml\\_pca](#), [ml\\_random\\_forest](#), [ml\\_survival\\_regression](#)

---

ml\_glm\_tidiers

*Tidying methods for Spark ML linear models*


---

**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, 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.

**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\_gradient\_boosted\_trees

*Spark ML – Gradient-Boosted Tree*

---

**Description**

Perform regression or classification using gradient-boosted trees.

**Usage**

```
ml_gradient_boosted_trees(x, response, features, impurity = c("auto", "gini",
  "entropy", "variance"), loss.type = c("auto", "logistic", "squared",
  "absolute"), max.bins = 32L, max.depth = 5L, num.trees = 20L,
  min.info.gain = 0, min.rows = 1L, learn.rate = 0.1, sample.rate = 1,
  type = c("auto", "regression", "classification"), thresholds = NULL,
  seed = NULL, checkpoint.interval = 10L, cache.node.ids = FALSE,
  max.memory = 256L, ml.options = ml_options(), ...)
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. <code>response ~ feature1 + feature2 + ...</code> . The intercept term can be omitted by using <code>- 1</code> in the model fit.
features	The name of features (terms) to use for the model fit.
impurity	Criterion used for information gain calculation One of 'auto', 'gini', 'entropy', or 'variance'. 'auto' defaults to 'gini' for classification and 'variance' for regression.
loss.type	Loss function which the algorithm tries to minimize. Defaults to <code>logistic</code> for classification and <code>squared</code> for regression.
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.
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.
num.trees	Number of trees to train ( $\geq 1$ ), defaults to 20.

min.info.gain	Minimum information gain for a split to be considered at a tree node. Should be $\geq 0$ , defaults to 0.
min.rows	Minimum number of instances each child must have after split.
learn.rate	The learning rate or step size, defaults to 0.1.
sample.rate	Fraction of the training data used for learning each decision tree, defaults to 1.0.
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.
thresholds	Thresholds in multi-class classification to adjust the probability of predicting each class. Vector 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.
seed	Seed for random numbers.
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	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.
ml.options	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when $x$ is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .

**See Also**

Other Spark ML routines: [ml\\_als\\_factorization](#), [ml\\_decision\\_tree](#), [ml\\_generalized\\_linear\\_regression](#), [ml\\_kmeans](#), [ml\\_lda](#), [ml\\_linear\\_regression](#), [ml\\_logistic\\_regression](#), [ml\\_multilayer\\_perceptron](#), [ml\\_naive\\_bayes](#), [ml\\_one\\_vs\\_rest](#), [ml\\_pca](#), [ml\\_random\\_forest](#), [ml\\_survival\\_regression](#)

---

ml\_kmeans

*Spark ML – K-Means Clustering*


---

**Description**

Perform k-means clustering on a Spark DataFrame.

**Usage**

```
ml_kmeans(x, centers, iter.max = 100, features = tbl_vars(x),
  compute.cost = TRUE, tolerance = 1e-04, ml.options = ml_options(), ...)
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
centers	The number of cluster centers to compute.
iter.max	The maximum number of iterations to use.
features	The name of features (terms) to use for the model fit.
compute.cost	Whether to compute cost for k-means model using Spark's <code>computeCost</code> .
tolerance	Param for the convergence tolerance for iterative algorithms.
ml.options	Optional arguments, used to affect the model generated. See <code>ml_options</code> for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <code>do</code> .

**Value**

`ml_model` object of class `kmeans` with overloaded `print`, `fitted` and `predict` functions.

**References**

Bahmani et al., Scalable K-Means++, VLDB 2012

**See Also**

For information on how Spark k-means clustering is implemented, please see <http://spark.apache.org/docs/latest/mllib-clustering.html#k-means>.

Other Spark ML routines: `ml_als_factorization`, `ml_decision_tree`, `ml_generalized_linear_regression`, `ml_gradient_boosted_trees`, `ml_lda`, `ml_linear_regression`, `ml_logistic_regression`, `ml_multilayer_perceptron`, `ml_naive_bayes`, `ml_one_vs_rest`, `ml_pca`, `ml_random_forest`, `ml_survival_regression`

---

ml\_lda

*Spark ML – Latent Dirichlet Allocation*


---

**Description**

Fit a Latent Dirichlet Allocation (LDA) model to a Spark DataFrame.

**Usage**

```
ml_lda(x, features = tbl_vars(x), k = length(features), alpha = (50/k) +
  1, beta = 0.1 + 1, optimizer = "online", max.iterations = 20,
  ml.options = ml_options(), ...)
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
features	The name of features (terms) to use for the model fit.
k	The number of topics to estimate.
alpha	Concentration parameter for the prior placed on documents' distributions over topics. This is a singleton which is replicated to a vector of length k in fitting (as currently EM optimizer only supports symmetric distributions, so all values in the vector should be the same). For Expectation-Maximization optimizer values should be > 1.0. By default $\alpha = (50 / k) + 1$ , where 50/k is common in LDA libraries and +1 follows from Asuncion et al. (2009), who recommend a +1 adjustment for EM.
beta	Concentration parameter for the prior placed on topics' distributions over terms. For Expectation-Maximization optimizer value should be > 1.0 and by default $\beta = 0.1 + 1$ , where 0.1 gives a small amount of smoothing and +1 follows Asuncion et al. (2009), who recommend a +1 adjustment for EM.
optimizer	The optimizer, either <code>online</code> for Online Variational Bayes or <code>em</code> for Expectation-Maximization.
max.iterations	Maximum number of iterations.
ml.options	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
...	Optional arguments. The <code>data</code> argument can be used to specify the data to be used when x is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .

**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.

For terminology used in LDA model see [Spark LDA documentation](#).

Expectation-Maximization: Asuncion et al. [On Smoothing and Inference for Topic Models](#). Uncertainty in Artificial Intelligence, 2009.

**References**

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

Asuncion et al. (2009)

**See Also**

Other Spark ML routines: [ml\\_als\\_factorization](#), [ml\\_decision\\_tree](#), [ml\\_generalized\\_linear\\_regression](#), [ml\\_gradient\\_boosted\\_trees](#), [ml\\_kmeans](#), [ml\\_linear\\_regression](#), [ml\\_logistic\\_regression](#), [ml\\_multilayer\\_perceptron](#), [ml\\_naive\\_bayes](#), [ml\\_one\\_vs\\_rest](#), [ml\\_pca](#), [ml\\_random\\_forest](#), [ml\\_survival\\_regression](#)

**Examples**

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

sc <- spark_connect(master = "local")

austen_books <- austen_books()
books_tbl <- sdf_copy_to(sc, austen_books, overwrite = TRUE)
first_tbl <- books_tbl %>% filter(nchar(text) > 0) %>% head(100)

first_tbl %>%
  ft_tokenizer("text", "tokens") %>%
  ft_count_vectorizer("tokens", "features") %>%
  ml_lda("features", k = 4)

## End(Not run)
```

---

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

---

**Description**

Perform linear regression on a Spark DataFrame.

**Usage**

```
ml_linear_regression(x, response, features, intercept = TRUE, alpha = 0,
  lambda = 0, weights.column = NULL, iter.max = 100L,
  ml.options = ml_options(), ...)
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. <code>response ~ feature1 + feature2 + ...</code> . The intercept term can be omitted by using <code>- 1</code> in the model fit.
features	The name of features (terms) to use for the model fit.
intercept	Boolean; should the model be fit with an intercept term?
alpha, lambda	Parameters controlling loss function penalization (for e.g. lasso, elastic net, and ridge regression). See <b>Details</b> for more information.
weights.column	The name of the column to use as weights for the model fit.

iter.max	The maximum number of iterations to use.
ml.options	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .

### Details

Spark implements for both  $L1$  and  $L2$  regularization in linear regression models. See the preamble in the [Spark Classification and Regression](#) documentation for more details on how the loss function is parameterized.

In particular, with alpha set to 1, the parameterization is equivalent to a [lasso](#) model; if alpha is set to 0, the parameterization is equivalent to a [ridge regression](#) model.

### See Also

Other Spark ML routines: [ml\\_als\\_factorization](#), [ml\\_decision\\_tree](#), [ml\\_generalized\\_linear\\_regression](#), [ml\\_gradient\\_boosted\\_trees](#), [ml\\_kmeans](#), [ml\\_lda](#), [ml\\_logistic\\_regression](#), [ml\\_multilayer\\_perceptron](#), [ml\\_naive\\_bayes](#), [ml\\_one\\_vs\\_rest](#), [ml\\_pca](#), [ml\\_random\\_forest](#), [ml\\_survival\\_regression](#)

---

ml\_logistic\_regression

*Spark ML – Logistic Regression*

---

### Description

Perform logistic regression on a Spark DataFrame.

### Usage

```
ml_logistic_regression(x, response, features, intercept = TRUE, alpha = 0,
  lambda = 0, weights.column = NULL, iter.max = 100L,
  ml.options = ml_options(), ...)
```

### Arguments

x	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supported; e.g. <code>response ~ feature1 + feature2 + ...</code> . The intercept term can be omitted by using <code>- 1</code> in the model fit.
features	The name of features (terms) to use for the model fit.
intercept	Boolean; should the model be fit with an intercept term?

alpha, lambda	Parameters controlling loss function penalization (for e.g. lasso, elastic net, and ridge regression). See <b>Details</b> for more information.
weights.column	The name of the column to use as weights for the model fit.
iter.max	The maximum number of iterations to use.
ml.options	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .

### Details

Spark implements for both  $L1$  and  $L2$  regularization in linear regression models. See the preamble in the [Spark Classification and Regression](#) documentation for more details on how the loss function is parameterized.

In particular, with alpha set to 1, the parameterization is equivalent to a [lasso](#) model; if alpha is set to 0, the parameterization is equivalent to a [ridge regression](#) model.

### See Also

Other Spark ML routines: [ml\\_als\\_factorization](#), [ml\\_decision\\_tree](#), [ml\\_generalized\\_linear\\_regression](#), [ml\\_gradient\\_boosted\\_trees](#), [ml\\_kmeans](#), [ml\\_lda](#), [ml\\_linear\\_regression](#), [ml\\_multilayer\\_perceptron](#), [ml\\_naive\\_bayes](#), [ml\\_one\\_vs\\_rest](#), [ml\\_pca](#), [ml\\_random\\_forest](#), [ml\\_survival\\_regression](#)

---

ml\_model

*Create an ML Model Object*

---

### Description

Create an ML model object, wrapping the result of a Spark ML routine call. The generated object will be an R list with S3 classes `c("ml_model_<class>", "ml_model")`.

### Usage

```
ml_model(class, model, ..., .call = sys.call(sys.parent()))
```

### Arguments

class	The name of the machine learning routine used in the encompassing model. Note that the model name generated will be generated as <code>ml_model_&lt;class&gt;</code> ; that is, <code>ml_model</code> will be prefixed.
model	The underlying Spark model object.
...	Additional model information; typically supplied as named values.
.call	The R call used in generating this model object (ie, the top-level R routine that wraps over the associated Spark ML routine). Typically used for print output in e.g. <code>print</code> and <code>summary</code> methods.

---

ml_model_data	<i>Extracts data associated with a Spark ML model</i>
---------------	---

---

**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	<i>Spark ML – Multilayer Perceptron</i>
--------------------------	---

---

**Description**

Creates and trains multilayer perceptron on a Spark DataFrame.

**Usage**

```
ml_multilayer_perceptron(x, response, features, layers, iter.max = 100,
  seed = sample(.Machine$integer.max, 1), ml.options = ml_options(), ...)
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. response ~ feature1 + feature2 + .... The intercept term can be omitted by using - 1 in the model fit.
features	The name of features (terms) to use for the model fit.
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.

<code>iter.max</code>	The maximum number of iterations to use.
<code>seed</code>	A random seed. Set this value if you need your results to be reproducible across repeated calls.
<code>ml.options</code>	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
<code>...</code>	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .

### See Also

Other Spark ML routines: [ml\\_als\\_factorization](#), [ml\\_decision\\_tree](#), [ml\\_generalized\\_linear\\_regression](#), [ml\\_gradient\\_boosted\\_trees](#), [ml\\_kmeans](#), [ml\\_lda](#), [ml\\_linear\\_regression](#), [ml\\_logistic\\_regression](#), [ml\\_naive\\_bayes](#), [ml\\_one\\_vs\\_rest](#), [ml\\_pca](#), [ml\\_random\\_forest](#), [ml\\_survival\\_regression](#)

---

<code>ml_naive_bayes</code>	<i>Spark ML – Naive-Bayes</i>
-----------------------------	-------------------------------

---

### Description

Perform regression or classification using naive bayes.

### Usage

```
ml_naive_bayes(x, response, features, lambda = 0, ml.options = ml_options(),
  ...)
```

### Arguments

<code>x</code>	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
<code>response</code>	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. <code>response ~ feature1 + feature2 + ...</code> . The intercept term can be omitted by using <code>- 1</code> in the model fit.
<code>features</code>	The name of features (terms) to use for the model fit.
<code>lambda</code>	The (Laplace) smoothing parameter. Defaults to zero.
<code>ml.options</code>	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
<code>...</code>	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .

**See Also**

Other Spark ML routines: [ml\\_als\\_factorization](#), [ml\\_decision\\_tree](#), [ml\\_generalized\\_linear\\_regression](#), [ml\\_gradient\\_boosted\\_trees](#), [ml\\_kmeans](#), [ml\\_lda](#), [ml\\_linear\\_regression](#), [ml\\_logistic\\_regression](#), [ml\\_multilayer\\_perceptron](#), [ml\\_one\\_vs\\_rest](#), [ml\\_pca](#), [ml\\_random\\_forest](#), [ml\\_survival\\_regression](#)

---

ml_one_vs_rest	<i>Spark ML – One vs Rest</i>
----------------	-------------------------------

---

**Description**

Perform regression or classification using one vs rest.

**Usage**

```
ml_one_vs_rest(x, classifier, response, features, ml.options = ml_options(),
  ...)
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
classifier	The classifier model. These model objects can be obtained through the use of the <code>only.model</code> parameter supplied with <a href="#">ml_options</a> .
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. <code>response ~ feature1 + feature2 + ...</code> . The intercept term can be omitted by using <code>- 1</code> in the model fit.
features	The name of features (terms) to use for the model fit.
ml.options	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
...	Optional arguments. The <code>data</code> argument can be used to specify the data to be used when <code>x</code> is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .

**See Also**

Other Spark ML routines: [ml\\_als\\_factorization](#), [ml\\_decision\\_tree](#), [ml\\_generalized\\_linear\\_regression](#), [ml\\_gradient\\_boosted\\_trees](#), [ml\\_kmeans](#), [ml\\_lda](#), [ml\\_linear\\_regression](#), [ml\\_logistic\\_regression](#), [ml\\_multilayer\\_perceptron](#), [ml\\_naive\\_bayes](#), [ml\\_pca](#), [ml\\_random\\_forest](#), [ml\\_survival\\_regression](#)

ml\_options

*Options for Spark ML Routines***Description**

Provide this object to the various Spark ML methods, to control certain facets of the model outputs produced.

**Usage**

```
ml_options(id.column = random_string("id"),
  response.column = random_string("response"),
  features.column = random_string("features"),
  output.column = random_string("output"), model.transform = NULL,
  only.model = FALSE, na.action = getOption("na.action", "na.omit"), ...)
```

**Arguments**

id.column	The name to assign to the generated id column.
response.column	The name to assign to the generated response column.
features.column	The name to assign to the generated features column.
output.column	The name to assign to the generated output column.
model.transform	An optional R function that accepts a Spark model and returns a Spark model. This can be used to supply optional Spark model fitting parameters not made available in the sparklyr APIs.
only.model	Boolean; should the Spark model object itself be returned without fitting the actual model? Useful for <a href="#">ml_one_vs_rest</a> .
na.action	An R function, or the name of an R function, indicating how missing values should be handled.
...	Optional arguments, reserved for future expansion.

ml\_pca

*Spark ML – Principal Components Analysis***Description**

Perform principal components analysis on a Spark DataFrame.

**Usage**

```
ml_pca(x, features = tbl_vars(x), k = length(features),
       ml.options = ml_options(), ...)
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
features	The columns to use in the principal components analysis. Defaults to all columns in x.
k	The number of principal components.
ml.options	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .

**See Also**

Other Spark ML routines: [ml\\_als\\_factorization](#), [ml\\_decision\\_tree](#), [ml\\_generalized\\_linear\\_regression](#), [ml\\_gradient\\_boosted\\_trees](#), [ml\\_kmeans](#), [ml\\_lda](#), [ml\\_linear\\_regression](#), [ml\\_logistic\\_regression](#), [ml\\_multilayer\\_perceptron](#), [ml\\_naive\\_bayes](#), [ml\\_one\\_vs\\_rest](#), [ml\\_random\\_forest](#), [ml\\_survival\\_regression](#)

---

`ml_prepare_dataframe` *Prepare a Spark DataFrame for Spark ML Routines*

---

**Description**

This routine prepares a Spark DataFrame for use by Spark ML routines.

**Usage**

```
ml_prepare_dataframe(x, features, response = NULL, ...,
                   ml.options = ml_options(), envir = new.env(parent = emptyenv()))
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
features	The name of features (terms) to use for the model fit.
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. <code>response ~ feature1 + feature2 + ...</code> . The intercept term can be omitted by using <code>- 1</code> in the model fit.

...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .
<code>ml.options</code>	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
<code>envir</code>	An R environment – when supplied, it will be filled with metadata describing the transformations that have taken place.

## Details

Spark DataFrames are prepared through the following transformations:

1. All specified columns are transformed into a numeric data type (using a simple cast for integer / logical columns, and [ft\\_string\\_indexer](#) for strings),
2. The [ft\\_vector\\_assembler](#) is used to combine the specified features into a single 'feature' vector, suitable for use with Spark ML routines.

After calling this function, the `envir` environment (when supplied) will be populated with a set of variables:

`features`: The name of the generated features vector.  
`response`: The name of the generated response vector.  
`labels`: When the response column is a string vector, the [ft\\_string\\_indexer](#) is used to transform the vector into a [0:

## Examples

```
## Not run:
# example of how 'ml_prepare_dataframe' might be used to invoke
# Spark's LinearRegression routine from the 'ml' package
envir <- new.env(parent = emptyenv())
tdf <- ml_prepare_dataframe(df, features, response, envir = envir)

lr <- invoke_new(
  sc,
  "org.apache.spark.ml.regression.LinearRegression"
)

# use generated 'features', 'response' vector names in model fit
model <- lr %>%
  invoke("setFeaturesCol", envir$features) %>%
  invoke("setLabelCol", envir$response)

## End(Not run)
```

---

`ml_prepare_response_features_intercept`

*Pre-process the Inputs to a Spark ML Routine*

---

**Description**

Pre-process / normalize the inputs typically passed to a Spark ML routine.

**Usage**

```
ml_prepare_response_features_intercept(x = NULL, response, features,
  intercept, envir = parent.frame(),
  categorical.transformations = new.env(parent = emptyenv()),
  ml.options = ml_options())
```

```
ml_prepare_features(x, features, envir = parent.frame(),
  ml.options = ml_options())
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. <code>response ~ feature1 + feature2 + ...</code> . The intercept term can be omitted by using <code>- 1</code> in the model fit.
features	The name of features (terms) to use for the model fit.
intercept	Boolean; should the model be fit with an intercept term?
envir	The R environment in which the response, features and intercept bindings should be mutated. (Typically, the parent frame).
categorical.transformations	An R environment used to record what categorical variables were binarized in this procedure. Categorical variables that included in the model formula will be transformed into binary variables, and the generated mappings will be stored in this environment.
ml.options	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.

**Details**

Pre-processing of these inputs typically involves:

1. Handling the case where response is itself a formula describing the model to be fit, thereby extracting the names of the response and features to be used,
2. Splitting categorical features into dummy variables (so they can easily be accommodated + specified in the underlying Spark ML model fit),
3. Mutating the associated variables *in the specified environment*.

Please take heed of the last point, as while this is useful in practice, the behavior will be very surprising if you are not expecting it.

**Examples**

```
## Not run:
# note that ml_prepare_features, by default, mutates the 'features'
# binding in the same environment in which the function was called
local({
  ml_prepare_features(features = ~ x1 + x2 + x3)
  print(features) # c("x1", "x2", "x3")
})

## End(Not run)
```

ml\_random\_forest

*Spark ML – Random Forests***Description**

Perform regression or classification using random forests with a Spark DataFrame.

**Usage**

```
ml_random_forest(x, response, features, col.sample.rate = NULL,
  impurity = c("auto", "gini", "entropy", "variance"), max.bins = 32L,
  max.depth = 5L, min.info.gain = 0, min.rows = 1L, num.trees = 20L,
  sample.rate = 1, thresholds = NULL, seed = NULL, type = c("auto",
  "regression", "classification"), checkpoint.interval = 10L,
  cache.node.ids = FALSE, max.memory = 256L, ml.options = ml_options(),
  ...)
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. <code>response ~ feature1 + feature2 + ...</code> . The intercept term can be omitted by using <code>- 1</code> in the model fit.
features	The name of features (terms) to use for the model fit.
col.sample.rate	The sampling rate of features to consider for splits at each tree node. Defaults to 1/3 for regression and $\sqrt{k}/k$ for classification where $k$ is number of features. For Spark versions prior to 2.0.0, arbitrary sampling rates are not supported, so the input is automatically mapped to one of "onethird", "sqrt", or "log2".
impurity	Criterion used for information gain calculation One of 'auto', 'gini', 'entropy', or 'variance'. 'auto' defaults to 'gini' for classification and 'variance' for regression.

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.
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.info.gain	Minimum information gain for a split to be considered at a tree node. Should be $\geq 0$ , defaults to 0.
min.rows	Minimum number of instances each child must have after split.
num.trees	Number of trees to train ( $\geq 1$ ), defaults to 20.
sample.rate	Fraction of the training data used for learning each decision tree, defaults to 1.0.
thresholds	Thresholds in multi-class classification to adjust the probability of predicting each class. Vector 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.
seed	Seed for random numbers.
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.
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	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.
ml.options	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when $x$ is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .

### See Also

Other Spark ML routines: [ml\\_als\\_factorization](#), [ml\\_decision\\_tree](#), [ml\\_generalized\\_linear\\_regression](#), [ml\\_gradient\\_boosted\\_trees](#), [ml\\_kmeans](#), [ml\\_lda](#), [ml\\_linear\\_regression](#), [ml\\_logistic\\_regression](#), [ml\\_multilayer\\_perceptron](#), [ml\\_naive\\_bayes](#), [ml\\_one\\_vs\\_rest](#), [ml\\_pca](#), [ml\\_survival\\_regression](#)

---

ml_saveload	<i>Save / Load a Spark ML Model Fit</i>
-------------	---

---

**Description**

Save / load a ml\_model fit.

**Usage**

```
ml_load(sc, file, meta = ml_load_meta(file))
```

```
ml_save(model, file, meta = ml_save_meta(model, file))
```

**Arguments**

sc	A spark_connection.
file	The path where the Spark model should be serialized / deserialized.
meta	The path where the R metadata should be serialized / deserialized. Currently, this must be a local filesystem path. Alternatively, this can be an R function that saves / loads the metadata object.
model	A ml_model fit.

**Details**

These functions are currently experimental and not yet ready for production use. Unfortunately, the training summary information for regression fits (linear, logistic, generalized) are currently not serialized as part of the model fit, and so model fits recovered through ml\_load will not work with e.g. fitted, residuals, and so on. Such fits should still be suitable for generating predictions with new data, however.

---

ml_survival_regression	<i>Spark ML – Survival Regression</i>
------------------------	---------------------------------------

---

**Description**

Perform survival regression on a Spark DataFrame, using an Accelerated failure time (AFT) model with potentially right-censored data.

**Usage**

```
ml_survival_regression(x, response, features, intercept = TRUE,
  censor = "censor", iter.max = 100L, ml.options = ml_options(), ...)
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
response	The name of the response vector (as a length-one character vector), or a formula, giving a symbolic description of the model to be fitted. When response is a formula, it is used in preference to other parameters to set the response, features, and intercept parameters (if available). Currently, only simple linear combinations of existing parameters is supposed; e.g. <code>response ~ feature1 + feature2 + ...</code> . The intercept term can be omitted by using <code>- 1</code> in the model fit.
features	The name of features (terms) to use for the model fit.
intercept	Boolean; should the model be fit with an intercept term?
tensor	The name of the vector that provides censoring information. This should be a numeric vector, with 0 marking uncensored data, and 1 marking right-censored data.
iter.max	The maximum number of iterations to use.
ml.options	Optional arguments, used to affect the model generated. See <a href="#">ml_options</a> for more details.
...	Optional arguments. The data argument can be used to specify the data to be used when x is a formula; this allows calls of the form <code>ml_linear_regression(y ~ x, data = tbl)</code> , and is especially useful in conjunction with <a href="#">do</a> .

**See Also**

Other Spark ML routines: [ml\\_als\\_factorization](#), [ml\\_decision\\_tree](#), [ml\\_generalized\\_linear\\_regression](#), [ml\\_gradient\\_boosted\\_trees](#), [ml\\_kmeans](#), [ml\\_lda](#), [ml\\_linear\\_regression](#), [ml\\_logistic\\_regression](#), [ml\\_multilayer\\_perceptron](#), [ml\\_naive\\_bayes](#), [ml\\_one\\_vs\\_rest](#), [ml\\_pca](#), [ml\\_random\\_forest](#)

---

ml\_tree\_feature\_importance

*Spark ML - Feature Importance for Tree Models*

---

**Description**

Spark ML - Feature Importance for Tree Models

**Usage**

```
ml_tree_feature_importance(sc, model)
```

**Arguments**

sc	A <code>spark_connection</code> .
model	An <code>ml_model</code> encapsulating the output from a decision tree.

**Value**

A sorted data frame with feature labels and their relative importance.

---

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.

---

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.

---

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	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
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 <code>spark_connection</code> object.
path	The path where the Spark DataFrame should be saved.

---

sdf_along	<i>Create DataFrame for along Object</i>
-----------	--

---

**Description**

Creates a DataFrame along the given object.

**Usage**

```
sdf_along(sc, along, repartition = NULL)
```

**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.

---

`sdf_bind`*Bind multiple Spark DataFrames by row and column*

---

### Description

`sdf_bind_rows()` and `sdf_bind_cols()` are implementation of the common pattern of `do.call(rbind, sdf)` or `do.call(cbind, sdf)` for binding many Spark DataFrames into one.

### Usage

```
sdf_bind_rows(..., id = NULL)
```

```
sdf_bind_cols(...)
```

### Arguments

<code>...</code>	Spark tbls to combine. Each argument can either be a Spark DataFrame or a list of Spark DataFrames When row-binding, columns are matched by name, and any missing columns will be filled with NA. When column-binding, rows are matched by position, so all data frames must have the same number of rows.
<code>id</code>	Data frame identifier. When <code>id</code> 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 <code>sdf_bind_rows()</code> . 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	<i>Broadcast hint</i>
---------------	-----------------------

---

**Description**

Used to force broadcast hash joins.

**Usage**

```
sdf_broadcast(x)
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, 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	An object coercable to a Spark DataFrame (typically, a tbl_spark).
partitions	number of partitions

---

`sdf_copy_to`*Copy an Object into Spark*

---

**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, ...)
```

```
sdf_import(x, sc, name, memory, repartition, overwrite, ...)
```

**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>...</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\\_partition](#), [sdf\\_predict](#), [sdf\\_register](#), [sdf\\_sample](#), [sdf\\_sort](#)

**Examples**

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

---

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_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                    An object coercable to a Spark DataFrame (typically, a tbl\_spark).

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)
```

**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.

---

sdf_mutate	<i>Mutate a Spark DataFrame</i>
------------	---------------------------------

---

**Description**

Use Spark's **feature transformers** to mutate a Spark DataFrame.

**Usage**

```
sdf_mutate(.data, ...)
```

```
sdf_mutate_(.data, ..., .dots)
```

**Arguments**

.data	A spark_tbl.
...	Named arguments, mapping new column names to the transformation to be applied.
.dots	A named list, mapping output names to transformations.

**Transforming Spark DataFrames**

The family of functions prefixed with `sdf_` generally access the Scala Spark DataFrame API directly, as opposed to the `dplyr` interface which uses Spark SQL. These functions will 'force' any pending SQL in a `dplyr` pipeline, such that the resulting `tbl_spark` object returned will no longer have the attached 'lazy' SQL operations. Note that the underlying Spark DataFrame *does* execute its operations lazily, so that even though the pending set of operations (currently) are not exposed at the R level, these operations will only be executed when you explicitly `collect()` the table.

## See Also

Other feature transformation routines: [ft\\_binarizer](#), [ft\\_bucketizer](#), [ft\\_count\\_vectorizer](#), [ft\\_discrete\\_cosine\\_transform](#), [ft\\_elementwise\\_product](#), [ft\\_index\\_to\\_string](#), [ft\\_one\\_hot\\_encoder](#), [ft\\_quantile\\_discretizer](#), [ft\\_regex\\_tokenizer](#), [ft\\_stop\\_words\\_remover](#), [ft\\_string\\_indexer](#), [ft\\_tokenizer](#), [ft\\_vector\\_assembler](#)

## Examples

```
## Not run:
# using the 'beaver1' dataset, binarize the 'temp' column
data(beavers, package = "datasets")
beaver_tbl <- copy_to(sc, beaver1, "beaver")
beaver_tbl %>%
  mutate(squared = temp ^ 2) %>%
  sdf_mutate(warm = ft_binarizer(squared, 1000)) %>%
  sdf_register("mutated")

# view our newly constructed tbl
head(beaver_tbl)

# note that we have two separate tbls registered
dplyr::src_tbls(sc)

## End(Not run)
```

---

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 An object coercable to a Spark DataFrame (typically, a `tbl_spark`).

---

sdf\_partition                      *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_partition(x, ..., weights = NULL, seed = sample(.Machine$integer.max,
1))
```

### Arguments

x	An object coercable to a Spark DataFrame.
...	Named parameters, mapping table names to weights. The weights will be normalized such that they sum to 1.
weights	An alternate mechanism for supplying weights – when specified, this takes precedence over the ... arguments.
seed	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_partition(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.

### Transforming Spark DataFrames

The family of functions prefixed with `sdf_` generally access the Scala Spark DataFrame API directly, as opposed to the `dplyr` interface which uses Spark SQL. These functions will 'force' any pending SQL in a `dplyr` pipeline, such that the resulting `tbl_spark` object returned will no longer have the attached 'lazy' SQL operations. Note that the underlying Spark DataFrame *does* execute its operations lazily, so that even though the pending set of operations (currently) are not exposed at the R level, these operations will only be executed when you explicitly `collect()` the table.

### See Also

Other Spark data frames: [sdf\\_copy\\_to](#), [sdf\\_predict](#), [sdf\\_register](#), [sdf\\_sample](#), [sdf\\_sort](#)

**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_partition(training = 0.6, test = 0.4)
print(partitions)

# alternate way of specifying weights
weights <- c(training = 0.6, test = 0.4)
diamonds_tbl %>% sdf_partition(weights = weights)

## 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")
```

**Arguments**

<code>x</code>	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
<code>storage.level</code>	The storage level to be used. Please view the <a href="#">Spark Documentation</a> for information on what storage levels are accepted.

**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

<code>x</code>	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
<code>formula</code>	A two-sided R formula of the form <code>x_1 + x_2 + ... ~ y_1</code> . 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.
<code>fun.aggregate</code>	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.

---

sdf\_predict *Model Predictions with Spark DataFrames*

---

### Description

Given a `ml_model` fit alongside a new data set, produce a new Spark DataFrame with predicted values encoded in the "prediction" column.

### Usage

```
sdf_predict(object, newdata, ...)
```

### Arguments

<code>object</code> , <code>newdata</code>	An object coercable to a Spark DataFrame.
<code>...</code>	Optional arguments; currently unused.

### See Also

Other Spark data frames: [sdf\\_copy\\_to](#), [sdf\\_partition](#), [sdf\\_register](#), [sdf\\_sample](#), [sdf\\_sort](#)

---

sdf_project	<i>Project features onto principal components</i>
-------------	---

---

**Description**

Project features onto principal components

**Usage**

```
sdf_project(object, newdata, features = dimnames(object$components)[[1]],
  feature.prefix = "PC", ...)
```

**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.

**Transforming Spark DataFrames**

The family of functions prefixed with `sdf_` generally access the Scala Spark DataFrame API directly, as opposed to the `dplyr` interface which uses Spark SQL. These functions will 'force' any pending SQL in a `dplyr` pipeline, such that the resulting `tbl_spark` object returned will no longer have the attached 'lazy' SQL operations. Note that the underlying Spark DataFrame *does* execute its operations lazily, so that even though the pending set of operations (currently) are not exposed at the R level, these operations will only be executed when you explicitly `collect()` the table.

---

sdf_quantile	<i>Compute (Approximate) Quantiles with a Spark DataFrame</i>
--------------	---

---

**Description**

Given a numeric column within a Spark DataFrame, compute approximate quantiles (to some relative error).

**Usage**

```
sdf_quantile(x, column, probabilities = c(0, 0.25, 0.5, 0.75, 1),
  relative.error = 1e-05)
```

**Arguments**

x	An object coercable to a Spark DataFrame (typically, a tbl_spark).
column	The column for which quantiles should be computed.
probabilities	A numeric vector of probabilities, for which quantiles should be computed.
relative.error	The relative error – lower values imply more precision in the computed quantiles.

---

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	An object coercable to a Spark DataFrame (typically, 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.

## Transforming Spark DataFrames

The family of functions prefixed with `sdf_` generally access the Scala Spark DataFrame API directly, as opposed to the `dplyr` interface which uses Spark SQL. These functions will 'force' any pending SQL in a `dplyr` pipeline, such that the resulting `tbl_spark` object returned will no longer have the attached 'lazy' SQL operations. Note that the underlying Spark DataFrame *does* execute its operations lazily, so that even though the pending set of operations (currently) are not exposed at the R level, these operations will only be executed when you explicitly `collect()` the table.

### See Also

Other Spark data frames: [sdf\\_copy\\_to](#), [sdf\\_partition](#), [sdf\\_predict](#), [sdf\\_sample](#), [sdf\\_sort](#)

---

sdf_repartition	<i>Repartition a Spark DataFrame</i>
-----------------	--------------------------------------

---

### Description

Repartition a Spark DataFrame

### Usage

```
sdf_repartition(x, partitions = NULL, partition_by = NULL)
```

### Arguments

<code>x</code>	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
<code>partitions</code>	number of partitions
<code>partition_by</code>	vector of column names used for partitioning, only supported for Spark 2.0+

---

sdf_residuals.ml_model_generalized_linear_regression	<i>Model Residuals</i>
--	------------------------

---

### 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_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.

**Transforming Spark DataFrames**

The family of functions prefixed with `sdf_` generally access the Scala Spark DataFrame API directly, as opposed to the `dplyr` interface which uses Spark SQL. These functions will 'force' any pending SQL in a `dplyr` pipeline, such that the resulting `tbl_spark` object returned will no longer have the attached 'lazy' SQL operations. Note that the underlying Spark DataFrame *does* execute its operations lazily, so that even though the pending set of operations (currently) are not exposed at the R level, these operations will only be executed when you explicitly `collect()` the table.

**See Also**

Other Spark data frames: [sdf\\_copy\\_to](#), [sdf\\_partition](#), [sdf\\_predict](#), [sdf\\_register](#), [sdf\\_sort](#)

---

sdf_schema	<i>Read the Schema of a Spark DataFrame</i>
------------	---

---

**Description**

Read the schema of a Spark DataFrame.

**Usage**

```
sdf_schema(x)
```

**Arguments**

`x` An object coercable to a Spark DataFrame (typically, a `tbl_spark`).

**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` An object coercable to a Spark DataFrame (typically, 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)
```

**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.

---

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.

**Transforming Spark DataFrames**

The family of functions prefixed with `sdf_` generally access the Scala Spark DataFrame API directly, as opposed to the `dplyr` interface which uses Spark SQL. These functions will 'force' any pending SQL in a `dplyr` pipeline, such that the resulting `tbl_spark` object returned will no longer have the attached 'lazy' SQL operations. Note that the underlying Spark DataFrame *does* execute its operations lazily, so that even though the pending set of operations (currently) are not exposed at the R level, these operations will only be executed when you explicitly `collect()` the table.

**See Also**

Other Spark data frames: [sdf\\_copy\\_to](#), [sdf\\_partition](#), [sdf\\_predict](#), [sdf\\_register](#), [sdf\\_sample](#)

---

`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**

<code>x</code>	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
<code>id</code>	The name of the column to host the generated IDs.
<code>from</code>	The starting value of the id column

---

`sdf_with_unique_id`*Add a Unique ID Column to a Spark DataFrame*

---

**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**

<code>x</code>	An object coercable to a Spark DataFrame (typically, a <code>tbl_spark</code> ).
<code>id</code>	The name of the column to host the generated IDs.

---

`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

`sc`                    A `spark_connection`.

### 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.

## Usage

```
spark_connect(master = "local", spark_home = Sys.getenv("SPARK_HOME"),
  method = c("shell", "livy", "databricks", "test"), app_name = "sparklyr",
  version = NULL, hadoop_version = NULL, config = spark_config(),
  extensions = sparklyr::registered_extensions(), ...)
```

```
spark_connection_is_open(sc)
```

```
spark_disconnect(sc, ...)
```

```
spark_disconnect_all()
```

## 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 be 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. Currently, only "shell" is supported.
app_name	The application name to be used while running in the Spark cluster.
version	The version of Spark to use. Only applicable to "local" Spark connections.
hadoop_version	The version of Hadoop to use. Only applicable to "local" Spark connections.
config	Custom configuration for the generated Spark connection. See <a href="#">spark_config</a> for details.
extensions	Extension 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.
...	Optional arguments; currently unused.
sc	A spark_connection.

**Examples**

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

spark_disconnect(sc)
```

---

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 = colnames(x), memory = TRUE, group_by = NULL,
  packages = TRUE, ...)
```

**Arguments**

x	An object (usually a spark_tbl) coercable to a Spark DataFrame.
f	A function that transforms a data frame partition into a data frame. The function f has signature <code>f(df, group1, group2, ...)</code> where df is a data frame with the data to be processed and group1 to groupN contain the values of the group_by values. When group_by is not specified, f takes only one argument.
columns	A vector of column names or a named vector of column types for the transformed object. Defaults to the names from the original object and adds indexed column names when not enough columns are specified.
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> . For clusters using Livy or Yarn cluster mode, packages must 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 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.
...	Optional arguments; currently unused.

---

spark_apply_bundle	<i>Create Bundle for Spark Apply</i>
--------------------	--------------------------------------

---

**Description**

Creates a bundle of packages for spark\_apply().

**Usage**

```
spark_apply_bundle(packages = TRUE, base_path = getwd())
```

**Arguments**

packages	List of packages to pack or TRUE to pack all.
base_path	Base path used to store the resulting bundle.

---

spark_apply_log	<i>Log Writer for Spark Apply</i>
-----------------	-----------------------------------

---

**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_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)
```

### Arguments

<code>spark_version</code>	The Spark version to build against. This can be left unset if the path to a suitable Spark home is supplied.
<code>spark_home</code>	The path to a Spark home installation. This can be left unset if <code>spark_version</code> is supplied; in such a case, <code>sparklyr</code> will attempt to discover the associated Spark installation using <code>spark_home_dir</code> .
<code>scalac_path</code>	The path to the <code>scalac</code> compiler to be used during compilation of your Spark extension. Note that you should ensure the version of <code>scalac</code> selected matches the version of <code>scala</code> used with the version of Spark you are compiling against.
<code>scala_filter</code>	An optional R function that can be used to filter which <code>scala</code> files are used during compilation. This can be useful if you have auxiliary files that should only be included with certain versions of Spark.
<code>jar_name</code>	The name to be assigned to the generated jar.
<code>jar_path</code>	The path to the <code>jar</code> tool to be used during compilation of your Spark extension.
<code>jar_dep</code>	An optional list of additional jar dependencies.

### 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_connection	<i>Retrieve the Spark Connection Associated with an R Object</i>
------------------	--

---

**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\_context\_config    *Runtime configuration interface for Spark.*

---

**Description**

Retrieves the runtime configuration interface for Spark.

**Usage**

```
spark_context_config(sc)
```

**Arguments**

sc                    A spark\_connection.

---

spark\_dataframe        *Retrieve a Spark DataFrame*

---

**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`*Default Compilation Specification for Spark Extensions*

---

**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**

<code>pkg</code>	The package containing Spark extensions to be compiled.
<code>locations</code>	Additional locations to scan. By default, the directories <code>/opt/scala</code> and <code>/usr/local/scala</code> will be scanned.

---

`spark_dependency`*Define a Spark dependency*

---

**Description**

Define a Spark dependency consisting of a set of custom JARs and Spark packages.

**Usage**

```
spark_dependency(jars = NULL, packages = NULL)
```

**Arguments**

<code>jars</code>	Character vector of full paths to JAR files
<code>packages</code>	Character vector of Spark packages names

**Value**

An object of type `'spark_dependency'`

---

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, verbose = getOption("sparklyr.verbose",
  is.null(path)))
```

**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.
verbose	Logical. Should the function explain what is it doing?

**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_install_sync	<i>helper function to sync sparkinstall project to sparklyr</i>
--------------------	---

---

**Description**

See: <https://github.com/rstudio/spark-install>

**Usage**

```
spark_install_sync(project_path)
```

**Arguments**

project_path	The path to the sparkinstall project
--------------	--------------------------------------

---

spark_jobj	<i>Retrieve a Spark JVM Object Reference</i>
------------	--

---

**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_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://", "s3n://" and "file://" protocols.
options	A list of strings with additional options. See <a href="http://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">http://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: [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [spark\\_write\\_parquet](#), [spark\\_write\\_source](#), [spark\\_write\\_table](#), [spark\\_write\\_text](#)

---

spark_log	<i>View Entries in the Spark Log</i>
-----------	--------------------------------------

---

**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_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, path, header = TRUE, columns = NULL,
infer_schema = TRUE, 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://", "s3n://" 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.
infer_schema	Boolean; should column types be automatically inferred? Requires one extra pass over the data. 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 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 (s3n://), as well as the local file system (file://).

If you are reading from a secure S3 bucket be sure that the AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY environment variables are both defined.

When header is FALSE, the column names are generated with a V prefix; e.g. V1, V2, ....

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [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="http://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">http://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.
...	Optional arguments; currently unused.

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [spark\\_write\\_parquet](#), [spark\\_write\\_source](#), [spark\\_write\\_table](#), [spark\\_write\\_text](#)

---

spark_read_json	<i>Read a JSON file into a Spark DataFrame</i>
-----------------	--

---

**Description**

Read a table serialized in the **JavaScript Object Notation** format into a Spark DataFrame.

**Usage**

```
spark_read_json(sc, name, path, 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://", "s3n://" 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.
...	Optional arguments; currently unused.

**Details**

You can read data from HDFS (hdfs://), S3 (s3n://), as well as the local file system (file://).

If you are reading from a secure S3 bucket be sure that the AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY environment variables are both defined.

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [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, path, 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://", "s3n://" and "file://" protocols.
options	A list of strings with additional options. See <a href="http://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">http://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.
...	Optional arguments; currently unused.

**Details**

You can read data from HDFS (hdfs://), S3 (s3n://), as well as the local file system (file://).

If you are reading from a secure S3 bucket be sure that the AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY environment variables are both defined.

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [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, 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.
source	A data source capable of reading data.
options	A list of strings with additional options. See <a href="http://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">http://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.
...	Optional arguments; currently unused.

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [spark\\_write\\_parquet](#), [spark\\_write\\_source](#), [spark\\_write\\_table](#), [spark\\_write\\_text](#)

---

spark_read_table	<i>Reads from a Spark Table into a Spark DataFrame.</i>
------------------	---

---

**Description**

Reads from a Spark Table into a Spark DataFrame.

**Usage**

```
spark_read_table(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="http://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">http://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.
...	Optional arguments; currently unused.

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [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, path, 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://", "s3n://" 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?
...	Optional arguments; currently unused.

**Details**

You can read data from HDFS (hdfs://), S3 (s3n://), as well as the local file system (file://).

If you are reading from a secure S3 bucket be sure that the AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY environment variables are both defined.

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [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 "hdfs://", "s3n://" and "file://" protocols.
mode	Specifies the behavior when data or table already exists.
options	A list of strings with additional options.

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [spark\\_write\\_parquet](#), [spark\\_write\\_source](#), [spark\\_write\\_table](#), [spark\\_write\\_text](#)

---

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

*Get the Spark Version Associated with a Spark Connection*


---

**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**

spark\_home            The path to a Spark installation.  
 default                The default version to be inferred, in case version lookup failed, e.g. no Spark installation was found at spark\_home.

---

spark_web	<i>Open the Spark web interface</i>
-----------	-------------------------------------

---

**Description**

Open the Spark web interface

**Usage**

```
spark_web(sc, ...)
```

**Arguments**

sc	A spark_connection.
...	Optional arguments; currently unused.

---

spark_write_csv	<i>Write a Spark DataFrame to a CSV</i>
-----------------	---

---

**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://", "s3n://" 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 "hdfs://".
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	Specifies the behavior when data or table already exists.
partition_by	Partitions the output by the given columns on the file system.
...	Optional arguments; currently unused.

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [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	Specifies the behavior when data or table already exists.
options	A list of strings with additional options.
partition_by	Partitions the output by the given columns on the file system.
...	Optional arguments; currently unused.

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_json](#), [spark\\_write\\_parquet](#), [spark\\_write\\_source](#), [spark\\_write\\_table](#), [spark\\_write\\_text](#)

---

spark_write_json	<i>Write a Spark DataFrame to a JSON file</i>
------------------	---

---

### 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://", "s3n://" and "file://" protocols.
mode	Specifies the behavior when data or table already exists.
options	A list of strings with additional options.
partition_by	Partitions the output by the given columns on the file system.
...	Optional arguments; currently unused.

### See Also

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_parquet](#), [spark\\_write\\_source](#), [spark\\_write\\_table](#), [spark\\_write\\_text](#)

---

spark_write_parquet	<i>Write a Spark DataFrame to a Parquet file</i>
---------------------	--

---

### 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://", "s3n://" and "file://" protocols.
mode	Specifies the behavior when data or table already exists.
options	A list of strings with additional options. See <a href="http://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">http://spark.apache.org/docs/latest/sql-programming-guide.html#configuration</a> .
partition_by	Partitions the output by the given columns on the file system.
...	Optional arguments; currently unused.

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [spark\\_write\\_source](#), [spark\\_write\\_table](#), [spark\\_write\\_text](#)

---

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, name, source, 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.
source	A data source capable of reading data.
mode	Specifies the behavior when data or table already exists.
options	A list of strings with additional options.
partition_by	Partitions the output by the given columns on the file system.
...	Optional arguments; currently unused.

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [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	Specifies the behavior when data or table already exists.
options	A list of strings with additional options.
partition_by	Partitions the output by the given columns on the file system.
...	Optional arguments; currently unused.

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [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://", "s3n://" and "file://" protocols.
mode	Specifies the behavior when data or table already exists.
options	A list of strings with additional options.
partition_by	Partitions the output by the given columns on the file system.
...	Optional arguments; currently unused.

**See Also**

Other Spark serialization routines: [spark\\_load\\_table](#), [spark\\_read\\_csv](#), [spark\\_read\\_jdbc](#), [spark\\_read\\_json](#), [spark\\_read\\_parquet](#), [spark\\_read\\_source](#), [spark\\_read\\_table](#), [spark\\_read\\_text](#), [spark\\_save\\_table](#), [spark\\_write\\_csv](#), [spark\\_write\\_jdbc](#), [spark\\_write\\_json](#), [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.

---

tbl_cache	<i>Cache a Spark Table</i>
-----------	----------------------------

---

**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.

# Index

augment.ml\_model\_generalized\_linear\_regressionoft\_vector\_assembler, [9–17](#), [17](#), [40](#), [53](#)  
    (ml\_glm\_tidiers), [27](#)  
augment.ml\_model\_linear\_regression  
    (ml\_glm\_tidiers), [27](#)

checkpoint\_directory, [4](#)  
compile\_package\_jars, [5](#), [68](#), [71](#)  
config, [69](#)  
connection\_config, [5](#)  
copy\_to.spark\_connection, [6](#)  
cut, [9](#)

do, [21](#), [25](#), [26](#), [29–31](#), [33](#), [34](#), [36](#), [37](#), [39](#), [40](#),  
    [43](#), [45](#)  
download\_scalac, [6](#)

ensure, [7](#)  
ensure\_scalar\_boolean (ensure), [7](#)  
ensure\_scalar\_character (ensure), [7](#)  
ensure\_scalar\_double (ensure), [7](#)  
ensure\_scalar\_integer (ensure), [7](#)

find\_scalac, [8](#)  
ft\_binarizer, [8](#), [9–18](#), [53](#)  
ft\_bucketizer, [9](#), [9](#), [10–18](#), [53](#)  
ft\_count\_vectorizer, [9](#), [10](#), [11–18](#), [53](#)  
ft\_discrete\_cosine\_transform, [9](#), [10](#), [11](#),  
    [12–18](#), [53](#)  
ft\_elementwise\_product, [9–11](#), [11](#), [12–18](#),  
    [53](#)  
ft\_index\_to\_string, [9–12](#), [12](#), [13–18](#), [53](#)  
ft\_one\_hot\_encoder, [9–12](#), [13](#), [14–18](#), [53](#)  
ft\_quantile\_discretizer, [9–13](#), [13](#), [15–18](#),  
    [53](#)  
ft\_regex\_tokenizer, [9–14](#), [14](#), [16–18](#), [53](#)  
ft\_sql\_transformer, [15](#)  
ft\_stop\_words\_remover, [9–15](#), [15](#), [16–18](#),  
    [53](#)  
ft\_string\_indexer, [9–16](#), [16](#), [17](#), [18](#), [40](#), [53](#)  
ft\_tokenizer, [9–16](#), [17](#), [18](#), [53](#)

glance.ml\_model\_generalized\_linear\_regression  
    (ml\_glm\_tidiers), [27](#)  
glance.ml\_model\_linear\_regression  
    (ml\_glm\_tidiers), [27](#)  
glm, [26](#)

hive\_context (spark-api), [64](#)  
hive\_context\_config, [18](#)

invoke, [18](#), [64](#), [73](#)  
invoke\_new (invoke), [18](#)  
invoke\_static (invoke), [18](#)

java\_context (spark-api), [64](#)

livy\_config, [19](#)  
livy\_service\_start, [20](#)  
livy\_service\_stop (livy\_service\_start),  
    [20](#)

ml\_als\_factorization, [21](#), [25](#), [27](#), [29–31](#),  
    [33](#), [34](#), [36](#), [37](#), [39](#), [43](#), [45](#)  
ml\_binary\_classification\_eval, [22](#)  
ml\_classification\_eval, [22](#)  
ml\_create\_dummy\_variables, [23](#)  
ml\_decision\_tree, [21](#), [24](#), [27](#), [29–31](#), [33](#), [34](#),  
    [36](#), [37](#), [39](#), [43](#), [45](#)  
ml\_generalized\_linear\_regression, [21](#),  
    [25](#), [26](#), [29–31](#), [33](#), [34](#), [36](#), [37](#), [39](#), [43](#),  
    [45](#)  
ml\_glm\_tidiers, [27](#)  
ml\_gradient\_boosted\_trees, [21](#), [25](#), [27](#), [28](#),  
    [30](#), [31](#), [33](#), [34](#), [36](#), [37](#), [39](#), [43](#), [45](#)  
ml\_kmeans, [21](#), [25](#), [27](#), [29](#), [29](#), [31](#), [33](#), [34](#), [36](#),  
    [37](#), [39](#), [43](#), [45](#)  
ml\_lda, [21](#), [25](#), [27](#), [29](#), [30](#), [30](#), [33](#), [34](#), [36](#), [37](#),  
    [39](#), [43](#), [45](#)  
ml\_linear\_regression, [21](#), [25–27](#), [29–31](#),  
    [32](#), [34](#), [36](#), [37](#), [39](#), [43](#), [45](#)

- ml\_load (ml\_saveload), 44
- ml\_logistic\_regression, 21, 25–27, 29–31, 33, 33, 36, 37, 39, 43, 45
- ml\_model, 30, 34
- ml\_model\_data, 35
- ml\_multilayer\_perceptron, 21, 25, 27, 29–31, 33, 34, 35, 37, 39, 43, 45
- ml\_naive\_bayes, 21, 25, 27, 29–31, 33, 34, 36, 36, 37, 39, 43, 45
- ml\_one\_vs\_rest, 21, 25, 27, 29–31, 33, 34, 36, 37, 37, 38, 39, 43, 45
- ml\_options, 21, 25, 26, 29–31, 33, 34, 36, 37, 38, 39–41, 43, 45
- ml\_pca, 21, 25, 27, 29–31, 33, 34, 36, 37, 38, 43, 45
- ml\_prepare\_dataframe, 39
- ml\_prepare\_features
  - (ml\_prepare\_response\_features\_intercept), 40
- ml\_prepare\_inputs
  - (ml\_prepare\_response\_features\_intercept), 40
- ml\_prepare\_response\_features\_intercept, 40
- ml\_random\_forest, 21, 25, 27, 29–31, 33, 34, 36, 37, 39, 42, 45
- ml\_save (ml\_saveload), 44
- ml\_saveload, 44
- ml\_survival\_regression, 21, 25, 27, 29–31, 33, 34, 36, 37, 39, 43, 44
- ml\_tree\_feature\_importance, 45
- model.matrix, 24
- NA, 46
- na.replace, 46
- numeric\_version, 82
- register\_extension, 46
- registered\_extensions
  - (register\_extension), 46
- sdf-saveload, 47
- sdf\_along, 47
- sdf\_bind, 48
- sdf\_bind\_cols (sdf\_bind), 48
- sdf\_bind\_rows (sdf\_bind), 48
- sdf\_broadcast, 49
- sdf\_checkpoint, 49
- sdf\_coalesce, 49
- sdf\_copy\_to, 50, 54, 56, 59, 60, 62
- sdf\_dim, 51
- sdf\_import (sdf\_copy\_to), 50
- sdf\_last\_index, 51
- sdf\_len, 52
- sdf\_load\_parquet (sdf-saveload), 47
- sdf\_load\_table (sdf-saveload), 47
- sdf\_mutate, 9–18, 52
- sdf\_mutate\_ (sdf\_mutate), 52
- sdf\_ncol (sdf\_dim), 51
- sdf\_nrow (sdf\_dim), 51
- sdf\_num\_partitions, 53
- sdf\_partition, 50, 54, 56, 59, 60, 62
- sdf\_persist, 55
- sdf\_pivot, 56
- sdf\_predict, 50, 54, 56, 59, 60, 62
- sdf\_project, 57
- sdf\_quantile, 57
- sdf\_read\_column, 58
- sdf\_register, 50, 54, 56, 58, 60, 62
- sdf\_repartition, 59
- sdf\_residuals
  - (sdf\_residuals.ml\_model\_generalized\_linear\_regression), 59
- sdf\_residuals.ml\_model\_generalized\_linear\_regression, 59
- sdf\_sample, 50, 54, 56, 59, 60, 62
- sdf\_save\_parquet (sdf-saveload), 47
- sdf\_save\_table (sdf-saveload), 47
- sdf\_schema, 61
- sdf\_separate\_column, 61
- sdf\_seq, 62
- sdf\_sort, 50, 54, 56, 59, 60, 62
- sdf\_with\_sequential\_id, 63
- sdf\_with\_unique\_id, 63
- spark-api, 64
- spark-connections, 65
- spark\_apply, 66
- spark\_apply\_bundle, 67
- spark\_apply\_log, 67
- spark\_compilation\_spec, 68
- spark\_config, 65, 69
- spark\_connect (spark-connections), 65
- spark\_connection, 69
- spark\_connection\_is\_open
  - (spark-connections), 65
- spark\_context (spark-api), 64
- spark\_context\_config, 70

spark\_dataframe, 70  
spark\_default\_compilation\_spec, 71  
spark\_dependency, 71  
spark\_disconnect (spark-connections), 65  
spark\_disconnect\_all  
    (spark-connections), 65  
spark\_get\_checkpoint\_dir  
    (checkpoint\_directory), 4  
spark\_home\_dir, 68  
spark\_home\_set, 72  
spark\_install, 65  
spark\_install\_sync, 72  
spark\_jobj, 70, 73  
spark\_load\_table, 73, 75–81, 84–88  
spark\_log, 74  
spark\_read\_csv, 74, 74, 76–81, 84–88  
spark\_read\_jdbc, 74, 75, 76, 77–81, 84–88  
spark\_read\_json, 74–76, 76, 78–81, 84–88  
spark\_read\_parquet, 74–77, 77, 79–81,  
    84–88  
spark\_read\_source, 74–78, 78, 80, 81,  
    84–88  
spark\_read\_table, 74–79, 79, 80, 81, 84–88  
spark\_read\_text, 74–80, 80, 81, 84–88  
spark\_save\_table, 74–80, 81, 84–88  
spark\_session (spark-api), 64  
spark\_set\_checkpoint\_dir  
    (checkpoint\_directory), 4  
spark\_table\_name, 81  
spark\_version, 82  
spark\_version\_from\_home, 82  
spark\_web, 83  
spark\_write\_csv, 74–81, 83, 84–88  
spark\_write\_jdbc, 74–81, 84, 84, 85–88  
spark\_write\_json, 74–81, 84, 85, 86–88  
spark\_write\_parquet, 74–81, 84, 85, 85,  
    86–88  
spark\_write\_source, 74–81, 84–86, 86, 87,  
    88  
spark\_write\_table, 74–81, 84–86, 87, 88  
spark\_write\_text, 74–81, 84–87, 87  
sparklyr::register\_extension, 65  
src\_databases, 88  
  
tbl\_cache, 88  
tbl\_change\_db, 89  
tbl\_uncache, 89  
tidy.ml\_model\_generalized\_linear\_regression  
    (ml\_glm\_tidiers), 27  
tidy.ml\_model\_linear\_regression  
    (ml\_glm\_tidiers), 27