Package ‘SparkR’

June 6, 2020

Type Package

Version 2.4.6

Title R Front End for ‘Apache Spark’


License Apache License (== 2.0)


BugReports https://spark.apache.org/contributing.html

SystemRequirements Java (== 8)

Depends R (>= 3.0), methods

Suggests knitr, rmarkdown, testthat, e1071, survival


RoxygenNote 7.1.0

VignetteBuilder knitr

NeedsCompilation no

Encoding UTF-8

Author Shivaram Venkataraman [aut, cre],
    Xiangrui Meng [aut],
    Felix Cheung [aut],
    The Apache Software Foundation [aut, cph]

Maintainer Shivaram Venkataraman <shivaram@cs.berkeley.edu>

Repository CRAN

Date/Publication 2020-06-06 04:20:06 UTC
## R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFTSurvivalRegressionModel-class</td>
<td>7</td>
</tr>
<tr>
<td>agg</td>
<td>7</td>
</tr>
<tr>
<td>alias</td>
<td>9</td>
</tr>
<tr>
<td>ALSModel-class</td>
<td>10</td>
</tr>
<tr>
<td>approxQuantile</td>
<td>10</td>
</tr>
<tr>
<td>arrange</td>
<td>11</td>
</tr>
<tr>
<td>as.data.frame</td>
<td>13</td>
</tr>
<tr>
<td>asc</td>
<td>14</td>
</tr>
<tr>
<td>attach,SparkDataFrame-method</td>
<td>15</td>
</tr>
<tr>
<td>avg</td>
<td>16</td>
</tr>
<tr>
<td>awaitTermination</td>
<td>17</td>
</tr>
<tr>
<td>between</td>
<td>18</td>
</tr>
<tr>
<td>BisectingKMeansModel-class</td>
<td>18</td>
</tr>
<tr>
<td>broadcast</td>
<td>19</td>
</tr>
<tr>
<td>cache</td>
<td>20</td>
</tr>
<tr>
<td>cacheTable</td>
<td>21</td>
</tr>
<tr>
<td>cancelJobGroup</td>
<td>22</td>
</tr>
<tr>
<td>cast</td>
<td>22</td>
</tr>
<tr>
<td>checkpoint</td>
<td>23</td>
</tr>
<tr>
<td>clearCache</td>
<td>24</td>
</tr>
<tr>
<td>clearJobGroup</td>
<td>25</td>
</tr>
<tr>
<td>coalesce</td>
<td>25</td>
</tr>
<tr>
<td>collect</td>
<td>26</td>
</tr>
<tr>
<td>colnames</td>
<td>28</td>
</tr>
<tr>
<td>coltypes</td>
<td>29</td>
</tr>
<tr>
<td>column</td>
<td>31</td>
</tr>
<tr>
<td>column_aggregate_functions</td>
<td>32</td>
</tr>
<tr>
<td>column_collection_functions</td>
<td>37</td>
</tr>
<tr>
<td>column_datetime_diff_functions</td>
<td>45</td>
</tr>
<tr>
<td>column_datetime_functions</td>
<td>48</td>
</tr>
<tr>
<td>column_math_functions</td>
<td>54</td>
</tr>
<tr>
<td>column_misc_functions</td>
<td>61</td>
</tr>
<tr>
<td>column_nonaggregate_functions</td>
<td>62</td>
</tr>
<tr>
<td>column_string_functions</td>
<td>68</td>
</tr>
<tr>
<td>column_window_functions</td>
<td>75</td>
</tr>
<tr>
<td>corr</td>
<td>78</td>
</tr>
<tr>
<td>count</td>
<td>79</td>
</tr>
<tr>
<td>cov</td>
<td>80</td>
</tr>
<tr>
<td>createDataFrame</td>
<td>82</td>
</tr>
<tr>
<td>createExternalTable</td>
<td>83</td>
</tr>
<tr>
<td>createOrReplaceTempView</td>
<td>84</td>
</tr>
<tr>
<td>createTable</td>
<td>85</td>
</tr>
<tr>
<td>crossJoin</td>
<td>86</td>
</tr>
<tr>
<td>crosstab</td>
<td>87</td>
</tr>
<tr>
<td>cube</td>
<td>88</td>
</tr>
<tr>
<td>currentDatabase</td>
<td>89</td>
</tr>
</tbody>
</table>
R topics documented:

dapply ................................................................. 90
dapplyCollect ...................................................... 91
DecisionTreeClassificationModel-class .......................... 93
DecisionTreeRegressionModel-class ............................. 93
describe ............................................................. 94
dim ................................................................. 95
distinct ............................................................. 96
drop ................................................................. 97
dropDuplicates ...................................................... 98
dropna ............................................................. 99
donTempTable ...................................................... 101
donTempView ..................................................... 102
dtypes ............................................................. 103
endsWith .......................................................... 104
except .............................................................. 104
exceptAll .......................................................... 105
explain ............................................................. 106
filter ............................................................... 108
first ................................................................. 109
fitted ............................................................... 110
FPGrowthModel-class ............................................. 111
freqItems ........................................................... 112
gapply ............................................................... 113
gapplyCollect ..................................................... 115
GaussianMixtureModel-class ..................................... 118
GBTClassificationModel-class .................................. 118
GBTRegressionModel-class ...................................... 119
GeneralizedLinearRegressionModel-class ...................... 119
getLocalProperty ............................................... 120
getNumPartitions ............................................... 120
glm.formula,ANY,SparkDataFrame-method ..................... 121
GroupedData-class ............................................... 123
group_by ........................................................ 123
hashCode .......................................................... 124
head ............................................................... 125
hint ............................................................... 126
histogram .......................................................... 127
insertInto ......................................................... 128
install.spark ...................................................... 130
intersect ........................................................ 131
intersectAll ....................................................... 132
isActive ........................................................... 133
isLocal ............................................................ 134
IsotonicRegressionModel-class ................................ 135
isStreaming ....................................................... 135
join ............................................................... 136
KMeansModel-class ............................................... 138
KSTest-class ...................................................... 138
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>last .........................................</td>
<td>138</td>
</tr>
<tr>
<td>lastProgress ..................................</td>
<td>139</td>
</tr>
<tr>
<td>LDAModel-class ................................</td>
<td>140</td>
</tr>
<tr>
<td>limit ..........................................</td>
<td>140</td>
</tr>
<tr>
<td>LinearSVCModel-class .........................</td>
<td>141</td>
</tr>
<tr>
<td>listColumns ...................................</td>
<td>142</td>
</tr>
<tr>
<td>listDatabases ..................................</td>
<td>142</td>
</tr>
<tr>
<td>listFunctions ..................................</td>
<td>143</td>
</tr>
<tr>
<td>listTables .....................................</td>
<td>144</td>
</tr>
<tr>
<td>localCheckpoint ..............................</td>
<td>144</td>
</tr>
<tr>
<td>LogisticRegressionModel-class ...............</td>
<td>146</td>
</tr>
<tr>
<td>merge ..........................................</td>
<td>146</td>
</tr>
<tr>
<td>MultilayerPerceptronClassificationModel-class</td>
<td>148</td>
</tr>
<tr>
<td>mutate .........................................</td>
<td>148</td>
</tr>
<tr>
<td>NaiveBayesModel-class .........................</td>
<td>150</td>
</tr>
<tr>
<td>ncol ..........................................</td>
<td>150</td>
</tr>
<tr>
<td>not ............................................</td>
<td>151</td>
</tr>
<tr>
<td>nrow ..........................................</td>
<td>152</td>
</tr>
<tr>
<td>orderBy ........................................</td>
<td>153</td>
</tr>
<tr>
<td>otherwise .....................................</td>
<td>154</td>
</tr>
<tr>
<td>over ...........................................</td>
<td>155</td>
</tr>
<tr>
<td>partitionBy ....................................</td>
<td>156</td>
</tr>
<tr>
<td>persist ........................................</td>
<td>157</td>
</tr>
<tr>
<td>pivot ..........................................</td>
<td>158</td>
</tr>
<tr>
<td>predict ........................................</td>
<td>159</td>
</tr>
<tr>
<td>print.obj ....................................</td>
<td>159</td>
</tr>
<tr>
<td>print.structField ............................</td>
<td>160</td>
</tr>
<tr>
<td>print.structType .............................</td>
<td>160</td>
</tr>
<tr>
<td>printSchema ..................................</td>
<td>161</td>
</tr>
<tr>
<td>queryName .....................................</td>
<td>162</td>
</tr>
<tr>
<td>RandomForestClassificationModel-class .......</td>
<td>163</td>
</tr>
<tr>
<td>RandomForestRegressionModel-class ..........</td>
<td>163</td>
</tr>
<tr>
<td>randomSplit ..................................</td>
<td>164</td>
</tr>
<tr>
<td>rangeBetween ..................................</td>
<td>165</td>
</tr>
<tr>
<td>rbind .........................................</td>
<td>166</td>
</tr>
<tr>
<td>read.df .......................................</td>
<td>167</td>
</tr>
<tr>
<td>read.jdbc .....................................</td>
<td>168</td>
</tr>
<tr>
<td>read.json .....................................</td>
<td>169</td>
</tr>
<tr>
<td>read.ml .......................................</td>
<td>170</td>
</tr>
<tr>
<td>read.orc .....................................</td>
<td>171</td>
</tr>
<tr>
<td>read.parquet ..................................</td>
<td>172</td>
</tr>
<tr>
<td>read.stream ...................................</td>
<td>172</td>
</tr>
<tr>
<td>read.text .....................................</td>
<td>174</td>
</tr>
<tr>
<td>recoverPartitions .............................</td>
<td>175</td>
</tr>
<tr>
<td>refreshByPath ..................................</td>
<td>175</td>
</tr>
<tr>
<td>refreshTable ..................................</td>
<td>176</td>
</tr>
<tr>
<td>registerTempTable ............................</td>
<td>177</td>
</tr>
<tr>
<td>rename ........................................</td>
<td>178</td>
</tr>
</tbody>
</table>
R topics documented:

repartition .................................................. 179
repartitionByRange ........................................ 180
rollup .......................................................... 181
rowsBetween .................................................. 183
sample .......................................................... 184
sampleBy ...................................................... 185
saveAsTable .................................................... 186
schema .......................................................... 187
select ............................................................ 188
selectExpr ...................................................... 190
setCheckpointDir ............................................. 191
setCurrentDatabase .......................................... 192
setJobDescription ........................................... 192
setJobGroup ................................................... 193
setLocalProperty ............................................. 194
setLogLevel .................................................... 194
show ............................................................ 195
showDF ........................................................ 196
spark.addFile ................................................ 197
spark.als ...................................................... 198
spark.bisectingKmeans ...................................... 201
spark.decisionTree .......................................... 203
spark.fpGrowth .............................................. 206
spark.gaussianMixture ...................................... 208
spark.gbt ..................................................... 210
spark.getSparkFiles ........................................ 214
spark.getSparkFilesRootDirectory ......................... 214
spark.glm ..................................................... 215
spark.isoreg .................................................. 218
spark.kmeans ................................................ 220
spark.kstest .................................................. 222
spark.lapply .................................................. 224
spark.lda ...................................................... 225
spark.logit .................................................... 228
spark.mlp ..................................................... 231
spark.naiveBayes .......................................... 234
spark.randomForest ......................................... 236
spark.survreg ................................................ 239
spark.svmLinear ............................................ 241
SparkDataFrame-class ....................................... 243
sparkR.callJMethod ......................................... 244
sparkR.callJStatic .......................................... 245
sparkR.conf ................................................... 246
sparkR.init ..................................................... 247
sparkR.newJObject .......................................... 248
sparkR.session .............................................. 249
sparkR.session.stop ........................................ 251
sparkR.uiWebUrl ............................................. 251
R topics documented:

- `sparkR.version` .................................................. 252
- `sparkRHive.init` .................................................. 252
- `sparkRSQl.init` .................................................... 253
- `sql` .................................................................. 254
- `startsWith` .......................................................... 255
- `status` ................................................................ 255
- `stopQuery` ............................................................ 256
- `storageLevel` ......................................................... 257
- `str` .................................................................... 258
- `StreamingQuery-class` ............................................. 259
- `structField` ........................................................... 259
- `structType` ............................................................. 260
- `subset` ................................................................. 261
- `substr` ................................................................. 263
- `summary` ............................................................... 264
- `tableNames` ........................................................... 265
- `tables` ................................................................. 266
- `tableToDF` ............................................................. 267
- `take` .................................................................. 268
- `toJSON` ................................................................ 269
- `uncacheTable` ....................................................... 270
- `union` ................................................................. 271
- `unionByName` ....................................................... 272
- `unpersist` ............................................................. 273
- `windowOrderBy` ...................................................... 274
- `windowPartitionBy` .................................................. 275
- `WindowSpec-class` ................................................... 276
- `with` ................................................................. 277
- `withColumn` .......................................................... 278
- `withWatermark` ..................................................... 279
- `write.df` ............................................................... 281
- `write.jdbc` ........................................................... 282
- `write.json` ............................................................. 284
- `write.ml` .............................................................. 285
- `write.orc` ............................................................. 286
- `write.parquet` ....................................................... 287
- `write.stream` ........................................................ 288
- `write.text` ............................................................ 290
- `%<=>%` ................................................................ 291
- `%in%` .................................................................. 292

**Index** .................................................................. 294
AFTSurvivalRegressionModel-class

S4 class that represents a AFTSurvivalRegressionModel

Description

S4 class that represents a AFTSurvivalRegressionModel

Arguments

jobj

a Java object reference to the backing Scala AFTSurvivalRegressionWrapper

Note

AFTSurvivalRegressionModel since 2.0.0

agg    summarize

Description

Aggregates on the entire SparkDataFrame without groups. The resulting SparkDataFrame will also contain the grouping columns.
Compute aggregates by specifying a list of columns

Usage

agg(x, ...)
summarize(x, ...)

## S4 method for signature 'GroupedData'
agg(x, ...)
summarize(x, ...)

## S4 method for signature 'SparkDataFrame'
agg(x, ...)
summarize(x, ...)
Arguments

x a SparkDataFrame or GroupedData.

Details

df2 <- agg(df, <column> = <aggFunction>) df2 <- agg(df, newColName = aggFunction(column))

Value

A SparkDataFrame.

Note

agg since 1.4.0
summarize since 1.4.0
agg since 1.4.0
summarize since 1.4.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with().write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

## Not run:
df2 <- agg(df, age = "sum")  # new column name will be created as 'SUM(age#0')
df3 <- agg(df, ageSum = sum(df$age))  # Creates a new column named ageSum
df4 <- summarize(df, ageSum = max(df$age))

## End(Not run)
### Description

Returns a new SparkDataFrame or a Column with an alias set. Equivalent to SQL "AS" keyword.

### Usage

```r
## S4 method for signature 'Column'
alias(object, data)

## S4 method for signature 'SparkDataFrame'
alias(object, data)
```

### Arguments

- **object**: x a SparkDataFrame or a Column
- **data**: new name to use

### Value

a SparkDataFrame or a Column

### Note

- alias(Column) since 1.4.0
- alias(SparkDataFrame) since 2.3.0

### See Also

Other column_func: `between()`, `cast()`, `endsWith()`, `otherwise()`, `over()`, `startsWith()`, `substr()`

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `arrange()`, `as.data.frame()`, `attach()`, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `groupBy()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isLocal()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `ncol()`, `nrow()`, `persist()`, `printSchema()`, `randomSplit()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()`, `sample()`, `saveAsTable()`, `schema()`, `selectExpr()`, `select()`, `showDF()`, `show()`, `storageLevel()`, `str()`, `subset()`, `summary()`, `take()`, `toJSON()`, `unionByName()`, `union()`, `unpersist()`, `withColumn()`, `withWatermark()`, `with()`, `write.df()`, `write.jdbc()`, `write.json()`, `write.orc()`, `write.parquet()`, `write.stream()`, `write.text()`
approxQuantile

Examples

```r
## Not run:
df <- createDataFrame(iris)

head(select(
  df, alias(df$Sepal_Leng, "slength"), alias(df$Petal_Leng, "plength")
))

## End(Not run)
## Not run:
df <- alias(createDataFrame(mtcars), "mtcars")
avg_mpg <- alias(agg(groupBy(df, df$cyl), avg(df$mpg)), "avg_mpg")

head(select(df, column("mtcars.mpg")))
head(join(df, avg_mpg, column("mtcars.cyl") == column("avg_mpg.cyl")))

## End(Not run)
```

ALSModel-class

S4 class that represents an ALSModel

Arguments

- `jobj` a Java object reference to the backing Scala ALSWrapper

Note

ALSModel since 2.1.0

approxQuantile

Calculates the approximate quantiles of numerical columns of a SparkDataFrame

Description

Calculates the approximate quantiles of numerical columns of a SparkDataFrame. The result of this algorithm has the following deterministic bound: If the SparkDataFrame has N elements and if we request the quantile at probability p up to error err, then the algorithm will return a sample x from the SparkDataFrame so that the *exact* rank of x is close to (p * N). More precisely, floor((p - err) * N) <= rank(x) <= ceil((p + err) * N). This method implements a variation of the Greenwald-Khanna algorithm (with some speed optimizations). The algorithm was first present in [http://dx.doi.org/10.1145/375663.375670 Space-efficient Online Computation of Quantile Summaries] by Greenwald and Khanna. Note that NA values will be ignored in numerical columns before calculation. For columns only containing NA values, an empty list is returned.
**arrange**

## Arrange Rows by Variables

**Description**

Sort a SparkDataFrame by the specified column(s).
Usage

arrange(x, col, ...)  
## S4 method for signature 'SparkDataFrame,Column'
arrange(x, col, ..., withinPartitions = FALSE)

## S4 method for signature 'SparkDataFrame,character'
arrange(x, col, ..., decreasing = FALSE, withinPartitions = FALSE)

## S4 method for signature 'SparkDataFrame,characterOrColumn'
orderBy(x, col, ...)

Arguments

x       a SparkDataFrame to be sorted.
col     a character or Column object indicating the fields to sort on
...     additional sorting fields
withinPartitions     a logical argument indicating whether to sort only within each partition
decreasing     a logical argument indicating sorting order for columns when a character vector
                is specified for col

Value

A SparkDataFrame where all elements are sorted.

Note

arrange(SparkDataFrame, Column) since 1.4.0
arrange(SparkDataFrame, character) since 1.4.0
orderBy(SparkDataFrame, characterOrColumn) since 1.4.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), as.data.frame(),
attach,SparkDataFrame-method,broadcast(),cache(),checkpoint(),coalesce(),collect(),
colnames(),coltypes(),createOrReplaceTempView(),crossJoin(),cube().dapplyCollect(),
dapply(),describe(),dim(),distinct(),dropDuplicates(),dropna(),drop(),dtypes(),
exceptAll(),except(),explain(),filter(),first(),gapplyCollect(),gapply(),getNumPartitions(),
group_by(),head(),hint(),histogram(),insertInto(),intersectAll(),intersect(),
isLocal(),isStreaming(),join(),limit(),localCheckpoint(),merge(),mutate(),ncol(),
nrow(),persist(),printSchema(),randomSplit(),rename(),repartitionByRange(),
repartition(),rollup(),sample(),saveAsTable(),schema(),selectExpr(),select(),
showDF(),show(),storageLevel(),str(),subset(),summary(),take(),toJSON(),unionByName(),
union(), unpersist(),withColumn(),withWatermark(),with().write.df().write.jdbc(),
write.json(),write.orc(),write.parquet(),write.stream(),write.text()
### Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
arrange(df, df$col1)
arrange(df, asc(df$col1), desc(abs(df$col2)))
arrange(df, "col1", decreasing = TRUE)
arrange(df, "col1", "col2", decreasing = c(TRUE, FALSE))
arrange(df, "col1", "col2", withinPartitions = TRUE)

## End(Not run)
```

---

**as.data.frame**

Download data from a SparkDataFrame into a R data.frame

**Description**

This function downloads the contents of a SparkDataFrame into an R’s data.frame. Since data.frames are held in memory, ensure that you have enough memory in your system to accommodate the contents.

**Usage**

```r
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

```r
## S4 method for signature 'SparkDataFrame'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

**Arguments**

- `x` a SparkDataFrame.
- `row.names` NULL or a character vector giving the row names for the data frame.
- `optional` If TRUE, converting column names is optional.
- `...` additional arguments to pass to base::as.data.frame.

**Value**

A data.frame.

**Note**

as.data.frame since 1.6.0
See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), attach, SparkDataFrame-methods, broadcast(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
irisDF <- createDataFrame(iris)
df <- as.data.frame(irisDF[irisDF$Species == "setosa", ])

## End(Not run)
```

---

**asc**  
_A set of operations working with SparkDataFrame columns_

### Description

A set of operations working with SparkDataFrame columns

### Usage

- `asc(x)`
- `contains(x, ...)`
- `desc(x)`
- `getField(x, ...)`
- `getItem(x, ...)`
- `isNaN(x)`
- `isNull(x)`
- `isNotNull(x)`
- `isNaN(x)`
- `isNull(x)`
- `isNotNull(x)`

like(x, ...)

rlike(x, ...)

Arguments

x  a Column object.

...  additional argument(s).

Description

The specified SparkDataFrame is attached to the R search path. This means that the SparkDataFrame
is searched by R when evaluating a variable, so columns in the SparkDataFrame can be accessed by
simply giving their names.

Usage

## S4 method for signature 'SparkDataFrame'
attach(
  what,
  pos = 2L,
  name = deparse(substitute(what), backtick = FALSE),
  warn.conflicts = TRUE
)

Arguments

what  (SparkDataFrame) The SparkDataFrame to attach
pos  (integer) Specify position in search() where to attach.
name  (character) Name to use for the attached SparkDataFrame. Names starting with
       package: are reserved for library.
warn.conflicts  (logical) If TRUE, warnings are printed about conflicts from attaching the database,
                unless that SparkDataFrame contains an object

Note

attach since 1.6.0
avg

See Also

detach

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()
See Also

Other aggregate functions: column_aggregate_functions, corr(), count(), cov(), first(), last()

Examples

```r
## Not run: avg(df$c)
```

### Description

Waits for the termination of the query, either by stopQuery or by an error.

### Usage

```r
awaitTermination(x, timeout = NULL)
```

### Arguments

- `x`: a StreamingQuery.
- `timeout`: time to wait in milliseconds, if omitted, wait indefinitely until stopQuery is called or an error has occurred.

### Details

If the query has terminated, then all subsequent calls to this method will return TRUE immediately.

### Value

TRUE if query has terminated within the timeout period; nothing if timeout is not specified.

### Note

awaitTermination(StreamingQuery) since 2.2.0

experimental

### See Also

Other StreamingQuery methods: explain(), isActive(), lastProgress(), queryName(), status(), stopQuery()

### Examples

```r
## Not run: awaitTermination(sq, 10000)
```
Description
Test if the column is between the lower bound and upper bound, inclusive.

Usage
between(x, bounds)

## S4 method for signature 'Column'
between(x, bounds)

Arguments
x a Column
bounds lower and upper bounds

Note
between since 1.5.0

See Also
Other column_func: alias(), cast(), endsWith(), otherwise(), over(), startsWith(), substr()

---

BisectingKMeansModel-class
S4 class that represents a BisectingKMeansModel

Description
S4 class that represents a BisectingKMeansModel

Arguments
jobj a Java object reference to the backing Scala BisectingKMeansModel

Note
BisectingKMeansModel since 2.2.0
Description

Return a new SparkDataFrame marked as small enough for use in broadcast joins.

Usage

broadcast(x)

## S4 method for signature 'SparkDataFrame'

broadcast(x)

Arguments

x a SparkDataFrame.

Details

Equivalent to `hint(x,"broadcast")`.

Value

a SparkDataFrame.

Note

broadcast since 2.3.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()
Examples

```r
## Not run:
df <- createDataFrame(mtcars)
avg_mpg <- mean(groupBy(createDataFrame(mtcars), "cyl"), "mpg")

head(join(df, broadcast(avg_mpg), df$cyl == avg_mpg$cyl))
## End(Not run)
```

---

**cache**

**Cache**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persist with the default storage level (MEMORY_ONLY).</td>
</tr>
</tbody>
</table>

**Usage**

```r
cache(x)
```

### S4 method for signature 'SparkDataFrame'

```r
cache(x)
```

**Arguments**

- **x** A SparkDataFrame

**Note**

cache since 1.4.0

**See Also**

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach()`, `SparkDataFrame-method`, `broadcast()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `group_by()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isLocal()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `ncol()`, `nrow()`, `persist()`, `printSchema()`, `randomSplit()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()`, `sample()`, `saveAsTable()`, `schema()`, `selectExpr()`, `select()`, `showDF()`, `show()`, `storageLevel()`, `str()`, `subset()`, `summary()`, `take()`, `toJSON()`, `unionByName()`, `union()`, `unpersist()`, `withColumn()`, `withWatermark()`, `with()`, `write.df()`, `write.jdbc()`, `write.json()`, `write.orc()`, `write.parquet()`, `write.stream()`, `write.text()`
Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
cache(df)

## End(Not run)
```

---

<table>
<thead>
<tr>
<th>cacheTable</th>
<th>Cache Table</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
<tr>
<td>Caches the specified table in-memory.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Usage</strong></th>
</tr>
</thead>
</table>
| ```r
## Default S3 method:
cacheTable(tableName)
``` |

<table>
<thead>
<tr>
<th><strong>Arguments</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tableName</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkDataFrame</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Note</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cacheTable</code> since 1.4.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Examples</strong></th>
</tr>
</thead>
</table>
| ```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
createOrReplaceTempView(df, "table")
createTable("table")

## End(Not run)
``` |
**cancelJobGroup**  
*Cancel active jobs for the specified group*

**Description**

Cancel active jobs for the specified group

**Usage**

```r
## Default S3 method:
cancelJobGroup(groupId)
```

**Arguments**

- **groupId**
  the ID of job group to be cancelled

**Note**

cancelJobGroup since 1.5.0

**Examples**

```r
## Not run:
sparkR.session()
cancelJobGroup("myJobGroup")
## End(Not run)
```

---

**cast**  
*Casts the column to a different data type.*

**Description**

Casts the column to a different data type.

**Usage**

```r
cast(x, dataType)
```

**Arguments**

- **x**
  a Column.
- **dataType**
  a character object describing the target data type. See Spark Data Types for available data types.
checkpoint

Note

cast since 1.4.0

See Also

Other column_func: alias(), between(), endsWith(), otherwise(), over(), startsWith(), substr()

Examples

```r
## Not run:
   cast(df$age, "string")

## End(Not run)
```

Description

Returns a checkpointed version of this SparkDataFrame. Checkpointing can be used to truncate the logical plan, which is especially useful in iterative algorithms where the plan may grow exponentially. It will be saved to files inside the checkpoint directory set with setCheckpointDir.

Usage

```r
checkpoint(x, eager = TRUE)
```

## S4 method for signature 'SparkDataFrame'

```r
checkpoint(x, eager = TRUE)
```

Arguments

- `x` A SparkDataFrame
- `eager` whether to checkpoint this SparkDataFrame immediately

Value

a new checkpointed SparkDataFrame

Note

checkpoint since 2.2.0
clearCache

Clear Cache

Description

Removes all cached tables from the in-memory cache.

Usage

```r
## Default S3 method:
clearCache()
```

Note

clearCache since 1.4.0

Examples

```r
## Not run:
clearCache()
## End(Not run)
```
clearJobGroup | Clear current job group ID and its description

Description
Clear current job group ID and its description

Usage
```r
## Default S3 method:
clearJobGroup()
```

Note
clearJobGroup since 1.5.0

Examples
```r
## Not run:
sparkR.session()
clearJobGroup()
## End(Not run)
```

coalesce | Coalesce

Description
Returns a new SparkDataFrame that has exactly `numPartitions` partitions. This operation results in a narrow dependency, e.g. if you go from 1000 partitions to 100 partitions, there will not be a shuffle, instead each of the 100 new partitions will claim 10 of the current partitions. If a larger number of partitions is requested, it will stay at the current number of partitions.

Usage
```r
coalesce(x, ...)
```

Arguments
- `x` a SparkDataFrame.
- `...` additional argument(s).
- `numPartitions` the number of partitions to use.
Details

However, if you’re doing a drastic coalesce on a SparkDataFrame, e.g. to `numPartitions = 1`, this may result in your computation taking place on fewer nodes than you like (e.g. one node in the case of `numPartitions = 1`). To avoid this, call `repartition`. This will add a shuffle step, but means the current upstream partitions will be executed in parallel (per whatever the current partitioning is).

Note

`coalesce(SparkDataFrame)` since 2.1.1

See Also

`repartition`, `repartitionByRange`

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach`, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `group_by()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isLocal()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `nrow()`, `persist()`, `printSchema()`, `randomSplit()`, `rbind()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()`, `sample()`, `saveAsTable()`, `schema()`, `selectExpr()`, `select()`, `showDF()`, `show()`, `storageLevel()`, `str()`, `subset()`, `summary()`, `take()`, `toJSON()`, `unionByName()` `union()`, `unpersist()`, `withColumn()`, `withWatermark()`, `with()`, `write.df()`, `write.jdbc()`, `write.json()`, `write.orc()`, `write.parquet()`, `write.stream()`, `write.text()`

Examples

```r
## Not run:
sparkR.session()
pth <- "path/to/file.json"
df <- read.json(pth)
newDF <- coalesce(df, 1L)
## End(Not run)
```

---

**collect**

Collects all the elements of a SparkDataFrame and coerces them into an R data.frame.

**Description**

Collects all the elements of a SparkDataFrame and coerces them into an R data.frame.
collect(x, ...)  
## S4 method for signature 'SparkDataFrame'  
collect(x, stringsAsFactors = FALSE)

Arguments

x  
a SparkDataFrame.

...  
further arguments to be passed to or from other methods.

stringsAsFactors  
(Optional) a logical indicating whether or not string columns should be converted to factors. FALSE by default.

Note

collect since 1.4.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByNames(), union(), unpersist(), withColumn(), withWatermark(), with().write.df(), .write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()
colnames  

Column Names of SparkDataFrame

Description

Return a vector of column names.

Usage

```r
colnames(x, do.NULL = TRUE, prefix = "col")

colnames(x) <- value

columns(x)

# S4 method for signature 'SparkDataFrame'
columns(x)

# S4 method for signature 'SparkDataFrame'
names(x)

# S4 replacement method for signature 'SparkDataFrame'
names(x) <- value

# S4 method for signature 'SparkDataFrame'
colnames(x)

# S4 replacement method for signature 'SparkDataFrame'
colnames(x) <- value
```

Arguments

- `x`  
a SparkDataFrame.
- `do.NULL`  
currently not used.
- `prefix`  
currently not used.
- `value`  
a character vector. Must have the same length as the number of columns to be renamed.

Note

columns since 1.4.0  
names since 1.5.0  
names<- since 1.5.0  
colnames since 1.6.0  
colnames<- since 1.6.0
See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
columns(df)
colnames(df)
## End(Not run)
```

coltypes  coltypes

description

Get column types of a SparkDataFrame

Set the column types of a SparkDataFrame.

Usage

```r
coltypes(x)
coltypes(x) <- value
```

## S4 method for signature 'SparkDataFrame'
coltypes(x)

## S4 replacement method for signature 'SparkDataFrame,character'
coltypes(x) <- value
Arguments

x  A SparkDataFrame

value A character vector with the target column types for the given SparkDataFrame. Column types can be one of integer, numeric/double, character, logical, or NA to keep that column as-is.

Value

value A character vector with the column types of the given SparkDataFrame

Note

coltypes since 1.6.0
coltypes<- since 1.6.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

## Not run:
irisDF <- createDataFrame(iris)
coltypes(irisDF) # get column types

## End(Not run)

## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
coltypes(df) <- c("character", "integer") # set column types
coltypes(df) <- c(NA, "numeric") # set column types

## End(Not run)
Description

The column class supports unary, binary operations on SparkDataFrame columns.

Returns a Column based on the given column name.

Usage

column(x)

## S4 method for signature 'jobi'
column(x)

## S4 method for signature 'character'
column(x)

Arguments

x Character column name.

Slots

jc reference to JVM SparkDataFrame column

Note

Column since 1.4.0

column since 1.6.0

See Also

Other non-aggregate functions: \texttt{column\_nonaggregate\_functions}, \texttt{not()}

Examples

## Not run: column("name")
Description

Aggregate functions defined for Column.

Usage

approxCountDistinct(x, ...)
collect_list(x)
collect_set(x)
countDistinct(x, ...)
grouping_bit(x)
grouping_id(x, ...)
kurtosis(x)
n_distinct(x, ...)
sd(x, na.rm = FALSE)
skewness(x)
stddev(x)
stddev_pop(x)
stddev_samp(x)
sumDistinct(x)
var(x, y = NULL, na.rm = FALSE, use)
variance(x)
var_pop(x)
var_samp(x)

## S4 method for signature 'Column'
approxCountDistinct(x, rsd = 0.05)

## S4 method for signature 'Column'
kurtosis(x)

## S4 method for signature 'Column'
max(x)

## S4 method for signature 'Column'
mean(x)

## S4 method for signature 'Column'
min(x)

## S4 method for signature 'Column'
sd(x)

## S4 method for signature 'Column'
skewness(x)

## S4 method for signature 'Column'
stddev(x)

## S4 method for signature 'Column'
stddev_pop(x)

## S4 method for signature 'Column'
stddev_samp(x)

## S4 method for signature 'Column'
sum(x)

## S4 method for signature 'Column'
sumDistinct(x)

## S4 method for signature 'Column'
var(x)

## S4 method for signature 'Column'
variance(x)

## S4 method for signature 'Column'
var_pop(x)

## S4 method for signature 'Column'
var_samp(x)

## S4 method for signature 'Column'
approxCountDistinct(x, rsd = 0.05)

## S4 method for signature 'Column'
countDistinct(x, ...)

## S4 method for signature 'Column'
n_distinct(x, ...)

## S4 method for signature 'Column'
collect_list(x)

## S4 method for signature 'Column'
collect_set(x)

## S4 method for signature 'Column'
grouping_bit(x)

## S4 method for signature 'Column'
grouping_id(x, ...)

Arguments

x            Column to compute on.
...           additional argument(s). For example, it could be used to pass additional Columns.
y, na.rm, use currently not used.
rsd           maximum estimation error allowed (default = 0.05).

Details

approxCountDistinct: Returns the approximate number of distinct items in a group.
kurtosis: Returns the kurtosis of the values in a group.
max: Returns the maximum value of the expression in a group.
mean: Returns the average of the values in a group. Alias for avg.
min: Returns the minimum value of the expression in a group.
mean: Returns the sum of all values in the expression.
stddev: Alias for std_dev.
stddev_pop: Returns the population standard deviation of the expression in a group.
stddev_samp: Returns the unbiased sample standard deviation of the expression in a group.
sd: Alias for stddev_samp.
skewness: Returns the skewness of the values in a group.
var: Alias for var_samp.
var_pop: Returns the population variance of the values in a group.
var_samp: Returns the unbiased variance of the values in a group.
column_aggregate_functions

countDistinct: Returns the number of distinct items in a group.
n_distinct: Returns the number of distinct items in a group.

collect_list: Creates a list of objects with duplicates. Note: the function is non-deterministic
because the order of collected results depends on order of rows which may be non-deterministic
after a shuffle.

collect_set: Creates a list of objects with duplicate elements eliminated. Note: the function is
non-deterministic because the order of collected results depends on order of rows which may be
non-deterministic after a shuffle.

grouping_bit: Indicates whether a specified column in a GROUP BY list is aggregated or not,
returns 1 for aggregated or 0 for not aggregated in the result set. Same as GROUPING in SQL and
grouping function in Scala.

grouping_id: Returns the level of grouping. Equals to grouping_bit(c1) * 2^(n-1) + grouping_bit(c2)
* 2^(n-2) + ... + grouping_bit(cn).

Note

approxCountDistinct(Column) since 1.4.0
kurtosis since 1.6.0
max since 1.5.0
mean since 1.5.0
min since 1.5.0
sd since 1.6.0
skewness since 1.6.0
stddev since 1.6.0
stddev_pop since 1.6.0
stddev_samp since 1.6.0
sum since 1.5.0
sumDistinct since 1.4.0
var since 1.6.0
variance since 1.6.0
var_pop since 1.5.0
var_samp since 1.6.0
approxCountDistinct(Column, numeric) since 1.4.0
countDistinct since 1.4.0
n_distinct since 1.4.0
collect_list since 2.3.0
collect_set since 2.3.0
grouping_bit since 2.3.0
grouping_id since 2.3.0
See Also

Other aggregate functions: `avg()`, `corr()`, `count()`, `cov()`, `first()`, `last()`

Examples

```r
## Not run:
# Dataframe used throughout this doc
df <- createDataFrame(cbind(model = rownames(mtcars), mtcars))
## End(Not run)

## Not run:
head(select(df, approxCountDistinct(df$gear)))
head(select(df, approxCountDistinct(df$gear, 0.02)))
head(select(df, countDistinct(df$gear, df$cyl)))
head(select(df, n_distinct(df$gear)))
head(distinct(select(df, "gear")))
## End(Not run)

## Not run:
head(select(df, mean(df$mpg), sd(df$mpg), skewness(df$mpg), kurtosis(df$mpg)))
## End(Not run)

## Not run:
head(select(df, avg(df$mpg), mean(df$mpg), sum(df$mpg), min(df$wt), max(df$qsec)))

# metrics by num of cylinders
tmp <- agg(groupBy(df, "cyl"), avg(df$mpg), avg(df$hp), avg(df$wt), avg(df$qsec))
head(orderBy(tmp, "cyl"))

# car with the max mpg
mpg_max <- as.numeric(collect(agg(df, max(df$mpg))))
head(where(df, df$mpg == mpg_max))
## End(Not run)

## Not run:
head(select(df, sd(df$mpg), stddev(df$mpg), stddev_pop(df$wt), stddev_samp(df$qsec)))
## End(Not run)

## Not run:
head(select(df, sumDistinct(df$gear)))
head(distinct(select(df, "gear")))
## End(Not run)

## Not run:
head(agg(df, var(df$mpg), variance(df$mpg), var_pop(df$mpg), var_samp(df$mpg)))
## End(Not run)

## Not run:
df2 = df[df$mpg > 20, ]
collect(select(df2, collect_list(df2$gear)))
collect(select(df2, collect_set(df2$gear)))
## End(Not run)
```
## Not run:
# With cube
agg(
    cube(df, "cyl", "gear", "am"),
    mean(df$mpg),
    grouping_bit(df$cyl), grouping_bit(df$gear), grouping_bit(df$am)
)

# With rollup
agg(
    rollup(df, "cyl", "gear", "am"),
    mean(df$mpg),
    grouping_bit(df$cyl), grouping_bit(df$gear), grouping_bit(df$am)
)
## End(Not run)

## Not run:
# With cube
agg(
    cube(df, "cyl", "gear", "am"),
    mean(df$mpg),
    grouping_id(df$cyl, df$gear, df$am)
)

# With rollup
agg(
    rollup(df, "cyl", "gear", "am"),
    mean(df$mpg),
    grouping_id(df$cyl, df$gear, df$am)
)
## End(Not run)

column_collection_functions

Collection functions for Column operations

**Description**

Collection functions defined for `Column`.

**Usage**

array_contains(x, value)

array_distinct(x)

array_except(x, y)
array_intersect(x, y)
array_join(x, delimiter, ...)
array_max(x)
array_min(x)
array_position(x, value)
array_remove(x, value)
array_repeat(x, count)
array_sort(x)
arrays_overlap(x, y)
array_union(x, y)
arrays_zip(x, ...)
concat(x, ...)
element_at(x, extraction)
explode(x)
explode_outer(x)
flatten(x)
from_json(x, schema, ...)
map_from_arrays(x, y)
map_keys(x)
map_values(x)
posexplode(x)
posexplode_outer(x)
reverse(x)
shuffle(x)
size(x)
slice(x, start, length)
sort_array(x, asc = TRUE)
to_json(x, ...)

## S4 method for signature 'Column'
reverse(x)

## S4 method for signature 'Column'
to_json(x, ...)

## S4 method for signature 'Column'
concat(x, ...)

## S4 method for signature 'Column,characterOrstructType'
from_json(x, schema, as.json.array = FALSE, ...)

## S4 method for signature 'Column'
array_contains(x, value)

## S4 method for signature 'Column'
array_distinct(x)

## S4 method for signature 'Column,Column'
array_except(x, y)

## S4 method for signature 'Column,Column'
array_intersect(x, y)

## S4 method for signature 'Column,character'
array_join(x, delimiter, nullReplacement = NULL)

## S4 method for signature 'Column'
array_max(x)

## S4 method for signature 'Column'
array_min(x)

## S4 method for signature 'Column'
array_position(x, value)

## S4 method for signature 'Column'
array_remove(x, value)

## S4 method for signature 'Column,numericsorColumn'
array_repeat(x, count)
## S4 method for signature 'Column'
array_sort(x)
## S4 method for signature 'Column'
arrays_overlap(x, y)
## S4 method for signature 'Column,Column'
array_union(x, y)
## S4 method for signature 'Column'
arrays_zip(x, ...)
## S4 method for signature 'Column'
shuffle(x)
## S4 method for signature 'Column'
flatten(x)
## S4 method for signature 'Column'
map_from_arrays(x, y)
## S4 method for signature 'Column,Column'
map_keys(x)
## S4 method for signature 'Column'
map_values(x)
## S4 method for signature 'Column'
element_at(x, extraction)
## S4 method for signature 'Column'
explode(x)
## S4 method for signature 'Column'
slice(x, start, length)
## S4 method for signature 'Column'
sort_array(x, asc = TRUE)
## S4 method for signature 'Column'
posexplode(x)
## S4 method for signature 'Column'
explode_outer(x)

## S4 method for signature 'Column'
posexplode_outer(x)

### Arguments

- **x**: Column to compute on. Note the difference in the following methods:
  - `to_json`: it is the column containing the struct, array of the structs, the map or array of maps.
  - `from_json`: it is the column containing the JSON string.
- **value**: A value to compute on.
  - `array_contains`: a value to be checked if contained in the column.
  - `array_position`: a value to locate in the given array.
  - `array_remove`: a value to remove in the given array.
- **y**: Column to compute on.
- **delimiter**: a character string that is used to concatenate the elements of column.
- **...**: additional argument(s). In `to_json` and `from_json`, this contains additional named properties to control how it is converted, accepts the same options as the JSON data source. In `arrays_zip`, this contains additional Columns of arrays to be merged.
- **count**: a Column or constant determining the number of repetitions.
- **extraction**: index to check for in array or key to check for in map
- **schema**: a structType object to use as the schema to use when parsing the JSON string. Since Spark 2.3, the DDL-formatted string is also supported for the schema.
- **start**: the starting index
- **length**: the length of the slice
- **asc**: a logical flag indicating the sorting order. TRUE, sorting is in ascending order. FALSE, sorting is in descending order.
- **as.json.array**: indicating if input string is JSON array of objects or a single object.
- **nullReplacement**: an optional character string that is used to replace the Null values.

### Details

- **reverse**: Returns a reversed string or an array with reverse order of elements.
- **to_json**: Converts a column containing a structType, a mapType or an arrayType into a Column of JSON string. Resolving the Column can fail if an unsupported type is encountered.
- **concat**: Concatenates multiple input columns together into a single column. The function works with strings, binary and compatible array columns.
- **from_json**: Parses a column containing a JSON string into a Column of structType with the specified schema or array of structType if `as.json.array` is set to TRUE. If the string is unparsable, the Column will contain the value NA.
array_contains: Returns null if the array is null, true if the array contains the value, and false otherwise.
array_distinct: Removes duplicate values from the array.
array_except: Returns an array of the elements in the first array but not in the second array, without duplicates. The order of elements in the result is not determined.
array_intersect: Returns an array of the elements in the intersection of the given two arrays, without duplicates.
array_join: Concatenates the elements of column using the delimiter. Null values are replaced with nullReplacement if set, otherwise they are ignored.
array_max: Returns the maximum value of the array.
array_min: Returns the minimum value of the array.
array_position: Locates the position of the first occurrence of the given value in the given array. Returns NA if either of the arguments are NA. Note: The position is not zero based, but 1 based index. Returns 0 if the given value could not be found in the array.
array_remove: Removes all elements that equal to element from the given array.
array_repeat: Creates an array containing x repeated the number of times given by count.
array_sort: Sorts the input array in ascending order. The elements of the input array must be orderable. NA elements will be placed at the end of the returned array.
arrays_overlap: Returns true if the input arrays have at least one non-null element in common. If not and both arrays are non-empty and any of them contains a null, it returns null. It returns false otherwise.
array_union: Returns an array of the elements in the union of the given two arrays, without duplicates.
arrays_zip: Returns a merged array of structs in which the N-th struct contains all N-th values of input arrays.
shuffle: Returns a random permutation of the given array.
flatten: Creates a single array from an array of arrays. If a structure of nested arrays is deeper than two levels, only one level of nesting is removed.
map_from_arrays: Creates a new map column. The array in the first column is used for keys. The array in the second column is used for values. All elements in the array for key should not be null.
map_keys: Returns an unordered array containing the keys of the map.
map_values: Returns an unordered array containing the values of the map.
element_at: Returns element of array at given index in extraction if x is array. Returns value for the given key in extraction if x is map. Note: The position is not zero based, but 1 based index.
explode: Creates a new row for each element in the given array or map column.
slice: Returns a length of array or map.
slice: Returns an array containing all the elements in x from the index start (array indices start at 1, or from the end if start is negative) with the specified length.
sort_array: Sorts the input array in ascending or descending order according to the natural ordering of the array elements. NA elements will be placed at the beginning of the returned array in ascending order or at the end of the returned array in descending order.
**column_collection_functions**

poseplode: Creates a new row for each element with position in the given array or map column.
extplode: Creates a new row for each element in the given array or map column. Unlike explode, if the array/map is null or empty then null is produced.
poseexplode_outer: Creates a new row for each element with position in the given array or map column. Unlike poseplode, if the array/map is null or empty then the row (null, null) is produced.

Note

- reverse since 1.5.0
- to_json since 2.2.0
- concat since 1.5.0
- from_json since 2.2.0
- array_contains since 1.6.0
- array_distinct since 2.4.0
- array_except since 2.4.0
- array_intersect since 2.4.0
- array_join since 2.4.0
- array_max since 2.4.0
- array_min since 2.4.0
- array_position since 2.4.0
- array_remove since 2.4.0
- array_repeat since 2.4.0
- array_sort since 2.4.0
- arrays_overlap since 2.4.0
- array_union since 2.4.0
- arrays_zip since 2.4.0
- shuffle since 2.4.0
- flatten since 2.4.0
- map_from_arrays since 2.4.0
- map_keys since 2.3.0
- map_values since 2.3.0
- element_at since 2.4.0
- explode since 1.5.0
- size since 1.5.0
- slice since 2.4.0
- sort_array since 1.6.0
- poseplode since 2.1.0
- explode_outer since 2.3.0
- poseexplode_outer since 2.3.0
Examples

## Not run:

```r
data_set <- createDataFrame(cbind(model = rownames(mtcars), mtcars))
tmp <- mutate(data_set, v1 = create_array(data_set$mpg, data_set$cyl, data_set$hp))
head(select(tmp, array_contains(tmp$v1, 21), size(tmp$v1), shuffle(tmp$v1)))
head(select(tmp, array_max(tmp$v1), array_min(tmp$v1), array_distinct(tmp$v1)))
head(select(tmp, array_position(tmp$v1, 21), array_repeat(data_set$mpg, 3), array_sort(tmp$v1)))
head(select(tmp, flatten(tmp$v1), reverse(tmp$v1), array_remove(tmp$v1, 21)))
tmp2 <- mutate(tmp, v2 = explode(tmp$v1))
head(tmp2)
head(select(tmp, posexplode(tmp$v1)))
head(select(tmp, slice(tmp$v1, 2L, 2L)))
head(select(tmp, sort_array(tmp$v1)))
head(select(tmp, sort_array(tmp$v1, asc = FALSE)))
tmp3 <- mutate(data_set, v3 = create_map(data_set$model, data_set$cyl))
head(select(tmp3, map_keys(tmp3$v3), map_values(tmp3$v3)))
head(select(tmp3, element_at(tmp3$v3, "Valiant")))
tmp4 <- mutate(data_set, v4 = create_array(data_set$mpg, data_set$cyl), v5 = create_array(data_set$cyl, data_set$hp))
head(select(tmp4, concat(tmp4$v4, tmp4$v5), arrays_overlap(tmp4$v4, tmp4$v5)))
head(select(tmp4, array_except(tmp4$v4, tmp4$v5), array_intersect(tmp4$v4, tmp4$v5)))
head(select(tmp4, arrays_zip(tmp4$v4, tmp4$v5)))
head(select(tmp, concat(data_set$mpg, data_set$cyl, data_set$hp)))
tmp5 <- mutate(data_set, v6 = create_array(data_set$model, data_set$model))
head(select(tmp5, array_join(tmp5$v6, ","), array_join(tmp5$v6, ",", "NULL")))
## End(Not run)
```

## Not run:

```r
data_set <- sql("SELECT named_struct('date', cast('2000-01-01' as date)) as d")
select(data_set, to_json(data_set$d, date_format = 'dd/MM/yyyy'))
```

## Not run:

```r
data_set <- sql("SELECT array(named_struct('name', 'Bob'), named_struct('name', 'Alice')) as people")
data_set <- mutate(data_set, people_json = to_json(data_set$people))
```

## Not run:

```r
data_set <- sql("SELECT named_struct('name', 'Bob') as people")
data_set <- mutate(data_set, people_json = to_json(data_set$people))
```

## Not run:

```r
data_set <- sql("SELECT named_struct('date', cast('2000-01-01' as date)) as d")
schema <- structType(structField("date", "string"))
head(select(data_set, from_json(data_set$d, schema, date_format = 'dd/MM/yyyy')))```
```
df2 <- mutate(df2, people_json = to_json(df2$people))
schema <- structType(structField("name", "string"))
head(select(df2, from_json(df2$people_json, schema)))
head(select(df2, from_json(df2$people_json, "name STRING")))
## Not run:
## Not run:
df2 <- createDataFrame(data.frame(
id = c(1, 2, 3), text = c("a,b,c", NA, "d,e")
))
head(select(df2, df2$id, explode_outer(split_string(df2$text, ","))))
head(select(df2, df2$id, posexplode_outer(split_string(df2$text, ","))))
## End(Not run)
```

column_datetime_diff_functions

Date time arithmetic functions for Column operations

Description

Date time arithmetic functions defined for Column.

Usage

- `add_months(y, x)`
- `datediff(y, x)`
- `date_add(y, x)`
- `date_format(y, x)`
- `date_sub(y, x)`
- `from_utc_timestamp(y, x)`
- `months_between(y, x)`
- `next_day(y, x)`
- `to_utc_timestamp(y, x)`

## S4 method for signature 'Column'

- `datediff(y, x)`
- `months_between(y, x)`

## S4 method for signature 'Column'
## S4 method for signature 'Column, character'
date_format(y, x)

## S4 method for signature 'Column, character'
from_utc_timestamp(y, x)

## S4 method for signature 'Column, character'
next_day(y, x)

## S4 method for signature 'Column, character'
to_utc_timestamp(y, x)

## S4 method for signature 'Column, numeric'
add_months(y, x)

## S4 method for signature 'Column, numeric'
date_add(y, x)

## S4 method for signature 'Column, numeric'
date_sub(y, x)

### Arguments

**y**  
Column to compute on.

**x**  
For class Column, it is the column used to perform arithmetic operations with column y. For class numeric, it is the number of months or days to be added to or subtracted from y. For class character, it is

- date_format: date format specification.
- from_utc_timestamp, to_utc_timestamp: time zone to use.
- next_day: day of the week string.

### Details

datediff: Returns the number of days from y to x. If y is later than x then the result is positive.

months_between: Returns number of months between dates y and x. If y is later than x, then the result is positive. If y and x are on the same day of month, or both are the last day of month, time of day will be ignored. Otherwise, the difference is calculated based on 31 days per month, and rounded to 8 digits.

date_format: Converts a date/timestamp/string to a value of string in the format specified by the date format given by the second argument. A pattern could be for instance dd.MM.yyyy and could return a string like '18.03.1993'. All pattern letters of java.text.SimpleDateFormat can be used. Note: Use when ever possible specialized functions like year. These benefit from a specialized implementation.

from_utc_timestamp: This is a common function for databases supporting TIMESTAMP WITHOUT TIMEZONE. This function takes a timestamp which is timezone-agnostic, and interprets it
as a timestamp in UTC, and renders that timestamp as a timestamp in the given time zone. However, timestamp in Spark represents number of microseconds from the Unix epoch, which is not timezone-agnostic. So in Spark this function just shift the timestamp value from UTC timezone to the given timezone. This function may return confusing result if the input is a string with timezone, e.g. (2018-03-13T06:18:23+00:00). The reason is that, Spark firstly cast the string to timestamp according to the timezone in the string, and finally display the result by converting the timestamp to string according to the session local timezone.

next_day: Given a date column, returns the first date which is later than the value of the date column that is on the specified day of the week. For example, next_day("2015-07-27","Sunday") returns 2015-08-02 because that is the first Sunday after 2015-07-27. Day of the week parameter is case insensitive, and accepts first three or two characters: "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun".

to_utc_timestamp: This is a common function for databases supporting TIMESTAMP WITHOUT TIMEZONE. This function takes a timestamp which is timezone-agnostic, and interprets it as a timestamp in the given timezone, and renders that timestamp as a timestamp in UTC. However, timestamp in Spark represents number of microseconds from the Unix epoch, which is not timezone-agnostic. So in Spark this function just shift the timestamp value from the given timezone to UTC timezone. This function may return confusing result if the input is a string with timezone, e.g. (2018-03-13T06:18:23+00:00). The reason is that, Spark firstly cast the string to timestamp according to the timezone in the string, and finally display the result by converting the timestamp to string according to the session local timezone.

add_months: Returns the date that is numMonths (x) after startDate (y).
date_add: Returns the date that is x days after.
date_sub: Returns the date that is x days before.

Note

datediff since 1.5.0
months_between since 1.5.0
date_format since 1.5.0
from_utc_timestamp since 1.5.0
next_day since 1.5.0
to_utc_timestamp since 1.5.0
add_months since 1.5.0
date_add since 1.5.0
date_sub since 1.5.0

See Also

Other data time functions: `column_datetime_functions`

Examples

```r
## Not run:
dts <- c("2005-01-02 18:47:22",
```
"2005-12-24 16:30:58",
"2005-10-28 07:30:05",
"2005-12-28 07:01:05",
"2006-01-24 00:01:10")
y <- c(2.0, 2.2, 3.4, 2.5, 1.8)
df <- createDataFrame(data.frame(time = as.POSIXct(dts), y = y))
## End(Not run)

## Not run:
tmp <- createDataFrame(data.frame(time_string1 = as.POSIXct(dts),
                              time_string2 = as.POSIXct(dts[order(runif(length(dts)))])))
tmp2 <- mutate(tmp, datediff = datediff(tmp$time_string1, tmp$time_string2),
               monthdiff = months_between(tmp$time_string1, tmp$time_string2))
head(tmp2)
## End(Not run)

## Not run:
tmp <- mutate(df, from_utc = from_utc_timestamp(df$time, "PST"),
               to_utc = to_utc_timestamp(df$time, "PST"))
head(tmp)
## End(Not run)

## Not run:
tmp <- mutate(df, t1 = add_months(df$time, 1),
               t2 = date_add(df$time, 2),
               t3 = date_sub(df$time, 3),
               t4 = next_day(df$time, "Sun"))
head(tmp)
## End(Not run)

column_datetime_functions

Date time functions for Column operations

Description
Date time functions defined for Column.

Usage

current_date(x = "missing")

current_timestamp(x = "missing")

date_trunc(format, x)

dayofmonth(x)

dayofweek(x)
dayofyear(x)

from_unixtime(x, ...)

hour(x)

last_day(x)

minute(x)

month(x)

quarter(x)

second(x)

to_date(x, format)

to_timestamp(x, format)

unix_timestamp(x, format)

weekofyear(x)

window(x, ...)

year(x)

## S4 method for signature 'Column'
dayofmonth(x)

## S4 method for signature 'Column'
dayofweek(x)

## S4 method for signature 'Column'
dayofyear(x)

## S4 method for signature 'Column'
hour(x)

## S4 method for signature 'Column'
last_day(x)

## S4 method for signature 'Column'
minute(x)

## S4 method for signature 'Column'
month(x)
## S4 method for signature 'Column'
quarter(x)
## S4 method for signature 'Column'
second(x)
## S4 method for signature 'Column'
to_date(x, format)
## S4 method for signature 'Column'
to_date(x, format)
## S4 method for signature 'Column'
to_timestamp(x, format)
## S4 method for signature 'Column'
to_timestamp(x, format)
## S4 method for signature 'Column'
weekofyear(x)
## S4 method for signature 'Column'
year(x)
## S4 method for signature 'Column'
from_unixtime(x, format = "yyyy-MM-dd HH:mm:ss")
## S4 method for signature 'Column'
window(x, windowDuration, slideDuration = NULL, startTime = NULL)
## S4 method for signature 'missing'
unix_timestamp(x, format)
## S4 method for signature 'missing'
unix_timestamp(x, format)
## S4 method for signature 'Column'
unix_timestamp(x, format = "yyyy-MM-dd HH:mm:ss")
## S4 method for signature 'Column'
trunc(x, format)
## S4 method for signature 'character'
date_trunc(format, x)
## S4 method for signature 'missing'
current_date()

## S4 method for signature 'missing'
current_timestamp()

### Arguments

- **x**
  - Column to compute on. In window, it must be a time Column of TimestampType.
  - This is not used with `current_date` and `current_timestamp`.

- **format**
  - The format for the given dates or timestamps in Column x.
  - See the format used in the following methods:
    - `to_date` and `to_timestamp`: it is the string to use to parse Column x to DateType or TimestampType.
    - `trunc`: it is the string to use to specify the truncation method. For example, "year", "yyy", "yy" for truncate by year, or "month", "mon", "mm" for truncate by month.
    - `date_trunc`: it is similar with trunc’s but additionally supports "day", "dd", "second", "minute", "hour", "week" and "quarter".

- **windowDuration**
  - A string specifying the width of the window, e.g. '1 second', '1 day 12 hours', '2 minutes'. Valid interval strings are 'week', 'day', 'hour', 'minute', 'second', 'millisecond', 'microsecond'. Note that the duration is a fixed length of time, and does not vary over time according to a calendar. For example, '1 day' always means 86,400,000 milliseconds, not a calendar day.

- **slideDuration**
  - A string specifying the sliding interval of the window. Same format as `windowDuration`.
  - A new window will be generated every `slideDuration`. Must be less than or equal to the `windowDuration`. This duration is likewise absolute, and does not vary according to a calendar.

- **startTime**
  - the offset with respect to 1970-01-01 00:00:00 UTC with which to start window intervals. For example, in order to have hourly tumbling windows that start 15 minutes past the hour, e.g. 12:15-13:15, 13:15-14:15... provide `startTime` as "15 minutes".

### Details

- **dayofmonth**: Extracts the day of the month as an integer from a given date/timestamp/string.
- **dayofweek**: Extracts the day of the week as an integer from a given date/timestamp/string.
- **dayofyear**: Extracts the day of the year as an integer from a given date/timestamp/string.
- **hour**: Extracts the hour as an integer from a given date/timestamp/string.
- **last_day**: Given a date column, returns the last day of the month which the given date belongs to.
  - For example, input "2015-07-27" returns "2015-07-31" since July 31 is the last day of the month in July 2015.
- **minute**: Extracts the minute as an integer from a given date/timestamp/string.
- **month**: Extracts the month as an integer from a given date/timestamp/string.
- **quarter**: Extracts the quarter as an integer from a given date/timestamp/string.
second: Extracts the second as an integer from a given date/timestamp/string.

to_date: Converts the column into a DateType. You may optionally specify a format according to the rules in: http://docs.oracle.com/javase/tutorial/i18n/format/simpleDateFormat.html. If the string cannot be parsed according to the specified format (or default), the value of the column will be null. By default, it follows casting rules to a DateType if the format is omitted (equivalent to cast(df$x,"date").

to_timestamp: Converts the column into a TimestampType. You may optionally specify a format according to the rules in: http://docs.oracle.com/javase/tutorial/i18n/format/simpleDateFormat.html. If the string cannot be parsed according to the specified format (or default), the value of the column will be null. By default, it follows casting rules to a TimestampType if the format is omitted (equivalent to cast(df$x,"timestamp").

weekofyear: Extracts the week number as an integer from a given date/timestamp/string.

year: Extracts the year as an integer from a given date/timestamp/string.

from_unixtime: Converts the number of seconds from unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the JVM in the given format. See Customizing Formats for available options.

window: Bucketizes rows into one or more time windows given a timestamp specifying column. Window starts are inclusive but the window ends are exclusive, e.g. 12:05 will be in the window [12:05,12:10) but not in [12:00,12:05). Windows can support microsecond precision. Windows in the order of months are not supported. It returns an output column of struct called 'window' by default with the nested columns 'start' and 'end'

unix_timestamp: Gets current Unix timestamp in seconds.

trunc: Returns date truncated to the unit specified by the format.

date_trunc: Returns timestamp truncated to the unit specified by the format.

current_date: Returns the current date as a date column.

current_timestamp: Returns the current timestamp as a timestamp column.

Note

dayofmonth since 1.5.0

dayofweek since 2.3.0

dayofyear since 1.5.0

hour since 1.5.0

last_day since 1.5.0

minute since 1.5.0

month since 1.5.0

quarter since 1.5.0

second since 1.5.0

to_date(Column) since 1.5.0

to_date(Column, character) since 2.2.0

to_timestamp(Column) since 2.2.0
to_timestamp(Column, character) since 2.2.0
weekofyear since 1.5.0
year since 1.5.0
from_unixtime since 1.5.0
window since 2.0.0
unix_timestamp since 1.5.0
unix_timestamp(Column) since 1.5.0
unix_timestamp(Column, character) since 1.5.0
trun since 2.3.0
date_trun since 2.3.0
current_date since 2.3.0
current_timestamp since 2.3.0

See Also

Other data time functions: column_datetime_diff_functions

Examples

```r
## Not run:
dts <- c("2005-01-02 18:47:22",
    "2005-12-24 16:30:58",
    "2005-10-28 07:30:05",
    "2005-12-28 07:01:05",
    "2006-01-24 00:01:10")
y <- c(2.0, 2.2, 3.4, 2.5, 1.8)
df <- createDataFrame(data.frame(time = as.POSIXct(dts), y = y))
## End(Not run)

## Not run:
head(select(df, df$time, year(df$time), quarter(df$time), month(df$time),
    dayofmonth(df$time), dayofweek(df$time), dayofyear(df$time), weekofyear(df$time))
head(agg(groupBy(df, year(df$time)), count(df$y), avg(df$y)))
head(agg(groupBy(df, month(df$time)), avg(df$y)))
## End(Not run)

## Not run:
head(select(df, hour(df$time), minute(df$time), second(df$time)))
head(agg(groupBy(df, dayofmonth(df$time)), avg(df$y)))
head(agg(groupBy(df, hour(df$time)), avg(df$y)))
head(agg(groupBy(df, minute(df$time)), avg(df$y)))
## End(Not run)

## Not run:
head(select(df, df$time, last_day(df$time), month(df$time)))
## End(Not run)

## Not run:
```
tmp <- createDataFrame(data.frame(time_string = dts))
tmp2 <- mutate(tmp, date1 = to_date(tmp$time_string),
    date2 = to_date(tmp$time_string, "yyyy-MM-dd"),
    date3 = date_format(tmp$time_string, "MM/dd/yyy"),
    time1 = to_timestamp(tmp$time_string),
    time2 = to_timestamp(tmp$time_string, "yyyy-MM-dd"))
head(tmp2)
## End(Not run)

## Not run:
tmp <- mutate(df, to_unix = unix_timestamp(df$time),
    to_unix2 = unix_timestamp(df$time, 'yyyy-MM-dd HH'),
    from_unix = from_unixtime(unix_timestamp(df$time)),
    from_unix2 = from_unixtime(unix_timestamp(df$time), 'yyyy-MM-dd HH:mm'))
head(tmp)
## End(Not run)

## Not run:
# One minute windows every 15 seconds 10 seconds after the minute, e.g. 09:00:10-09:01:10,
# 09:00:25-09:01:25, 09:00:40-09:01:40, ...
window(df$time, "1 minute", "15 seconds", "10 seconds")

# One minute tumbling windows 15 seconds after the minute, e.g. 09:00:15-09:01:15,
# 09:01:15-09:02:15...
window(df$time, "1 minute", startTime = "15 seconds")

# Thirty-second windows every 10 seconds, e.g. 09:00:00-09:00:30, 09:00:10-09:00:40, ...
window(df$time, "30 seconds", "10 seconds")
## End(Not run)

## Not run:
head(select(df, df$time, trunc(df$time, "year"), trunc(df$time, "yy"),
    trunc(df$time, "month"), trunc(df$time, "mon")))
## End(Not run)

## Not run:
head(select(df, df$time, date_trunc("hour", df$time), date_trunc("minute", df$time),
    date_trunc("week", df$time), date_trunc("quarter", df$time)))
## End(Not run)

## Not run:
head(select(df, current_date(), current_timestamp()))
## End(Not run)

---

**column_math_functions**

Math functions for Column operations

**Description**

Math functions defined for Column.
**Usage**

- `bin(x)`
- `bround(x, ...)`
- `cbrt(x)`
- `ceil(x)`
- `conv(x, fromBase, toBase)`
- `hex(x)`
- `hypot(y, x)`
- `pmod(y, x)`
- `rint(x)`
- `shiftLeft(y, x)`
- `shiftRight(y, x)`
- `shiftRightUnsigned(y, x)`
- `signum(x)`
- `toDegrees(x)`
- `toRadians(x)`
- `unhex(x)`

## S4 method for signature 'Column'

- `abs(x)`
- `acos(x)`
- `asin(x)`
- `atan(x)`
- `bin(x)`
## S4 method for signature 'Column'
cbrt(x)

## S4 method for signature 'Column'
ceil(x)

## S4 method for signature 'Column'
ceiling(x)

## S4 method for signature 'Column'
cos(x)

## S4 method for signature 'Column'
cosh(x)

## S4 method for signature 'Column'
exp(x)

## S4 method for signature 'Column'
expm1(x)

## S4 method for signature 'Column'
factorial(x)

## S4 method for signature 'Column'
floor(x)

## S4 method for signature 'Column'
hex(x)

## S4 method for signature 'Column'
log(x)

## S4 method for signature 'Column'
log10(x)

## S4 method for signature 'Column'
log1p(x)

## S4 method for signature 'Column'
log2(x)

## S4 method for signature 'Column'
rint(x)

## S4 method for signature 'Column'
round(x)
## S4 method for signature 'Column'
\texttt{bround(x, scale = 0)}

## S4 method for signature 'Column'
\texttt{signum(x)}

## S4 method for signature 'Column'
\texttt{sign(x)}

## S4 method for signature 'Column'
\texttt{sin(x)}

## S4 method for signature 'Column'
\texttt{sinh(x)}

## S4 method for signature 'Column'
\texttt{sqrt(x)}

## S4 method for signature 'Column'
\texttt{tan(x)}

## S4 method for signature 'Column'
\texttt{tanh(x)}

## S4 method for signature 'Column'
\texttt{toDegrees(x)}

## S4 method for signature 'Column'
\texttt{toRadians(x)}

## S4 method for signature 'Column'
\texttt{unhex(x)}

## S4 method for signature 'Column'
\texttt{atan2(y, x)}

## S4 method for signature 'Column'
\texttt{hypot(y, x)}

## S4 method for signature 'Column'
\texttt{pmod(y, x)}

## S4 method for signature 'Column,numeric'
\texttt{shiftLeft(y, x)}

## S4 method for signature 'Column,numeric'
\texttt{shiftRight(y, x)}
## S4 method for signature 'Column, numeric'
shiftRightUnsigned(y, x)

## S4 method for signature 'Column, numeric, numeric'
conv(x, fromBase, toBase)

### Arguments

- **x**: Column to compute on. In `shiftLeft`, `shiftRight` and `shiftRightUnsigned`, this is the number of bits to shift.
- **...**: additional argument(s).
- **fromBase**: base to convert from.
- **toBase**: base to convert to.
- **y**: Column to compute on.
- **scale**: round to `scale` digits to the right of the decimal point when `scale > 0`, the nearest even number when `scale = 0`, and `scale` digits to the left of the decimal point when `scale < 0`.

### Details

- **abs**: Computes the absolute value.
- **acos**: Returns the inverse cosine of the given value, as if computed by `java.lang.Math.acos()`.
- **asin**: Returns the inverse sine of the given value, as if computed by `java.lang.Math.asin()`.
- **atan**: Returns the inverse tangent of the given value, as if computed by `java.lang.Math.atan()`.
- **bin**: Returns the string representation of the binary value of the given long column. For example, `bin("12")` returns "1100".
- **cbrt**: Computes the cube-root of the given value.
- **ceil**: Computes the ceiling of the given value.
- **ceiling**: Alias for `ceil`.
- **cos**: Returns the cosine of the given value, as if computed by `java.lang.Math.cos()`. Units in radians.
- **cosh**: Returns the hyperbolic cosine of the given value, as if computed by `java.lang.Math.cosh()`.
- **exp**: Computes the exponential of the given value.
- **expm1**: Computes the exponential of the given value minus one.
- **factorial**: Computes the factorial of the given value.
- **floor**: Computes the floor of the given value.
- **hex**: Computes hex value of the given column.
- **log**: Computes the natural logarithm of the given value.
- **log10**: Computes the logarithm of the given value in base 10.
- **log1p**: Computes the natural logarithm of the given value plus one.
- **log2**: Computes the logarithm of the given column in base 2.
rint: Returns the double value that is closest in value to the argument and is equal to a mathematical integer.

round: Returns the value of the column rounded to 0 decimal places using HALF_UP rounding mode.

bround: Returns the value of the column e rounded to scale decimal places using HALF_EVEN rounding mode if scale >= 0 or at integer part when scale < 0. Also known as Gaussian rounding or bankers’ rounding that rounds to the nearest even number. bround(2.5, 0) = 2, bround(3.5, 0) = 4.

signum: Computes the signum of the given value.

sign: Alias for signum.

sin: Returns the sine of the given value, as if computed by java.lang.Math.sin(). Units in radians.

sinh: Returns the hyperbolic sine of the given value, as if computed by java.lang.Math.sinh().
sqrt: Computes the square root of the specified float value.

tan: Returns the tangent of the given value, as if computed by java.lang.Math.tan(). Units in radians.

tanh: Returns the hyperbolic tangent of the given value, as if computed by java.lang.Math.tanh().
toDegrees: Converts an angle measured in radians to an approximately equivalent angle measured in degrees.
toRadians: Converts an angle measured in degrees to an approximately equivalent angle measured in radians.

unhex: Inverse of hex. Interprets each pair of characters as a hexadecimal number and converts to the byte representation of number.

atan2: Returns the angle theta from the conversion of rectangular coordinates (x, y) to polar coordinates (r, theta), as if computed by java.lang.Math.atan2(). Units in radians.
hypot: Computes "sqrt(a^2 + b^2)" without intermediate overflow or underflow.
pmod: Returns the positive value of dividend mod divisor. Column x is divisor column, and column y is the dividend column.

shiftLeft: Shifts the given value numBits left. If the given value is a long value, this function will return a long value else it will return an integer value.

shiftRight: (Signed) shifts the given value numBits right. If the given value is a long value, it will return a long value else it will return an integer value.

shiftRightUnsigned: (Unsigned) shifts the given value numBits right. If the given value is a long value, it will return a long value else it will return an integer value.

conv: Converts a number in a string column from one base to another.

Note

abs since 1.5.0
acos since 1.5.0
asin since 1.5.0
atan since 1.5.0
bin since 1.5.0
cbrt since 1.4.0
ceil since 1.5.0
ceiling since 1.5.0
cos since 1.5.0
cosh since 1.5.0
exp since 1.5.0
expm1 since 1.5.0
factorial since 1.5.0
floor since 1.5.0
hex since 1.5.0
log since 1.5.0
log10 since 1.5.0
log1p since 1.5.0
log2 since 1.5.0
rint since 1.5.0
round since 1.5.0
bround since 2.0.0
signum since 1.5.0
sign since 1.5.0
sin since 1.5.0
sinh since 1.5.0
sqrt since 1.5.0
tan since 1.5.0
tanh since 1.5.0
toDegrees since 1.4.0
toRadians since 1.4.0
unhex since 1.5.0
atan2 since 1.5.0
hypot since 1.4.0
pmod since 1.5.0
shiftLeft since 1.5.0
shiftRight since 1.5.0
shiftRightUnsigned since 1.5.0
conv since 1.5.0
Examples

```r
## Not run:
# Dataframe used throughout this doc
df <- createDataFrame(cbind(model = rownames(mtcars), mtcars))
tmp <- mutate(df, v1 = log(df$mpg), v2 = cbrt(df$disp),
              v3 = bround(df$wt, 1), v4 = bin(df$cyl),
              v5 = hex(df$wt), v6 = toDegrees(df$gear),
              v7 = atan2(df$cyl, df$am), v8 = hypot(df$cyl, df$am),
              v9 = pmod(df$hp, df$cyl), v10 = shiftLeft(df$disp, 1),
              v11 = conv(df$hp, 10, 16), v12 = sign(df$vs - 0.5),
              v13 = sqrt(df$disp), v14 = ceil(df$wt))
head(tmp)
## End(Not run)
```

---

**column_misc_functions**  
*Miscellaneous functions for Column operations*

**Description**

Miscellaneous functions defined for Column.

**Usage**

crc32(x)

hash(x, ...)

md5(x)

sha1(x)

sha2(y, x)

## S4 method for signature 'Column'
crc32(x)

## S4 method for signature 'Column'
hash(x, ...)

## S4 method for signature 'Column'
md5(x)

## S4 method for signature 'Column'
sha1(x)

## S4 method for signature 'Column,numeric'
sha2(y, x)
Arguments

- **x**: Column to compute on. In sha2, it is one of 224, 256, 384, or 512.
- **y**: Column to compute on.
- Additional Columns.

Details

crc32: Calculates the cyclic redundancy check value (CRC32) of a binary column and returns the value as a bigint.

hash: Calculates the hash code of given columns, and returns the result as an int column.

md5: Calculates the MD5 digest of a binary column and returns the value as a 32 character hex string.

sha1: Calculates the SHA-1 digest of a binary column and returns the value as a 40 character hex string.

sha2: Calculates the SHA-2 family of hash functions of a binary column and returns the value as a hex string. The second argument \( x \) specifies the number of bits, and is one of 224, 256, 384, or 512.

Note

crc32 since 1.5.0
hash since 2.0.0
md5 since 1.5.0
sha1 since 1.5.0
sha2 since 1.5.0

Examples

```r
## Not run:
# Dataframe used throughout this doc
df <- createDataFrame(cbind(model = rownames(mtcars), mtcars)[, 1:2])
tmp <- mutate(df, v1 = crc32(df$model), v2 = hash(df$model),
               v3 = hash(df$model, df$mpg), v4 = md5(df$model),
               v5 = sha1(df$model), v6 = sha2(df$model, 256))
head(tmp)
## End(Not run)
```

Description

Non-aggregate functions defined for Column.
Usage

    when(condition, value)
    bitwiseNOT(x)
    create_array(x, ...)
    create_map(x, ...)
    expr(x)
    greatest(x, ...)
    input_file_name(x = "missing")
    isnan(x)
    least(x, ...)
    lit(x)
    monotonically_increasing_id(x = "missing")
    nanvl(y, x)
    negate(x)
    rand(seed)
    randn(seed)
    spark_partition_id(x = "missing")
    struct(x, ...)

## S4 method for signature 'ANY'
    lit(x)

## S4 method for signature 'Column'
    bitwiseNOT(x)

## S4 method for signature 'Column'
    coalesce(x, ...)

## S4 method for signature 'Column'
    isnan(x)

## S4 method for signature 'Column'
is.na(x)

## S4 method for signature 'missing'
monotonically_increasing_id()

## S4 method for signature 'Column'

negate(x)

## S4 method for signature 'missing'
spark_partition_id()

## S4 method for signature 'characterOrColumn'
struct(x, ...)

## S4 method for signature 'Column'
nanvl(y, x)

## S4 method for signature 'Column'
greatest(x, ...)

## S4 method for signature 'Column'
least(x, ...)

## S4 method for signature 'character'
expr(x)

## S4 method for signature 'missing'
rand(seed)

## S4 method for signature 'numeric'
rand(seed)

## S4 method for signature 'missing'
randn(seed)

## S4 method for signature 'numeric'
randn(seed)

## S4 method for signature 'Column'
when(condition, value)

## S4 method for signature 'Column'
ifelse(test, yes, no)

## S4 method for signature 'Column'
create_array(x, ...)

## S4 method for signature 'Column'
create_map(x, ...)

## S4 method for signature 'missing'
input_file_name()

**Arguments**

- **condition**: the condition to test on. Must be a Column expression.
- **value**: result expression.
- **x**: Column to compute on. In `lit`, it is a literal value or a Column. In `expr`, it contains an expression character object to be parsed.
- **...**: additional Columns.
- **y**: Column to compute on.
- **seed**: a random seed. Can be missing.
- **test**: a Column expression that describes the condition.
- **yes**: return values for `TRUE` elements of test.
- **no**: return values for `FALSE` elements of test.

**Details**

- **lit**: A new Column is created to represent the literal value. If the parameter is a Column, it is returned unchanged.
- **bitwiseNOT**: Computes bitwise NOT.
- **coalesce**: Returns the first column that is not NA, or NA if all inputs are.
- **isnan**: Returns true if the column is NaN.
- **is.nan**: Alias for isnan.
- **monotonically_increasing_id**: Returns a column that generates monotonically increasing 64-bit integers. The generated ID is guaranteed to be monotonically increasing and unique, but not consecutive. The current implementation puts the partition ID in the upper 31 bits, and the record number within each partition in the lower 33 bits. The assumption is that the SparkDataFrame has less than 1 billion partitions, and each partition has less than 8 billion records. As an example, consider a SparkDataFrame with two partitions, each with 3 records. This expression would return the following IDs: 0, 1, 2, 8589934592 (1L « 33), 8589934593, 8589934594. This is equivalent to the MONOTONICALLY_INCREASING_ID function in SQL. The method should be used with no argument. Note: the function is non-deterministic because its result depends on partition IDs.
- **negate**: Unary minus, i.e. negate the expression.
- **spark_partition_id**: Returns the partition ID as a SparkDataFrame column. Note that this is nondeterministic because it depends on data partitioning and task scheduling. This is equivalent to the SPARK_PARTITION_ID function in SQL.
- **struct**: Creates a new struct column that composes multiple input columns.
- **nanvl**: Returns the first column (y) if it is not NaN, or the second column (x) if the first column is NaN. Both inputs should be floating point columns (DoubleType or FloatType).
- **greatest**: Returns the greatest value of the list of column names, skipping null values. This function takes at least 2 parameters. It will return null if all parameters are null.
least: Returns the least value of the list of column names, skipping null values. This function takes at least 2 parameters. It will return null if all parameters are null.

expr: Parses the expression string into the column that it represents, similar to SparkDataFrame.selectExpr

rand: Generates a random column with independent and identically distributed (i.i.d.) samples uniformly distributed in [0.0, 1.0). Note: the function is non-deterministic in general case.

randn: Generates a column with independent and identically distributed (i.i.d.) samples from the standard normal distribution. Note: the function is non-deterministic in general case.

when: Evaluates a list of conditions and returns one of multiple possible result expressions. For unmatched expressions null is returned.

ifelse: Evaluates a list of conditions and returns yes if the conditions are satisfied. Otherwise no is returned for unmatched conditions.

create_array: Creates a new array column. The input columns must all have the same data type.

create_map: Creates a new map column. The input columns must be grouped as key-value pairs, e.g. (key1, value1, key2, value2, ...). The key columns must all have the same data type, and can’t be null. The value columns must all have the same data type.

input_file_name: Creates a string column with the input file name for a given row. The method should be used with no argument.

Note

lit since 1.5.0
bitwiseNOT since 1.5.0
goalesce(Column) since 2.1.1
isnan since 2.0.0
is.nan since 2.0.0
negate since 1.5.0
spark_partition_id since 2.0.0
struct since 1.6.0
nanvl since 1.5.0
greatest since 1.5.0
least since 1.5.0
expr since 1.5.0
rand since 1.5.0
rand(numeric) since 1.5.0
randn since 1.5.0
randn(numeric) since 1.5.0
when since 1.5.0
ifelse since 1.5.0
create_array since 2.3.0
create_map since 2.3.0
input_file_name since 2.3.0
column_nonaggregate_functions

See Also
coaalesce, SparkDataFrame-method
Other non-aggregate functions: column(), not()

Examples

```
## Not run:
# Dataframe used throughout this doc
df <- createDataFrame(cbind(model = rownames(mtcars), mtcars))
## End(Not run)

## Not run:
tmp <- mutate(df, v1 = lit(df$mpg), v2 = lit("x"), v3 = lit("2015-01-01"),
v4 = negate(df$mpg), v5 = expr('length(model)'),
v6 = greatest(df$vs, df$am), v7 = least(df$vs, df$am),
v8 = column("mpg"))
head(tmp)
## End(Not run)

## Not run:
head(select(df, bitwiseNOT(cast(df$vs, "int"))))
## End(Not run)

## Not run: head(select(df, monotonically_increasing_id()))
## Not run: head(select(df, spark_partition_id()))

## Not run:
tmp <- mutate(df, v1 = struct(df$mpg, df$cyl), v2 = struct("hp", "wt", "vs"),
v3 = create_array(df$mpg, df$cyl, df$hp),
v4 = create_map(lit("x"), lit(1.0), lit("y"), lit(-1.0)))
head(tmp)
## End(Not run)

## Not run:
tmp <- mutate(df, r1 = rand(), r2 = rand(10), r3 = randn(), r4 = randn(10))
head(tmp)
## End(Not run)

## Not run:
tmp <- mutate(df, mpg_na = otherwise(when(df$mpg > 20, df$mpg), lit(NaN)),
mpg2 = ifelse(df$mpg > 20 & df$am > 0, 0, 1),
mpg3 = ifelse(df$mpg > 20, df$mpg, 20.0))
head(tmp)
tmp <- mutate(tmp, ind_na1 = is.nan(tmp$mpg_na), ind_na2 = isnan(tmp$mpg_na))
head(select(tmp, coalesce(tmp$mpg_na, tmp$mpg)))
head(select(tmp, nanvl(tmp$mpg_na, tmp$hp)))
## End(Not run)

## Not run:
tmp <- read.text("README.md")
head(select(tmp, input_file_name()))
```
String functions for Column operations

**Description**

String functions defined for Column.

**Usage**

- `ascii(x)`
- `base64(x)`
- `concat_ws(sep, x, ...)`
- `decode(x, charset)`
- `encode(x, charset)`
- `format_number(y, x)`
- `format_string(format, x, ...)`
- `initcap(x)`
- `instr(y, x)`
- `levenshtein(y, x)`
- `locate(substr, str, ...)`
- `lower(x)`
- `lpad(x, len, pad)`
- `ltrim(x, trimString)`
- `regexp_extract(x, pattern, idx)`
- `regexp_replace(x, pattern, replacement)`
- `repeat_string(x, n)`
- `rpad(x, len, pad)`
rtrim(x, trimString)
split_string(x, pattern)
soundex(x)
substring_index(x, delim, count)
translate(x, matchingString, replaceString)
trim(x, trimString)
unbase64(x)
upper(x)

## S4 method for signature 'Column'
ascii(x)

## S4 method for signature 'Column'
base64(x)

## S4 method for signature 'Column,character'
decode(x, charset)

## S4 method for signature 'Column,character'
encode(x, charset)

## S4 method for signature 'Column'
initcap(x)

## S4 method for signature 'Column'
length(x)

## S4 method for signature 'Column'
lower(x)

## S4 method for signature 'Column,missing'
ltrim(x, trimString)

## S4 method for signature 'Column,character'
ltrim(x, trimString)

## S4 method for signature 'Column,missing'
rtrim(x, trimString)

## S4 method for signature 'Column,character'
column_string_functions

rtrim(x, trimString)
## S4 method for signature 'Column'
soundex(x)
## S4 method for signature 'Column'
trim(x, trimString)
## S4 method for signature 'Column,missing'
trim(x, trimString)
## S4 method for signature 'Column,character'
trim(x, trimString)
## S4 method for signature 'Column'
unbase64(x)
## S4 method for signature 'Column'
upper(x)
## S4 method for signature 'Column'
levenshtein(y, x)
## S4 method for signature 'Column,character'
instr(y, x)
## S4 method for signature 'Column,numeric'
format_number(y, x)
## S4 method for signature 'character,Column'
concat_ws(sep, x, ...)
## S4 method for signature 'character,Column'
format_string(format, x, ...)
## S4 method for signature 'character,Column'
locate(substr, str, pos = 1)
## S4 method for signature 'Column,numerics,character'
lpad(x, len, pad)
## S4 method for signature 'Column,character,numerics'
regexp_extract(x, pattern, idx)
## S4 method for signature 'Column,character,character'
regexp_replace(x, pattern, replacement)
## S4 method for signature 'Column,numerics,character'
rpad(x, len, pad)
## S4 method for signature 'Column,character,numerics'
column_string_functions

substring_index(x, delim, count)

## S4 method for signature 'Column,character,character'
translate(x, matchingString, replaceString)

## S4 method for signature 'Column,character'
split_string(x, pattern)

## S4 method for signature 'Column,numeric'
repeat_string(x, n)

Arguments

x Column to compute on except in the following methods:
  • instr: character, the substring to check. See 'Details'.
  • format_number: numeric, the number of decimal place to format to. See 'Details'.
sep separator to use.
... additional Columns.
charset character set to use (one of "US-ASCII", "ISO-8859-1", "UTF-8", "UTF-16BE", "UTF-16LE", "UTF-16").
y Column to compute on.
format a character object of format strings.
substr a character string to be matched.
str a Column where matches are sought for each entry.
len maximum length of each output result.
pad a character string to be padded with.
trimString a character string to trim with
pattern a regular expression.
idx a group index.
replacement a character string that a matched pattern is replaced with.
n number of repetitions.
delim a delimiter string.
count number of occurrences of delim before the substring is returned. A positive number means counting from the left, while negative means counting from the right.
matchingString a source string where each character will be translated.
replaceString a target string where each matchingString character will be replaced by the character in replaceString at the same location, if any.
pos start position of search.
Details

ascii: Computes the numeric value of the first character of the string column, and returns the result as an int column.

base64: Computes the BASE64 encoding of a binary column and returns it as a string column. This is the reverse of unbase64.

decode: Computes the first argument into a string from a binary using the provided character set.
encode: Computes the first argument into a binary from a string using the provided character set.

initcap: Returns a new string column by converting the first letter of each word to uppercase. Words are delimited by whitespace. For example, "hello world" will become "Hello World".

length: Computes the character length of a string data or number of bytes of a binary data. The length of string data includes the trailing spaces. The length of binary data includes binary zeros.

lower: Converts a string column to lower case.
ltrim: Trims the spaces from left end for the specified string value. Optionally a trimString can be specified.
rtrim: Trims the spaces from right end for the specified string value. Optionally a trimString can be specified.

soundex: Returns the soundex code for the specified expression.
trim: Trims the spaces from both ends for the specified string column. Optionally a trimString can be specified.

unbase64: Decodes a BASE64 encoded string column and returns it as a binary column. This is the reverse of base64.

upper: Converts a string column to upper case.

levenshtein: Computes the Levenshtein distance of the two given string columns.
instr: Locates the position of the first occurrence of a substring (x) in the given string column (y). Returns null if either of the arguments are null. Note: The position is not zero based, but 1 based index. Returns 0 if the substring could not be found in the string column.

format_number: Formats numeric column y to a format like '#,###,###.##', rounded to x decimal places with HALF_EVEN round mode, and returns the result as a string column. If x is 0, the result has no decimal point or fractional part. If x < 0, the result will be null.

concat_ws: Concatenates multiple input string columns together into a single string column, using the given separator.

format_string: Formats the arguments in printf-style and returns the result as a string column.
locate: Locates the position of the first occurrence of substr. Note: The position is not zero based, but 1 based index. Returns 0 if substr could not be found in str.
lpad: Left-padded with pad to a length of len.
regexp_extract: Extracts a specific idx group identified by a Java regex, from the specified string column. If the regex did not match, or the specified group did not match, an empty string is returned.
regexp_replace: Replaces all substrings of the specified string value that match regexp with rep.
rpad: Right-padded with pad to a length of len.

substring_index: Returns the substring from string (x) before count occurrences of the delimiter (delim). If count is positive, everything the left of the final delimiter (counting from left) is
returned. If count is negative, every to the right of the final delimiter (counting from the right) is returned. `substring_index` performs a case-sensitive match when searching for the delimiter.

translate: Translates any character in the src by a character in replaceString. The characters in replaceString is corresponding to the characters in matchingString. The translate will happen when any character in the string matching with the character in the matchingString.

`split_string`: Splits string on regular expression. Equivalent to `split SQL function`.

`repeat_string`: Repeats string n times. Equivalent to `repeat SQL function`.

**Note**

- ascii since 1.5.0
- base64 since 1.5.0
- decode since 1.6.0
- encode since 1.6.0
- initcap since 1.5.0
- length since 1.5.0
- lower since 1.4.0
- ltrim since 1.5.0
- ltrim(Column, character) since 2.3.0
- rtrim since 1.5.0
- rtrim(Column, character) since 2.3.0
- soundex since 1.5.0
- trim since 1.5.0
- trim(Column, character) since 2.3.0
- unbase64 since 1.5.0
- upper since 1.4.0
- levenshtein since 1.5.0
- instr since 1.5.0
- format_number since 1.5.0
- concat_ws since 1.5.0
- format_string since 1.5.0
- locate since 1.5.0
- lpad since 1.5.0
- regexp_extract since 1.5.0
- regexp_replace since 1.5.0
- rpad since 1.5.0
- substring_index since 1.5.0
- translate since 1.5.0
- split_string 2.3.0
- repeat_string since 2.3.0
Examples

```r
## Not run:
# Dataframe used throughout this doc
df <- createDataFrame(as.data.frame(Titanic, stringsAsFactors = FALSE))
## End(Not run)

## Not run:
head(select(df, ascii(df$Class), ascii(df$Sex)))
## End(Not run)

## Not run:
tmp <- mutate(df, s1 = encode(df$Class, "UTF-8"))
str(tmp)
tmp2 <- mutate(tmp, s2 = base64(tmp$s1), s3 = decode(tmp$s1, "UTF-8"),
            s4 = soundex(tmp$Sex))
head(tmp2)
head(select(tmp2, unbase64(tmp2$s2)))
## End(Not run)

## Not run:
tmp <- mutate(df, sex_lower = lower(df$Sex), age_upper = upper(df$age),
               sex_age = concat_ws(" ", lower(df$sex), lower(df$age)))
head(tmp)
tmp2 <- mutate(tmp, s1 = initcap(tmp$sex_lower), s2 = initcap(tmp$sex_age),
               s3 = reverse(df$Sex))
head(tmp2)
## End(Not run)

## Not run:
tmp <- mutate(df, SexLpad = lpad(df$Sex, 6, " "), SexRpad = rpad(df$Sex, 7, " "))
head(select(tmp, length(tmp$Sex), length(tmp$SexLpad), length(tmp$SexRpad)))
tmp2 <- mutate(tmp, SexLtrim = ltrim(tmp$SexLpad), SexRtrim = rtrim(tmp$SexRpad),
               SexTrim = trim(tmp$SexLpad))
head(select(tmp2, length(tmp$Sex), length(tmp$SexLtrim),
       length(tmp$SexRtrim), length(tmp$SexTrim)))
tmp <- mutate(df, SexLpad = lpad(df$Sex, 6, "xx"), SexRpad = rpad(df$Sex, 7, "xx"))
head(tmp)
## End(Not run)

## Not run:
tmp <- mutate(df, d1 = levenshtein(df$Class, df$Sex),
               d2 = levenshtein(df$Age, df$Sex),
               d3 = levenshtein(df$Age, df$Age))
head(tmp)
## End(Not run)

## Not run:
tmp <- mutate(df, s1 = instr(df$Sex, "m"), s2 = instr(df$Sex, "M"),
               s3 = locate("m", df$Sex), s4 = locate("m", df$Sex, pos = 4))
head(tmp)
## End(Not run)
```
column_window_functions

Window functions for Column operations

Description

Window functions defined for Column.

Usage

cume_dist(x = "missing")
dense_rank(x = "missing")
lag(x, ...)
lead(x, offset, defaultValue = NULL)
ntile(x)
percent_rank(x = "missing")
rank(x, ...)
row_number(x = "missing")

## S4 method for signature 'missing'
cume_dist()

## S4 method for signature 'missing'
dense_rank()

## S4 method for signature 'characterOrColumn'
lag(x, offset = 1, defaultValue = NULL)

## S4 method for signature 'characterOrColumn,numeric'
lead(x, offset = 1, defaultValue = NULL)

## S4 method for signature 'numeric'
ntile(x)

## S4 method for signature 'missing'
percent_rank()

## S4 method for signature 'missing'
rnk()

## S4 method for signature 'ANY'
rank(x, ...)

## S4 method for signature 'missing'
row_number()

**Arguments**

- **x** In *lag* and *lead*, it is the column as a character string or a Column to compute on. In *ntile*, it is the number of *ntile* groups.
- **...** additional argument(s).
- **offset** In *lag*, the number of rows back from the current row from which to obtain a value. In *lead*, the number of rows after the current row from which to obtain a
value. If not specified, the default is 1.

defaultValue (optional) default to use when the offset row does not exist.

Details

cume_dist: Returns the cumulative distribution of values within a window partition, i.e. the fraction of rows that are below the current row: \((\text{number of values before and including } x) / (\text{total number of rows in the partition})\). This is equivalent to the CUME_DIST function in SQL. The method should be used with no argument.

dense_rank: Returns the rank of rows within a window partition, without any gaps. The difference between rank and dense_rank is that dense_rank leaves no gaps in ranking sequence when there are ties. That is, if you were ranking a competition using dense_rank and had three people tie for second place, you would say that all three were in second place and that the next person came in third. Rank would give me sequential numbers, making the person that came in third place (after the ties) would register as coming in fifth. This is equivalent to the DENSE_RANK function in SQL. The method should be used with no argument.

lag: Returns the value that is offset rows before the current row, and defaultValue if there is less than offset rows before the current row. For example, an offset of one will return the previous row at any given point in the window partition. This is equivalent to the LAG function in SQL.

lead: Returns the value that is offset rows after the current row, and defaultValue if there is less than offset rows after the current row. For example, an offset of one will return the next row at any given point in the window partition. This is equivalent to the LEAD function in SQL.

ntile: Returns the ntile group id (from 1 to n inclusive) in an ordered window partition. For example, if n is 4, the first quarter of the rows will get value 1, the second quarter will get 2, the third quarter will get 3, and the last quarter will get 4. This is equivalent to the NTILE function in SQL.

percent_rank: Returns the relative rank (i.e. percentile) of rows within a window partition. This is computed by: \((\text{rank of row in its partition - 1}) / (\text{number of rows in the partition - 1})\). This is equivalent to the PERCENT_RANK function in SQL. The method should be used with no argument.

rank: Returns the rank of rows within a window partition. The difference between rank and dense_rank is that dense_rank leaves no gaps in ranking sequence when there are ties. That is, if you were ranking a competition using dense_rank and had three people tie for second place, you would say that all three were in second place and that the next person came in third. Rank would give me sequential numbers, making the person that came in third place (after the ties) would register as coming in fifth. This is equivalent to the RANK function in SQL. The method should be used with no argument.

row_number: Returns a sequential number starting at 1 within a window partition. This is equivalent to the ROW_NUMBER function in SQL. The method should be used with no argument.

Note

cume_dist since 1.6.0
dense_rank since 1.6.0
lag since 1.6.0
lead since 1.6.0
corr

## Examples

```r
## Not run:
# Dataframe used throughout this doc
df <- createDataFrame(cbind(model = rownames(mtcars), mtcars))
ws <- orderBy(windowPartitionBy("am"), "hp")
tmp <- mutate(df, dist = over(cume_dist(), ws),
              dense_rank = over(dense_rank(), ws),
              lag = over(lag(df$mpg), ws),
              lead = over(lead(df$mpg, 1), ws),
              percent_rank = over(percent_rank(), ws),
              rank = over(rank(), ws),
              row_number = over(row_number(), ws))

# Get ntile group id (1-4) for hp
ntile(4), for hp

tmp <- mutate(tmp, ntile = over(ntile(4), ws))
head(tmp)
## End(Not run)
```

---

corr

### Description

Computes the Pearson Correlation Coefficient for two Columns.

### Usage

```r
corr(x, ...)
```

```r
## S4 method for signature 'Column'
corr(x, col2)
```

```r
## S4 method for signature 'SparkDataFrame'
corr(x, colName1, colName2, method = "pearson")
```

### Arguments

- `x` a Column or a SparkDataFrame.
- `...` additional argument(s). If `x` is a Column, a Column should be provided. If `x` is a SparkDataFrame, two column names should be provided.
- `col2` a (second) Column.
- `colName1` the name of the first column
- `colName2` the name of the second column
- `method` Optional. A character specifying the method for calculating the correlation. only "pearson" is allowed now.
count

Value

The Pearson Correlation Coefficient as a Double.

Note

corr since 1.6.0
corr since 1.6.0

See Also

Other aggregate functions: \texttt{avg()}, \texttt{column_aggregate_functions}, \texttt{count()}, \texttt{cov()}, \texttt{first()}, \texttt{last()}
Other stat functions: \texttt{approxQuantile()}, \texttt{cov()}, \texttt{crosstab()}, \texttt{freqItems()}, \texttt{sampleBy()}

Examples

\begin{verbatim}
## Not run:
df <- createDataFrame(cbind(model = rownames(mtcars), mtcars))
head(select(df, corr(df$mpg, df$hp)))
## End(Not run)

## Not run:
corr(df, "mpg", "hp")
corr(df, "mpg", "hp", method = "pearson")
## End(Not run)
\end{verbatim}

count

Description

Count the number of rows for each group when we have \texttt{GroupedData} input. The resulting SparkDataFrame will also contain the grouping columns.

This can be used as a column aggregate function with \texttt{Column} as input, and returns the number of items in a group.

Usage

count(x)

\begin{verbatim}
## S4 method for signature 'GroupedData'
count(x)

## S4 method for signature 'Column'
count(x)
\end{verbatim}
## S4 method for signature 'Column'
n(x)

**Arguments**

- `x`: a GroupedData or Column.

**Value**

A SparkDataFrame.

**Note**

- count since 1.4.0
- count since 1.4.0
- n since 1.4.0

**See Also**

Other aggregate functions: `avg()`, `column_aggregate_functions`, `corr()`, `cov()`, `first()`, `last()`

**Examples**

```r
## Not run:
count(groupBy(df, "name"))

## End(Not run)
## Not run: count(df$c)
## Not run: n(df$c)
```

---

cov  
cov

desc

description

describe

descricao

descripción
cov

```r
## S4 method for signature 'characterOrColumn,characterOrColumn'
covar_samp(col1, col2)

## S4 method for signature 'characterOrColumn,characterOrColumn'
covar_pop(col1, col2)

## S4 method for signature 'SparkDataframe'
cov(x, colName1, colName2)
```

**Arguments**

- `x` a Column or a SparkDataFrame.
- `...` additional argument(s). If `x` is a Column, a Column should be provided. If `x` is a SparkDataFrame, two column names should be provided.
- `col1` the first Column.
- `col2` the second Column.
- `colName1` the name of the first column
- `colName2` the name of the second column

**Details**

cov: Compute the sample covariance between two expressions.
covar_sample: Alias for `cov`.
covar_pop: Computes the population covariance between two expressions.
cov: When applied to SparkDataFrame, this calculates the sample covariance of two numerical columns of one SparkDataFrame.

**Value**

The covariance of the two columns.

**Note**

cov since 1.6.0
covar_samp since 2.0.0
covar_pop since 2.0.0
cov since 1.6.0

**See Also**

Other aggregate functions: `avg()`, `column_aggregate_functions`, `corr()`, `count()`, `first()`, `last()`

Other stat functions: `approxQuantile()`, `corr()`, `crosstab()`, `freqItems()`, `sampleBy()`
createDataFrame

Create a SparkDataFrame

Description

Converts R data.frame or list into SparkDataFrame.

Usage

```r
## Default S3 method:
createDataFrame(data, schema = NULL, samplingRatio = 1, numPartitions = NULL)
```

```r
## Default S3 method:
as.DataFrame(data, schema = NULL, samplingRatio = 1, numPartitions = NULL)
as.DataFrame(data, ...)
```

Arguments

- `data` - a list or data.frame.
- `schema` - a list of column names or named list (StructType), optional.
- `samplingRatio` - Currently not used.
- `numPartitions` - the number of partitions of the SparkDataFrame. Defaults to 1, this is limited by length of the list or number of rows of the data.frame
- `...` - additional argument(s).

Value

A SparkDataFrame.

Note

createDataFrame since 1.4.0
as.DataFrame since 1.6.0
createExternalTable

Examples

```r
# Not run:
sparkR.session()
df1 <- as.DataFrame(iris)
df2 <- as.DataFrame(list(3,4,5,6))
df3 <- createDataFrame(iris)
df4 <- createDataFrame(cars, numPartitions = 2)
```

## End(Not run)

createExternalTable  (Deprecated) Create an external table

Description

Creates an external table based on the dataset in a data source, Returns a SparkDataFrame associated with the external table.

Usage

```r
## Default S3 method:
createExternalTable(tableName, path = NULL, source = NULL, schema = NULL, ...)
```

Arguments

- `tableName`: a name of the table.
- `path`: the path of files to load.
- `source`: the name of external data source.
- `schema`: the schema of the data required for some data sources.
- `...`: additional argument(s) passed to the method.

Details

The data source is specified by the `source` and a set of options(...). If `source` is not specified, the default data source configured by "spark.sql.sources.default" will be used.

Value

A SparkDataFrame.

Note

createExternalTable since 1.4.0

See Also

createTable
createOrReplaceTempView

## Not run:
```r
sparkR.session()
```
```r
df <- createExternalTable("myjson", path="path/to/json", source="json", schema)
```
```r
## End(Not run)
```

## createOrReplaceTempView

### Description

Creates a temporary view using the given name.

### Usage

```r
createOrReplaceTempView(x, viewName)
```

### Arguments

- **x**: A SparkDataFrame
- **viewName**: A character vector containing the name of the table

### Note

createOrReplaceTempView since 2.0.0

### See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method,broadcast(),cache(),checkpoint(),coalesce(),collect(),
colnames(), coltypes(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(),
dim(),distinct(),dropDuplicates(),dropna(),drop(),dtypes(),exceptAll(),except(),
explain(),filter(),first(),gapplyCollect(),gapply(),getNumPartitions(),group_by(),
head(),hint(),histogram(),insertInto(),intersectAll(),intersect(),isLocal(),isStreaming(),
join(),limit(),localCheckpoint(),merge(),mutate(),ncol(),nrow(),persist(),printSchema(),
randomSplit(),rbind(),rename(),repartitionByRange(),repartition(),rollup(),sample(),
saveAsTable(),schema(),selectExpr(),select(),showDF(),show(),storageLevel(),str(),
subset(),summary(),take(),toJSON(),unionByName(),union(),unpersist(),withColumn(),
withWatermark(),with(),write.df(),write.jdbc(),write.json(),write.orc(),write.parquet(),
write.stream(),write.text()
Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
createOrReplaceTempView(df, "json_df")
new_df <- sql("SELECT * FROM json_df")
## End(Not run)
```

### createTable

`createTable` *Creates a table based on the dataset in a data source*

#### Description

Creates a table based on the dataset in a data source. Returns a SparkDataFrame associated with the table.

#### Usage

```r
createTable(tableName, path = NULL, source = NULL, schema = NULL, ...)
```

#### Arguments

- `tableName` the qualified or unqualified name that designates a table. If no database identifier is provided, it refers to a table in the current database.
- `path` (optional) the path of files to load.
- `source` (optional) the name of the data source.
- `schema` (optional) the schema of the data required for some data sources.
- `...` additional named parameters as options for the data source.

#### Details

The data source is specified by the `source` and a set of options(...). If `source` is not specified, the default data source configured by "spark.sql.sources.default" will be used. When a path is specified, an external table is created from the data at the given path. Otherwise a managed table is created.

#### Value

A SparkDataFrame.

#### Note

`createTable` since 2.2.0
Examples

```r
## Not run:
sparkR.session()
df <- createTable("myjson", path="path/to/json", source="json", schema)
createTable("people", source = "json", schema = schema)
insertInto(df, "people")
## End(Not run)
```

crossJoin

### Description

Returns Cartesian Product on two SparkDataFrames.

### Usage

```r
## S4 method for signature 'SparkDataFrame,SparkDataFrame'
crossJoin(x, y)
```

### Arguments

- `x`: A SparkDataFrame
- `y`: A SparkDataFrame

### Value

A SparkDataFrame containing the result of the join operation.

### Note

crossJoin since 2.1.0

### See Also

merge join

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach()`, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `group_by()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isLocal()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `ncol()`, `nrow()`, `persist()`, `printSchema()`, `randomSplit()`, `rbind()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()`, `sample()`, `saveAsTable()`, `schema()`, `selectExpr()`, `select()`, ...
crosstab

\[ \text{crosstab}(x, \text{col1}, \text{col2}) \]

**Arguments**
- \( x \): a SparkDataFrame
- \( \text{col1} \): name of the first column. Distinct items will make the first item of each row.
- \( \text{col2} \): name of the second column. Distinct items will make the column names of the output.

**Value**
- a local R data.frame representing the contingency table. The first column of each row will be the distinct values of \( \text{col1} \) and the column names will be the distinct values of \( \text{col2} \). The name of the first column will be "\( \text{col1}_\text{col2} \)". Pairs that have no occurrences will have zero as their counts.

**Note**
- crosstab since 1.5.0

**See Also**
- Other stat functions: \text{approxQuantile()}, \text{corr()}, \text{cov()}, \text{freqItems()}, \text{sampleBy()}
Examples

## Not run:
df <- read.json("/path/to/file.json")
ct <- crosstab(df, "title", "gender")

## End(Not run)

cube
cube

description
Create a multi-dimensional cube for the SparkDataFrame using the specified columns.

Usage

cube(x, ...)

## S4 method for signature 'SparkDataFrame'
cube(x, ...)

Arguments

x
a SparkDataFrame.

... character name(s) or Column(s) to group on.

Details

If grouping expression is missing cube creates a single global aggregate and is equivalent to direct application of agg.

Value

A GroupedData.

Note

cube since 2.3.0

See Also

agg, groupBy, rollup

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(),
currentDatabase

Returns the current default database.

Usage

currentDatabase()

Value

name of the current default database.

Note

since 2.2.0

Examples

## Not run:
sparkR.session()
currentDatabase()

## End(Not run)
**Description**

Apply a function to each partition of a SparkDataFrame.

**Usage**

dapply(x, func, schema)

## S4 method for signature 'SparkDataFrame,`function`,characterOrstructType'
dapply(x, func, schema)

**Arguments**

x
A SparkDataFrame

func
A function to be applied to each partition of the SparkDataFrame. func should have only one parameter, to which a R data.frame corresponds to each partition will be passed. The output of func should be a R data.frame.

schema
The schema of the resulting SparkDataFrame after the function is applied. It must match the output of func. Since Spark 2.3, the DDL-formatted string is also supported for the schema.

**Note**

dapply since 2.0.0

**See Also**

dapplyCollect

Other SparkDataFrame functions: SparkDataFrame-class,agg(),alias(),arrange(),as.data.frame(),attach,SparkDataFrame-method,broadcast(),cache(),checkpoint(),coalesce(),collect(),colnames(),coltypes().createOrReplaceTempView().crossJoin().cube().dapplyCollect().describe().dim().distinct().dropDuplicates().dropna().drop().dtypes().exceptAll().except().explain().filter().first().gapplyCollect().gapply().getNumPartitions().groupBy().head().hint().histogram().insertInto().intersectAll().intersect().isLocal().isStreaming().join().limit().localCheckpoint().merge().mutate().ncol().nrow().persist().printSchema().randomSplit().rename().repartitionByRange().repartition().rollup().sample().saveAsTable().schema().selectExpr().select().showDF().show().storageLevel().str().subset().summary().take().toJSON().unionByName().union(). unpersist().withColumn().withColumnRenamed().withWatermark().with().write.df().write.jdbc().write.json().write.orc().write.parquet().write.stream().write.text()
Examples

```r
## Not run:
df <- createDataFrame(iris)
df1 <- dapply(df, function(x) { x }, schema(df))
collect(df1)

# filter and add a column
df <- createDataFrame(
    list(list(1L, 1, "1"), list(2L, 2, "2"), list(3L, 3, "3")),
    c("a", "b", "c"))
schema <- structType(structField("a", "integer"), structField("b", "double"),
    structField("c", "string"), structField("d", "integer"))
df1 <- dapply(
    df,
    function(x) {
        y <- x[x[1] > 1,]
        y <- cbind(y, y[1] + 1L)
    },
    schema)

# The schema also can be specified in a DDL-formatted string.
schema <- "a INT, d DOUBLE, c STRING, d INT"
df1 <- dapply(
    df,
    function(x) {
        y <- x[x[1] > 1,]
        y <- cbind(y, y[1] + 1L)
    },
    schema)

collect(df1)
# the result
# a  b  c  d
# 1 2 2 2
# 3 3 3 4

## End(Not run)
```

Description

Apply a function to each partition of a SparkDataFrame and collect the result back to R as a data.frame.

Usage

dapplyCollect(x, func)
dapplyCollect

## S4 method for signature 'SparkDataFrame,'function'
dapplyCollect(x, func)

Arguments

x
A SparkDataFrame

func
A function to be applied to each partition of the SparkDataFrame. func should have only one parameter, to which a R data.frame corresponds to each partition will be passed. The output of func should be a R data.frame.

Note
dapplyCollect since 2.0.0

See Also
dapply

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.text(),

Examples

## Not run:
df <- createDataFrame(iris)
ldf <- dapplyCollect(df, function(x) { x })

# filter and add a column
df <- createDataFrame(
  list(list(1L, 1, "1"), list(2L, 2, "2"), list(3L, 3, "3")),
c("a", "b", "c")
ldf <- dapplyCollect(
  df,
  function(x) {
    y <- x[x[1] > 1, ]
    y <- cbind(y, y[1] + 1L)
  })
# the result
#   a  b  c  d
# 2 2 2 3
### DecisionTreeClassificationModel-class

S4 class that represents a DecisionTreeClassificationModel

#### Description

S4 class that represents a DecisionTreeClassificationModel

#### Arguments

- **jobj**
  
a Java object reference to the backing Scala DecisionTreeClassificationModel

#### Note

DecisionTreeClassificationModel since 2.3.0

### DecisionTreeRegressionModel-class

S4 class that represents a DecisionTreeRegressionModel

#### Description

S4 class that represents a DecisionTreeRegressionModel

#### Arguments

- **jobj**
  
a Java object reference to the backing Scala DecisionTreeRegressionModel

#### Note

DecisionTreeRegressionModel since 2.3.0
**Description**

Computes statistics for numeric and string columns. If no columns are given, this function computes statistics for all numerical or string columns.

**Usage**

```r
describe(x, col, ...)
```

## S4 method for signature 'SparkDataFrame,character'
```r
describe(x, col, ...)
```

## S4 method for signature 'SparkDataFrame,ANY'
```r
describe(x)
```

**Arguments**

- `x` a SparkDataFrame to be computed.
- `col` a string of name.
- `...` additional expressions.

**Value**

A SparkDataFrame.

**Note**

describe(SparkDataFrame, character) since 1.4.0

describe(SparkDataFrame) since 1.4.0

**See Also**

See `summary` for expanded statistics and control over which statistics to compute.

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method,broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with().write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()
Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
describe(df)
describe(df, "col1")
describe(df, "col1", "col2")
## End(Not run)
```

**dim**

*dim* Returns the dimensions of SparkDataFrame

**Description**

Returns the dimensions (number of rows and columns) of a SparkDataFrame

**Usage**

```r
## S4 method for signature 'SparkDataFrame'

#dim(x)
```

**Arguments**

- **x**: a SparkDataFrame

**Note**

dim since 1.5.0

**See Also**

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach`, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `group_by()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isLocal()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `ncol()`, `nrow()`, `persist()`, `printSchema()`, `randomSplit()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()` , `sample()` , `saveAsTable()` , `schema()` , `selectExpr()` , `select()` , `showDF()` , `show()` , `storageLevel()` , `str()` , `subset()` , `summary()` , `take()` , `toJSON()` , `unionByName()` , `union()` , ` unpersist()` , `withColumn()` , `withWatermark()` , `with()` , `write.df()` , `write.jdbc()` , `write.json()` , `write.orc()` , `write.parquet()` , `write.stream()` , `write.text()`
## distinct

### Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
dim(df)

## End(Not run)
```

### Description

Return a new SparkDataFrame containing the distinct rows in this SparkDataFrame.

### Usage

```r
distinct(x)
```

#### S4 method for signature 'SparkDataFrame'

```r
distinct(x)
```

#### S4 method for signature 'SparkDataFrame'

```r
unique(x)
```

### Arguments

- `x`: A SparkDataFrame

### Note

- distinct since 1.4.0
- unique since 1.5.0

### See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()
Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
distinctDF <- distinct(df)

## End(Not run)
```

Description

Returns a new SparkDataFrame with columns dropped. This is a no-op if schema doesn’t contain column name(s).

Usage

```r
drop(x, ...)
```

## S4 method for signature 'SparkDataFrame'
drop(x, col)

## S4 method for signature 'ANY'
drop(x)

Arguments

- `x` a SparkDataFrame.
- `...` further arguments to be passed to or from other methods.
- `col` a character vector of column names or a Column.

Value

A SparkDataFrame.

Note

`drop` since 2.0.0
See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
drop(df, "col1")
drop(df, c("col1", "col2"))
drop(df, df$col1)
## End(Not run)
```

---

### Description

Returns a new SparkDataFrame with duplicate rows removed, considering only the subset of columns.

### Usage

```r
dropDuplicates(x, ...)
```

```
## S4 method for signature 'SparkDataFrame'
dropDuplicates(x, ...)
```

### Arguments

- `x`: A SparkDataFrame.
- `...`: A character vector of column names or string column names. If the first argument contains a character vector, the followings are ignored.
dropna

A set of SparkDataFrame functions working with NA values

Description

dropna, na.omit - Returns a new SparkDataFrame omitting rows with null values.

Usage

dropna(x, how = c("any", "all"), minNonNulls = NULL, cols = NULL)

na.omit(object, ...)

fillna(x, value, cols = NULL)
## S4 method for signature 'SparkDataFrame'
dropna(x, how = c("any", "all"), minNonNulls = NULL, cols = NULL)

## S4 method for signature 'SparkDataFrame'
na.omit(object, how = c("any", "all"), minNonNulls = NULL, cols = NULL)

## S4 method for signature 'SparkDataFrame'
fillna(x, value, cols = NULL)

### Arguments

- **x**: a SparkDataFrame.
- **how**: "any" or "all". if "any", drop a row if it contains any nulls. if "all", drop a row only if all its values are null. if minNonNulls is specified, how is ignored.
- **minNonNulls**: if specified, drop rows that have less than minNonNulls non-null values. This overwrites the how parameter.
- **cols**: optional list of column names to consider. In fillna, columns specified in cols that do not have matching data type are ignored. For example, if value is a character, and subset contains a non-character column, then the non-character column is simply ignored.
- **object**: a SparkDataFrame.
- **...**: further arguments to be passed to or from other methods.
- **value**: value to replace null values with. Should be an integer, numeric, character or named list. If the value is a named list, then cols is ignored and value must be a mapping from column name (character) to replacement value. The replacement value must be an integer, numeric or character.

### Value

A SparkDataFrame.

### Note

- dropna since 1.4.0
- na.omit since 1.5.0
- fillna since 1.4.0

### See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach(), SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(),
dropTempTable

nrow().persist().printSchema().randomSplit().rbind().rename().repartitionByRange().
repartition().rollup().sample().saveAsTable().schema().selectExpr().select().
showDF().show().storageLevel().str().subset().summary().take().toJSON().unionByName().
union().unpersist().withColumn().withWatermark().with().write.df().write.jdbc().
write.json().write.orc().write.parquet().write.stream().write.text()

Examples

## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
dropna(df)

## End(Not run)
## Not run:
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
fillna(df, 1)
fillna(df, list("age" = 20, "name" = "unknown"))

## End(Not run)

dropTempTable (Deprecated) Drop Temporary Table

Description

Drops the temporary table with the given table name in the catalog. If the table has been cached/persisted
before, it’s also unpersisted.

Usage

## Default S3 method:
dropTempTable(tableName)

Arguments

tableName The name of the SparkSQL table to be dropped.

Note

dropTempTable since 1.4.0

See Also

dropTempView
dropTempView

Drops the temporary view with the given view name in the catalog.

Description

Drops the temporary view with the given view name in the catalog. If the view has been cached before, then it will also be uncached.

Usage

dropTempView(viewName)

Arguments

viewName the name of the temporary view to be dropped.

Value

TRUE if the view is dropped successfully, FALSE otherwise.

Note

since 2.0.0

Examples

```r
## Not run:
sparkR.session()
df <- read.df(path, "parquet")
createOrReplaceTempView(df, "table")
dropTempTable("table")

dropTempView("table")
```

## End(Not run)
dtypes

<table>
<thead>
<tr>
<th>dtypes</th>
<th>DataTypes</th>
</tr>
</thead>
</table>

**Description**

Return all column names and their data types as a list

**Usage**

```
dtypes(x)
```

```r
## S4 method for signature 'SparkDataFrame'
dtypes(x)
```

**Arguments**

- `x` A SparkDataFrame

**Note**

dtypes since 1.4.0

**See Also**

Other SparkDataFrame functions: SparkDataFrame-class, ag(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

**Examples**

```r
## Not run:
sparkR.session()
peth <- "path/to/file.json"
df <- read.json(path)
dtypes(df)

## End(Not run)```
endsWith

Description
Determines if entries of x end with string (entries of) suffix respectively, where strings are recycled to common lengths.

Usage
endsWith(x, suffix)

## S4 method for signature 'Column'
endsWith(x, suffix)

Arguments
x vector of character string whose "ends" are considered
suffix character vector (often of length one)

Note
endsWith since 1.4.0

See Also
Other column_func: alias(), between(), cast(), otherwise(), over(), startsWith(), substr()

except

Description
Return a new SparkDataFrame containing rows in this SparkDataFrame but not in another SparkDataFrame. This is equivalent to EXCEPT DISTINCT in SQL.

Usage
except(x, y)

## S4 method for signature 'SparkDataFrame,SparkDataFrame'
except(x, y)

Arguments
x a SparkDataFrame.
y a SparkDataFrame.
exceptAll

Value

A SparkDataFrame containing the result of the except operation.

Note

except since 1.4.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text

Examples

## Not run:
sparkR.session()
df1 <- read.json(path)
df2 <- read.json(path2)
extexceptDF <- except(df, df2)

## End(Not run)

exceptAll

Description

Return a new SparkDataFrame containing rows in this SparkDataFrame but not in another SparkDataFrame while preserving the duplicates. This is equivalent to EXCEPT ALL in SQL. Also as standard in SQL, this function resolves columns by position (not by name).

Usage

exceptAll(x, y)

## S4 method for signature 'SparkDataFrame,SparkDataFrame'
exceptAll(x, y)
Arguments

- x: a SparkDataFrame.
- y: a SparkDataFrame.

Value

A SparkDataFrame containing the result of the except all operation.

Note

exceptAll since 2.4.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapply(), dapplyCollect(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), except(), explain(), filter(), first(), gapply(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
df1 <- read.json(path)
df2 <- read.json(path2)
exceptAllDF <- exceptAll(df1, df2)
## End(Not run)
```

---

**Description**

Print the logical and physical Catalyst plans to the console for debugging.
Usage

```r
explain(x, ...)
```

## S4 method for signature 'SparkDataFrame'
```r
explain(x, extended = FALSE)
```

## S4 method for signature 'StreamingQuery'
```r
explain(x, extended = FALSE)
```

Arguments

- `x` a SparkDataFrame or a StreamingQuery.
- `...` further arguments to be passed to or from other methods.
- `extended` Logical. If extended is FALSE, prints only the physical plan.

Note

`explain` since 1.4.0
`explain(StreamingQuery)` since 2.2.0

See Also

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach`, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `group_by()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isLocal()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `ncol()`, `nrow()`, `persist()`, `printSchema()`, `randomSplit()`, `rbind()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()`, `sample()`, `saveAsTable()`, `schema()`, `selectExpr()`, `select()`, `showDF()`, `show()`, `storageLevel()`, `str()`, `subset()`, `summary()`, `take()`, `toJSON()`, `unionByName()`, `union()`, `unpersist()`, `withColumn()`, `withWatermark()`, `with()`, `write.df()`, `write.jdbc()`, `write.json()`, `write.orc()`, `write.parquet()`, `write.stream()`, `write.text()`

Other StreamingQuery methods: `awaitTermination()`, `isActive()`, `lastProgress()`, `queryName()`, `status()`, `stopQuery()`

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
explain(df, TRUE)
```

## End(Not run)
```r
## Not run: explain(sq)
```
Description

Filter the rows of a SparkDataFrame according to a given condition.

Usage

```r
filter(x, condition)
where(x, condition)
```

Arguments

- `x` A SparkDataFrame to be sorted.
- `condition` The condition to filter on. This may either be a Column expression or a string containing a SQL statement.

Value

A SparkDataFrame containing only the rows that meet the condition.

Note

- filter since 1.4.0
- where since 1.4.0

See Also

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach()`, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `group_by()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isLocal()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `nrow()`, `persist()`, `printSchema()`, `randomSplit()`, `rbind()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()`, `sample()`, `saveAsTable()`, `schema()`, `selectExpr()`, `select()`, `showDF()`, `show()`, `storageLevel()`, `str()`, `subset()`, `summary()`, `take()`, `toJSON()`, `unionByName()`,
union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Other subsetting functions: select(), subset()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
filter(df, "col1 > 0")
filter(df, df$col2 != "abcdefg")

## End(Not run)
```

### first

Return the first row of a SparkDataFrame

#### Description

Aggregate function: returns the first value in a group.

#### Usage

```r
first(x, ...)
```

- `x`: a SparkDataFrame or a column used in aggregation function.
- `...`: further arguments to be passed to or from other methods.
- `na.rm`: a logical value indicating whether NA values should be stripped before the computation proceeds.

#### Details

The function by default returns the first values it sees. It will return the first non-missing value it sees when `na.rm` is set to true. If all values are missing, then NA is returned. Note: the function is non-deterministic because its results depends on order of rows which may be non-deterministic after a shuffle.
Note

first(SparkDataFrame) since 1.4.0
first(characterOrColumn) since 1.4.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withColumnRenamed(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Other aggregate functions: avg(), column_aggregate_functions, corr(), count(), cov(), last()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
first(df)

## End(Not run)
## Not run:
first(df$c)
first(df$c, TRUE)

## End(Not run)
```

---

*fitted*  
Get fitted result from a k-means model

**Description**

Get fitted result from a k-means model, similarly to R's `fitted()`. Note: A saved-loaded model does not support this method.

**Usage**

```r
fitted(object, ...)  
```

**S4 method for signature 'KMeansModel'**

```r
fitted(object, method = c("centers", "classes"))
```
**Arguments**

- **object**: a fitted k-means model.
- **...**: additional argument(s) passed to the method.
- **method**: type of fitted results, "centers" for cluster centers or "classes" for assigned classes.

**Value**

- `fitted` returns a SparkDataFrame containing fitted values.

**Note**

- `fitted` since 2.0.0

**Examples**

```r
## Not run:
model <- spark.kmeans(trainingData, ~ ., 2)
fitted.model <- fitted(model)
showDF(fitted.model)
## End(Not run)
```

---

**Description**

S4 class that represents a FPGrowthModel

**Arguments**

- **jobj**: a Java object reference to the backing Scala FPGrowthModel

**Note**

- FPGrowthModel since 2.2.0
freqItems

Finding frequent items for columns, possibly with false positives

Description

Finding frequent items for columns, possibly with false positives. Using the frequent element count algorithm described in http://dx.doi.org/10.1145/762471.762473, proposed by Karp, Schenker, and Papadimitriou.

Usage

## S4 method for signature 'SparkDataFrame,character'
freqItems(x, cols, support = 0.01)

Arguments

x
A SparkDataFrame.

cols
A vector column names to search frequent items in.

support
(Optional) The minimum frequency for an item to be considered frequent. Should be greater than 1e-4. Default support = 0.01.

Value

a local R data.frame with the frequent items in each column

Note

cfreqItems since 1.6.0

See Also

Other stat functions: approxQuantile(), corr(), cov(), crosstab(), sampleBy()

Examples

## Not run:
df <- read.json("/path/to/file.json")
fi = freqItems(df, c("title", "gender"))

## End(Not run)
Description

Groups the SparkDataFrame using the specified columns and applies the R function to each group.

Usage

gapply(x, ...)

---

## S4 method for signature 'GroupedData'
gapply(x, func, schema)

## S4 method for signature 'SparkDataFrame'
gapply(x, cols, func, schema)

Arguments

- **x**: a SparkDataFrame or GroupedData.
- **...**: additional argument(s) passed to the method.
- **func**: a function to be applied to each group partition specified by grouping column of the SparkDataFrame. See Details.
- **schema**: the schema of the resulting SparkDataFrame after the function is applied. The schema must match to output of `func`. It has to be defined for each output column with preferred output column name and corresponding data type. Since Spark 2.3, the DDL-formatted string is also supported for the schema.
- **cols**: grouping columns.

Details

`func` is a function of two arguments. The first, usually named `key` (though this is not enforced) corresponds to the grouping key, will be an unnamed list of length(cols) length-one objects corresponding to the grouping columns’ values for the current group.

The second, herein `x`, will be a local `data.frame` with the columns of the input not in `cols` for the rows corresponding to `key`.

The output of `func` must be a `data.frame` matching `schema` – in particular this means the names of the output `data.frame` are irrelevant.

Value

A SparkDataFrame.

Note

gapply(GroupedData) since 2.0.0
gapply(SparkDataFrame) since 2.0.0
See Also

gapplyCollect

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(). str(). subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(). write.df(). write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(). write.text()

Examples

```
## Not run:
# Computes the arithmetic mean of the second column by grouping
# on the first and third columns. Output the grouping values and the average.

df <- createDataFrame(
  list(list(1L, 1, "1", 0.1), list(1L, 2, "1", 0.2), list(3L, 3, "3", 0.3)),
  c("a", "b", "c", "d"))

# Here our output contains three columns, the key which is a combination of two
# columns with data types integer and string and the mean which is a double.
schema <- structType(structField("a", "integer"), structField("c", "string"),
  structField("avg", "double"))
result <- gapply(
  df,
  c("a", "c"),
  function(key, x) {
    # key will either be list(1L, '1') (for the group where a=1L,c='1') or
    # list(3L, '3') (for the group where a=3L,c='3')
    y <- data.frame(key, mean(x$b), stringsAsFactors = FALSE)
  }, schema)

# The schema also can be specified in a DDL-formatted string.
schema <- "a INT, c STRING, avg DOUBLE"
result <- gapply(
  df,
  c("a", "c"),
  function(key, x) {
    y <- data.frame(key, mean(x$b), stringsAsFactors = FALSE)
  }, schema)

# We can also group the data and afterwards call gapply on GroupedData.
# For example:
gdf <- group_by(df, "a", "c")
```
result <- gapply(
  gdf,
  function(key, x) {
    y <- data.frame(key, mean(x$b), stringsAsFactors = FALSE)
  }, schema)

collect(result)

# Result
# ------
# a c avg
# 3 3 3.0
# 1 1 1.5

# Fits linear models on iris dataset by grouping on the 'Species' column and
# using 'Sepal_Length' as a target variable, 'Sepal_Width', 'Petal_Length'
# and 'Petal_Width' as training features.

df <- createDataFrame(iris)
schema <- structType(structField("(Intercept)", "double"),
                      structField("Sepal_Width", "double"), structField("Petal_Length", "double"),
                      structField("Petal_Width", "double"))

df1 <- gapply(
  df,
  df$"Species",
  function(key, x) {
    m <- suppressWarnings(lm(Sepal_Length ~
                           Sepal_Width + Petal_Length + Petal_Width, x))
    data.frame(t(coef(m)))
  }, schema)

collect(df1)

# Result
# --------
# Model  (Intercept) Sepal_Width Petal_Length Petal_Width
# 1 0.699883 0.3303370 0.9455356 -0.1697527
# 2 1.895540 0.3868576 0.9083370 -0.6792238
# 3 2.351890 0.6548350 0.2375602 0.2521257

## End(Not run)

gapplyCollect
gapplyCollect

Description

Groups the SparkDataFrame using the specified columns, applies the R function to each group and
collects the result back to R as data.frame.
Usage

gapplyCollect(x, ...)

## S4 method for signature 'GroupedData'
gapplyCollect(x, func)

## S4 method for signature 'SparkDataFrame'
gapplyCollect(x, cols, func)

Arguments

- **x**: a SparkDataFrame or GroupedData.
- **...**: additional argument(s) passed to the method.
- **func**: a function to be applied to each group partition specified by grouping column of the SparkDataFrame. See Details.
- **cols**: grouping columns.

Details

func is a function of two arguments. The first, usually named key (though this is not enforced) corresponds to the grouping key, will be an unnamed list of length(cols) length-one objects corresponding to the grouping columns’ values for the current group.

The second, herein x, will be a local data.frame with the columns of the input not in cols for the rows corresponding to key.

The output of func must be a data.frame matching schema – in particular this means the names of the output data.frame are irrelevant.

Value

A data.frame.

Note

gapplyCollect(GroupedData) since 2.0.0

gapplyCollect(SparkDataFrame) since 2.0.0

See Also

gapply

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(),
Examples

```r
## Not run:
# Computes the arithmetic mean of the second column by grouping
# on the first and third columns. Output the grouping values and the average.

df <- createDataFrame(
  list(list(1L, 1, "1", 0.1), list(1L, 2, "1", 0.2), list(3L, 3, "3", 0.3)),
  c("a", "b", "c", "d"))
result <- gapplyCollect(
  df,
  c("a", "c"),
  function(key, x) {
    y <- data.frame(key, mean(x$b), stringsAsFactors = FALSE)
    colnames(y) <- c("key_a", "key_c", "mean_b")
    y
  })

# We can also group the data and afterwards call gapply on GroupedData.
# For example:
gdf <- group_by(df, "a", "c")
result <- gapplyCollect(
  gdf,
  function(key, x) {
    y <- data.frame(key, mean(x$b), stringsAsFactors = FALSE)
    colnames(y) <- c("key_a", "key_c", "mean_b")
    y
  })

# Result
# -----
# key_a key_c mean_b
# 3 3 3.0
# 1 1 1.5

# Fits linear models on iris dataset by grouping on the 'Species' column and
# using 'Sepal_Length' as a target variable, 'Sepal_Width', 'Petal_Length'
# and 'Petal_Width' as training features.

df <- createDataFrame(iris)
result <- gapplyCollect(
  df,
  df$"Species",
  function(key, x) {
    m <- suppressWarnings(lm(Sepal_Length ~
```
```r
GBTClassificationModel-class

S4 class that represents a GBTClassificationModel

Description

S4 class that represents a GBTClassificationModel

Arguments

jobj  a Java object reference to the backing Scala GBTClassificationModel

Note

GBTClassificationModel since 2.1.0
```

---

```r
GaussianMixtureModel-class

S4 class that represents a GaussianMixtureModel

Description

S4 class that represents a GaussianMixtureModel

Arguments

jobj  a Java object reference to the backing Scala GaussianMixtureModel

Note

GaussianMixtureModel since 2.1.0
```
GBTRegressionModel-class

S4 class that represents a GBTRegressionModel

Description

S4 class that represents a GBTRegressionModel

Arguments

jobj a Java object reference to the backing Scala GBTRegressionModel

Note

GBTRegressionModel since 2.1.0

GeneralizedLinearRegressionModel-class

S4 class that represents a generalized linear model

Description

S4 class that represents a generalized linear model

Arguments

jobj a Java object reference to the backing Scala GeneralizedLinearRegressionWrapper

Note

GeneralizedLinearRegressionModel since 2.0.0
getLocalProperty

*Description*

Get a local property set in this thread, or NULL if it is missing. See `setLocalProperty`.

*Usage*

```r
getLocalProperty(key)
```

*Arguments*

- **key**
  The key for a local property.

*Note*

`getLocalProperty` since 2.3.0

*Examples*

```r
# Not run:
getLocalProperty("spark.scheduler.pool")
# End(Not run)
```

getNumPartitions

*Description*

Return the number of partitions

*Usage*

```r
## S4 method for signature 'SparkDataFrame'
getNumPartitions(x)
```

*Arguments*

- **x**
  A SparkDataFrame

*Note*

`getNumPartitions` since 2.1.1
Generalized Linear Models (R-compliant)

Description

Fits a generalized linear model, similarly to R’s glm().

Usage

```r
## S4 method for signature 'formula,ANY,SparkDataFrame'

glm(
  formula,
  family = gaussian,
  data,
  epsilon = 1e-06,
  maxit = 25,
  weightCol = NULL,
  var.power = 0,
  link.power = 1 - var.power,
  stringIndexerOrderType = c("frequencyDesc", "frequencyAsc", "alphabetDesc",
                            "alphabetAsc"),
  offsetCol = NULL
)
```
Arguments

- **formula**: a symbolic description of the model to be fitted. Currently only a few formula operators are supported, including `~`, `.`., `:`., `+`, and `-`.
- **family**: a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. Refer R family at [https://stat.ethz.ch/R-manual/R-devel/library/stats/html/family.html](https://stat.ethz.ch/R-manual/R-devel/library/stats/html/family.html). Currently these families are supported: binomial, gaussian, poisson, Gamma, and tweedie.

- **data**: a SparkDataFrame or R’s glm data for training.
- **epsilon**: positive convergence tolerance of iterations.
- **maxit**: integer giving the maximal number of IRLS iterations.
- **weightCol**: the weight column name. If this is not set or NULL, we treat all instance weights as 1.0.
- **var.power**: the index of the power variance function in the Tweedie family.
- **link.power**: the index of the power link function in the Tweedie family.
- **stringIndexerOrderType**: how to order categories of a string feature column. This is used to decide the base level of a string feature as the last category after ordering is dropped when encoding strings. Supported options are "frequencyDesc", "frequencyAsc", "alphabetDesc", and "alphabetAsc". The default value is "frequencyDesc". When the ordering is set to "alphabetDesc", this drops the same category as R when encoding strings.

- **offsetCol**: the offset column name. If this is not set or empty, we treat all instance offsets as 0.0. The feature specified as offset has a constant coefficient of 1.0.

Value

- **glm**: returns a fitted generalized linear model.

Note

- **glm since 1.5.0**

See Also

- **spark.glm**

Examples

```r
## Not run:
sparkR.session()
t <- as.data.frame(Titanic)
df <- createDataFrame(t)
model <- glm(Freq ~ Sex + Age, df, family = "gaussian")
summary(model)

## End(Not run)
```
Description

GroupedDatas can be created using groupBy() on a SparkDataFrame

Usage

groupedData(sgd)

Arguments

sgd A Java object reference to the backing Scala GroupedData

Note

GroupedData since 1.4.0

See Also

groupBy

description

Description

Groups the SparkDataFrame using the specified columns, so we can run aggregation on them.

Usage

group_by(x, ...)

groupBy(x, ...)

## S4 method for signature 'SparkDataFrame'
groupBy(x, ...)

## S4 method for signature 'SparkDataFrame'
group_by(x, ...)

Arguments

x a SparkDataFrame.

... character name(s) or Column(s) to group on.
hashCode

Compute the hashCode of an object

Description

Java-style function to compute the hashCode for the given object. Returns an integer value.

Usage

hashCode(key)
**Arguments**

key the object to be hashed

**Details**

This only works for integer, numeric and character types right now.

**Value**

the hash code as an integer

**Note**

hashCode since 1.4.0

**Examples**

```r
## Not run:
hashCode(1L) # 1
hashCode(1.0) # 1072693248
hashCode("1") # 49

## End(Not run)
```

---

**Description**

Return the first `num` rows of a SparkDataFrame as a R data.frame. If `num` is not specified, then `head()` returns the first 6 rows as with R data.frame.

**Usage**

```r
## S4 method for signature 'SparkDataFrame'
head(x, num = 6L)
```

**Arguments**

x a SparkDataFrame.

num the number of rows to return. Default is 6.

**Value**

A data.frame.

**Note**

head since 1.4.0
See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
head(df)

## End(Not run)
```

---

### Description

Specifies execution plan hint and return a new SparkDataFrame.

### Usage

```r
hint(x, name, ...)  
```

### Arguments

- `x`  
  a SparkDataFrame.

- `name`  
  a name of the hint.

- `...`  
  optional parameters for the hint.

### Value

A SparkDataFrame.
### histogram

Compute histogram statistics for given column

**Description**

This function computes a histogram for a given SparkR Column.

**Usage**

```r
## S4 method for signature 'SparkDataFrame,characterOrColumn'
histogram(df, col, nbins = 10)
```

**Arguments**

- `df`: the SparkDataFrame containing the Column to build the histogram from.
- `col`: the column as Character string or a Column to build the histogram from.
- `nbins`: the number of bins (optional). Default value is 10.
Value

a data.frame with the histogram statistics, i.e., counts and centroids.

Note

histogram since 2.0.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first().gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()
Usage

```r
insertInto(x, tableName, ...)  
```

## S4 method for signature 'SparkDataFrame,character'

```r
insertInto(x, tableName, overwrite = FALSE)
```

Arguments

- `x` a SparkDataFrame.
- `tableName` a character vector containing the name of the table.
- `...` further arguments to be passed to or from other methods. the existing rows in the table.
- `overwrite` a logical argument indicating whether or not to overwrite.

Note

`insertInto` since 1.4.0

See Also

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach()`, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `group_by()`, `head()`, `hint()`, `histogram()`, `intersectAll()`, `intersect()`, `isLocal()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `ncol()`, `nrow()`, `persist()`, `printSchema()`, `randomSplit()`, `rbind()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()`, `sample()`, `saveAsTable()`, `schema()`, `selectExpr()`, `select()`, `showDF()`, `show()`, `storageLevel()`, `str()`, `subset()`, `summary()`, `take()`, `toJSON()`, `unionByName()`, `union()`, `unpersist()`, `withColumn()`, `withWatermark()`, `with()`, `write.df()`, `write.jdbc()` `write.json()`, `write.orc()`, `write.parquet()`, `write.stream()`, `write.text()`

Examples

```r
## Not run:
sparkR.session()

df <- read.df(path, "parquet")

df2 <- read.df(path2, "parquet")

saveAsTable(df, "table1")

insertInto(df2, "table1", overwrite = TRUE)

## End(Not run)
```
install.spark  
Download and Install Apache Spark to a Local Directory

Description

install.spark downloads and installs Spark to a local directory if it is not found. If SPARK_HOME is set in the environment, and that directory is found, that is returned. The Spark version we use is the same as the SparkR version. Users can specify a desired Hadoop version, the remote mirror site, and the directory where the package is installed locally.

Usage

install.spark(
    hadoopVersion = "2.7",
    mirrorUrl = NULL,
    localDir = NULL,
    overwrite = FALSE
)

Arguments

hadoopVersion  Version of Hadoop to install. Default is "2.7". It can take other version number in the format of "x.y" where x and y are integer. If hadoopVersion = "without", "Hadoop free" build is installed. See "Hadoop Free" Build for more information. Other patched version names can also be used, e.g. "cdh4"

mirrorUrl  base URL of the repositories to use. The directory layout should follow Apache mirrors.

localDir  a local directory where Spark is installed. The directory contains version-specific folders of Spark packages. Default is path to the cache directory:

  • Mac OS X: '~/Library/Caches/spark'
  • Unix: $XDG_CACHE_HOME if defined, otherwise '~/cache/spark'
  • Windows: '%LOCALAPPDATA%\Apache\Spark\Cache'.

overwrite  If TRUE, download and overwrite the existing tar file in localDir and force reinstall Spark (in case the local directory or file is corrupted)

Details

The full url of remote file is inferred from mirrorUrl and hadoopVersion. mirrorUrl specifies the remote path to a Spark folder. It is followed by a subfolder named after the Spark version (that corresponds to SparkR), and then the tar filename. The filename is composed of four parts, i.e. [Spark version]-bin-[Hadoop version].tgz. For example, the full path for a Spark 2.0.0 package for Hadoop 2.7 from http://apache.osuosl.org has path: http://apache.osuosl.org/spark/spark-2.0.0/spark-2.0.0-bin-hadoop2.7.tgz.

For hadoopVersion = "without", [Hadoop version] in the filename is then without-hadoop.

Value

the (invisible) local directory where Spark is found or installed
**intersect**

**Note**

install.spark since 2.1.0

**See Also**

See available Hadoop versions: Apache Spark

**Examples**

```r
## Not run:
install.spark()

## End(Not run)
```

---

<table>
<thead>
<tr>
<th>intersect</th>
<th>Intersect</th>
</tr>
</thead>
</table>

**Description**

Return a new SparkDataFrame containing rows only in both this SparkDataFrame and another SparkDataFrame. This is equivalent to INTERSECT in SQL.

**Usage**

```r
intersect(x, y)
```

```
## S4 method for signature 'SparkDataFrame,SparkDataFrame'
intersect(x, y)
```

**Arguments**

- `x` A SparkDataFrame
- `y` A SparkDataFrame

**Value**

A SparkDataFrame containing the result of the intersect.

**Note**

intersect since 1.4.0
### intersectAll

Return a new SparkDataFrame containing rows in both this SparkDataFrame and another SparkDataFrame while preserving the duplicates. This is equivalent to `INTERSECT ALL` in SQL. Also as standard in SQL, this function resolves columns by position (not by name).

#### Usage

```r
intersectAll(x, y)
```

#### Arguments

- `x`: a SparkDataFrame.
- `y`: a SparkDataFrame.

#### Value

A SparkDataFrame containing the result of the intersect all operation.

---

#### Examples

```r
## Not run:
sparkR.session()
df1 <- read.json(path)
df2 <- read.json(path2)
intersectDF <- intersect(df1, df2)
## End(Not run)
```
isActive

Note
intersectAll since 2.4.0

See Also
Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text

Examples
```r
## Not run:
sparkR.session()
df1 <- read.json(path)
df2 <- read.json(path2)
intersectAllDF <- intersectAll(df1, df2)

## End(Not run)
```

isActive  isActive

Description
Returns TRUE if this query is actively running.

Usage
```r
isActive(x)
```

Arguments
- `x` a StreamingQuery.

Value
TRUE if query is actively running, FALSE if stopped.
isLocal

Note
isActive(StreamingQuery) since 2.2.0
experimental

See Also
Other StreamingQuery methods: `awaitTermination()`, `explain()`, `lastProgress()`, `queryName()`, `status()`, `stopQuery()`

Examples
```r
## Not run: isActive(sq)
```

### Description
Returns True if the collect and take methods can be run locally (without any Spark executors).

### Usage
```r
isLocal(x)
```

### Arguments
- `x` A SparkDataFrame

### Note
isLocal since 1.4.0

### See Also
Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach()`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `group_by()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `ncol()`, `nrow()`, `persist()`, `printSchema()`, `randomSplit()`, `rbind()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()`, `sample()`, `saveAsTable()`, `schema()`, `selectExpr()`, `select()`, `showDF()`, `show()`, `storageLevel()`, `str()`, `subset()`, `summary()`, `take()`, `toJSON()`, `unionByName()`, `union()`, `unpersist()`, `withColumn()`, `withWatermark()`, `with()`, `write.df()`, `write.jdbc()`, `write.json()`, `write.orc()`, `write.parquet()`, `write.stream()`, `write.text()`
Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
isLocal(df)

## End(Not run)
```

**IsotonicRegressionModel-class**

*S4 class that represents an IsotonicRegressionModel*

Description

S4 class that represents an IsotonicRegressionModel

Arguments

jobj

a Java object reference to the backing Scala IsotonicRegressionModel

Note

IsotonicRegressionModel since 2.1.0

**isStreaming**

*isStreaming*

Description

Returns TRUE if this SparkDataFrame contains one or more sources that continuously return data as it arrives. A dataset that reads data from a streaming source must be executed as a StreamingQuery using write.stream.

Usage

`isStreaming(x)`

```r
## S4 method for signature 'SparkDataFrame'
isStreaming(x)
```

Arguments

x

A SparkDataFrame
**Value**

TRUE if this SparkDataFrame is from a streaming source

**Note**

isStreaming since 2.2.0

experimental

**See Also**

read.stream write.stream

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

**Examples**

```r
## Not run:
sparkR.session()
df <- read.stream("socket", host = "localhost", port = 9999)
isStreaming(df)

## End(Not run)
```

---

### join

#### Join

**Description**

Joins two SparkDataFrames based on the given join expression.

**Usage**

```r
## S4 method for signature 'SparkDataFrame,SparkDataFrame'
join(x, y, joinExpr = NULL, joinType = NULL)
```
Arguments

- **x**: A SparkDataFrame
- **y**: A SparkDataFrame
- **joinExpr**: (Optional) The expression used to perform the join. `joinExpr` must be a Column expression. If `joinExpr` is omitted, the default, inner join is attempted and an error is thrown if it would be a Cartesian Product. For Cartesian join, use `crossJoin` instead.

- **joinType**: The type of join to perform, default 'inner'. Must be one of: 'inner', 'cross', 'outer', 'full', 'full_outer', 'left', 'left_outer', 'right', 'right_outer', 'left_semi', or 'left_anti'.

Value

A SparkDataFrame containing the result of the join operation.

Note

join since 1.4.0

See Also

merge, crossJoin

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
df1 <- read.json(path)
df2 <- read.json(path2)
join(df1, df2, df1$col1 == df2$col2) # Performs an inner join based on expression
join(df1, df2, df1$col1 == df2$col2, "right_outer")
join(df1, df2) # Attempts an inner join
```

## End(Not run)
KMeansModel-class

S4 class that represents a KMeansModel

Description

S4 class that represents a KMeansModel

Arguments

- `jobj` a Java object reference to the backing Scala KMeansModel

Note

KMeansModel since 2.0.0

KSTest-class

S4 class that represents a KSTest

Description

S4 class that represents a KSTest

Arguments

- `jobj` a Java object reference to the backing Scala KSTestWrapper

Note

KSTest since 2.1.0

last

Aggregate function: returns the last value in a group.

Usage

```
last(x, ...)
```

```r
## S4 method for signature 'characterOrColumn'
last(x, na.rm = FALSE)
```
**Arguments**

- **x**
  - column to compute on.
- **...**
  - further arguments to be passed to or from other methods.
- **na.rm**
  - a logical value indicating whether NA values should be stripped before the computation proceeds.

**Details**

The function by default returns the last values it sees. It will return the last non-missing value it sees when na.rm is set to true. If all values are missing, then NA is returned. Note: the function is non-deterministic because its results depend on order of rows which may be non-deterministic after a shuffle.

**Note**

last since 1.4.0

**See Also**

Other aggregate functions: `avg()`, `column_aggregate_functions`, `corr()`, `count()`, `cov()`, `first()`

**Examples**

```r
## Not run:
last(df$c)
last(df$c, TRUE)
## End(Not run)
```

---

### Description

Prints the most recent progress update of this streaming query in JSON format.

### Usage

```r
lastProgress(x)
```

```r
# S4 method for signature 'StreamingQuery'
lastProgress(x)
```

### Arguments

- **x**
  - a StreamingQuery.
Note

lastProgress(StreamingQuery) since 2.2.0
experimental

See Also

Other StreamingQuery methods: awaitTermination(), explain(), isActive(), queryName(), status(), stopQuery()

Examples

## Not run: lastProgress(sq)

---

LDAModel-class

S4 class that represents an LDAModel

Description

S4 class that represents an LDAModel

Arguments

- obj: a Java object reference to the backing Scala LDAWrapper

Note

LDAModel since 2.1.0

---

limit

Limit

Description

Limit the resulting SparkDataFrame to the number of rows specified.

Usage

limit(x, num)

## S4 method for signature 'SparkDataFrame,numeric'
limit(x, num)

Arguments

- x: A SparkDataFrame
- num: The number of rows to return
Value

A new SparkDataFrame containing the number of rows specified.

Note

 limit since 1.4.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method,broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
limitedDF <- limit(df, 10)
## End(Not run)
```

---

LinearSVCModel-class  
S4 class that represents an LinearSVCModel

Description

S4 class that represents an LinearSVCModel

Arguments

jobj a Java object reference to the backing Scala LinearSVCModel

Note

LinearSVCModel since 2.2.0
listColumns

Returns a list of columns for the given table/view in the specified database.

Usage

listColumns(tableName, databaseName = NULL)

Arguments

tableName the qualified or unqualified name that designates a table/view. If no database identifier is provided, it refers to a table/view in the current database. If databaseName parameter is specified, this must be an unqualified name.

databaseName (optional) name of the database

Value

a SparkDataFrame of the list of column descriptions.

Note

since 2.2.0

Examples

## Not run:
sparkR.session()
listColumns("mytable")

## End(Not run)

listDatabases

Returns a list of databases available.

Description

Returns a list of databases available.

Usage

listDatabases()
listFunctions

Value

a SparkDataFrame of the list of databases.

Note

since 2.2.0

Examples

## Not run:
sparkR.session()
listDatabases()

## End(Not run)

listFunctions  Returns a list of functions registered in the specified database

Description

Returns a list of functions registered in the specified database. This includes all temporary functions.

Usage

listFunctions(databaseName = NULL)

Arguments

databaseName  (optional) name of the database

Value

a SparkDataFrame of the list of function descriptions.

Note

since 2.2.0

Examples

## Not run:
sparkR.session()
listFunctions()

## End(Not run)
### listTables

*Returns a list of tables or views in the specified database*

**Description**

Returns a list of tables or views in the specified database. This includes all temporary views.

**Usage**

```r
listTables(databaseName = NULL)
```

**Arguments**

- `databaseName` (optional) name of the database

**Value**

a SparkDataFrame of the list of tables.

**Note**

since 2.2.0

**See Also**

tables

**Examples**

```r
## Not run:
sparkR.session()
listTables()
listTables("default")

## End(Not run)
```

---

### localCheckpoint

*localCheckpoint*

**Description**

Returns a locally checkpointed version of this SparkDataFrame. Checkpointing can be used to truncate the logical plan, which is especially useful in iterative algorithms where the plan may grow exponentially. Local checkpoints are stored in the executors using the caching subsystem and therefore they are not reliable.
Usage

localCheckpoint(x, eager = TRUE)

## S4 method for signature 'SparkDataFrame'
localCheckpoint(x, eager = TRUE)

Arguments

x 
A SparkDataFrame

eager 
whether to locally checkpoint this SparkDataFrame immediately

Value

a new locally checkpointed SparkDataFrame

Note

localCheckpoint since 2.3.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

## Not run:
df <- localCheckpoint(df)

## End(Not run)
LogisticRegressionModel-class

_S4 class that represents an LogisticRegressionModel_

Description

_S4 class that represents an LogisticRegressionModel_

Arguments

_jobj_ a Java object reference to the backing Scala LogisticRegressionModel

Note

LogisticRegressionModel since 2.1.0

merge

_Merges two data frames_

Description

Merges two data frames

Usage

merge(x, y, ...)

## S4 method for signature 'SparkDataFrame,SparkDataFrame'
merge(
x,
y,
by = intersect(names(x), names(y)),
by.x = by,
by.y = by,
all = FALSE,
all.x = all,
all.y = all,
sort = TRUE,
suffixes = c("_x", "_y"),
...
)
merge

Arguments

- **x** the first data frame to be joined.
- **y** the second data frame to be joined.
- **...** additional argument(s) passed to the method.
- **by** a character vector specifying the join columns. If by is not specified, the common column names in x and y will be used. If by or both by.x and by.y are explicitly set to NULL or of length 0, the Cartesian Product of x and y will be returned.
- **by.x** a character vector specifying the joining columns for x.
- **by.y** a character vector specifying the joining columns for y.
- **all** a boolean value setting all.x and all.y if any of them are unset.
- **all.x** a boolean value indicating whether all the rows in x should be including in the join.
- **all.y** a boolean value indicating whether all the rows in y should be including in the join.
- **sort** a logical argument indicating whether the resulting columns should be sorted.
- **suffixes** a string vector of length 2 used to make colnames of x and y unique. The first element is appended to each colname of x. The second element is appended to each colname of y.

Details

If all.x and all.y are set to FALSE, a natural join will be returned. If all.x is set to TRUE and all.y is set to FALSE, a left outer join will be returned. If all.x is set to FALSE and all.y is set to TRUE, a right outer join will be returned. If all.x and all.y are set to TRUE, a full outer join will be returned.

Note

merge since 1.5.0

See Also

join crossJoin

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename().repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()
Examples

```r
## Not run:
sparkR.session()
df1 <- read.json(path)
df2 <- read.json(path2)
merge(df1, df2) # Performs an inner join by common columns
merge(df1, df2, by = "col1") # Performs an inner join based on expression
merge(df1, df2, by.x = "col1", by.y = "col2", all.y = TRUE)
merge(df1, df2, by.x = "col1", by.y = "col2", all.x = TRUE)
merge(df1, df2, by.x = "col1", by.y = "col2", all.x = TRUE, all.y = TRUE)
merge(df1, df2, by.x = "col1", by.y = "col2", all = TRUE, sort = FALSE)
merge(df1, df2, by = "col1", all = TRUE, suffixes = c("-X", "-Y"))
merge(df1, df2, by = NULL) # Performs a Cartesian join
```
## End(Not run)

---

**MultilayerPerceptronClassificationModel-class**

*S4 class that represents a MultilayerPerceptronClassificationModel*

---

**Description**

S4 class that represents a MultilayerPerceptronClassificationModel

**Arguments**

- `jobj`  
  a Java object reference to the backing Scala MultilayerPerceptronClassifier-Wrapper

**Note**

MultilayerPerceptronClassificationModel since 2.1.0

---

**mutate**

*Mutate*

---

**Description**

Return a new SparkDataFrame with the specified columns added or replaced.
**mutate**

Usage

```r
mutate(.data, ...)  
transform(`_data`, ...)
```

## S4 method for signature 'SparkDataFrame'

```r
mutate(.data, ...)
```

## S4 method for signature 'SparkDataFrame'

```r
transform(`_data`, ...)
```

### Arguments

- `.data` a SparkDataFrame.
- `...` additional column argument(s) each in the form name = col.
- `_data` a SparkDataFrame.

### Value

A new SparkDataFrame with the new columns added or replaced.

### Note

- mutate since 1.4.0
- transform since 1.5.0

### See Also

- `rename withColumn`

Other SparkDataFrame functions: `SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except().explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hist(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

### Examples

```r
## Not run:  
sparkR.session()  
path <- "path/to/file.json"  
df <- read.json(path)
```
newDF <- mutate(df, newCol = df$col1 * 5, newCol2 = df$col1 * 2)
names(newDF) # Will contain newCol, newCol2
newDF2 <- transform(df, newCol = df$col1 / 5, newCol2 = df$col1 * 2)

df <- createDataFrame(list(list("Andy", 30L), list("Justin", 19L)), c("name", "age"))
# Replace the "age" column
df1 <- mutate(df, age = df$age + 1L)

## End(Not run)

---

**NaiveBayesModel-class**  
*S4 class that represents a NaiveBayesModel*

**Description**  
S4 class that represents a NaiveBayesModel

**Arguments**  

**obj**  
a Java object reference to the backing Scala NaiveBayesWrapper

**Note**  
NaiveBayesModel since 2.0.0

---

**ncol**  
*Returns the number of columns in a SparkDataFrame*

**Description**  
Returns the number of columns in a SparkDataFrame

**Usage**  

```r
## S4 method for signature 'SparkDataFrame'
ncol(x)
```

**Arguments**  

**x**  
a SparkDataFrame

**Note**  
ncol since 1.5.0
See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
ncol(df)

## End(Not run)
```

```
not
!
```

Description

Inversion of boolean expression.

Inversion of boolean expression.

Usage

```r	not(x)
```

```
# S4 method for signature 'Column'
!x
```

```
# S4 method for signature 'Column'
not(x)
```

Arguments

- `x`: Column to compute on
Details
not and ! cannot be applied directly to numerical column. To achieve R-like truthiness column has to be casted to BooleanType.

Note
! since 2.3.0
not since 2.3.0

See Also
Other non-aggregate functions: column_nonaggregate_functions, column()

Examples
## Not run:
df <- createDataFrame(data.frame(x = c(-1, 0, 1)))

head(select(df, !column("x") > 0))

## End(Not run)
## Not run:
df <- createDataFrame(data.frame(
  is_true = c(TRUE, FALSE, NA),
  flag = c(1, 0, 1)
))

head(select(df, not(df$is_true)))

# Explicit cast is required when working with numeric column
head(select(df, not(cast(df$flag, "boolean"))))

## End(Not run)

nrow

Returns the number of rows in a SparkDataFrame

Description
Returns the number of rows in a SparkDataFrame

Usage
## S4 method for signature 'SparkDataFrame'
count(x)

## S4 method for signature 'SparkDataFrame'
nrow(x)
Arguments

x a SparkDataFrame.

Note

count since 1.4.0
nrow since 1.5.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class.agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByNames(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
count(df)

## End(Not run)
```

Description

Defines the ordering columns in a WindowSpec.

Usage

```r
orderBy(x, col, ...)
```

```
# S4 method for signature 'WindowSpec,character'
orderBy(x, col, ...)
```

```
# S4 method for signature 'WindowSpec,Column'
orderBy(x, col, ...)
```
otherwise

Arguments

x a WindowSpec
col a character or Column indicating an ordering column
... additional sorting fields

Value
A WindowSpec.

Note
orderBy(WindowSpec, character) since 2.0.0
orderBy(WindowSpec, Column) since 2.0.0

See Also
See arrange for use in sorting a SparkDataFrame
Other windowspec_method: partitionBy(), rangeBetween(), rowsBetween()

Examples

## Not run:
orderBy(ws, "col1", "col2")
orderBy(ws, df$col1, df$col2)
## End(Not run)

otherwise

Description
If values in the specified column are null, returns the value. Can be used in conjunction with when to specify a default value for expressions.

Usage

otherwise(x, value)

## S4 method for signature 'Column'
otherwise(x, value)

Arguments

x a Column.
value value to replace when the corresponding entry in x is NA. Can be a single value or a Column.
over

Note
otherwise since 1.5.0

See Also
Other column_func: alias(), between(), cast(), endsWith(), over(), startsWith(), substr()

Description
Define a windowing column.

Usage
over(x, window)

## S4 method for signature 'Column,WindowSpec'
over(x, window)

Arguments
x a Column, usually one returned by window function(s).
window a WindowSpec object. Can be created by windowPartitionBy or windowOrderBy and configured by other WindowSpec methods.

Note
over since 2.0.0

See Also
Other column_func: alias(), between(), cast(), endsWith(), otherwise(), startsWith(), substr()

Examples
## Not run:
df <- createDataFrame(mtcars)

# Partition by am (transmission) and order by hp (horsepower)
ws <- orderBy(windowPartitionBy("am"), "hp")

# Rank on hp within each partition
out <- select(df, over(rank(), ws), df$hp, df$am)

# Lag mpg values by 1 row on the partition-and-ordered table
out <- select(df, over(lead(df$mpg), ws), df$mpg, df$hp, df$am)

## End(Not run)

---

**partitionBy**

**Description**

Defines the partitioning columns in a WindowSpec.

**Usage**

```r
partitionBy(x, ...)
```

```r
# S4 method for signature 'WindowSpec'
partitionBy(x, col, ...)
```

**Arguments**

- `x` a WindowSpec.
- `...` additional column(s) to partition on.
- `col` a column to partition on (described by the name or `Column`).

**Value**

A WindowSpec.

**Note**

`partitionBy(WindowSpec)` since 2.0.0

**See Also**

Other `windowspec_method`: `orderBy()`, `rangeBetween()`, `rowsBetween()`

**Examples**

```r
# Not run:
partitionBy(ws, "col1", "col2")
partitionBy(ws, df$col1, df$col2)

# End(Not run)
```
Persist this SparkDataFrame with the specified storage level. For details of the supported storage levels, refer to http://spark.apache.org/docs/latest/rdd-programming-guide.html#rdd-persistence.

Usage

```r
persist(x, newLevel)
```

## S4 method for signature 'SparkDataFrame,character'

`persist(x, newLevel)`

Arguments

- **x**: the SparkDataFrame to persist.
- **newLevel**: storage level chosen for the persistence. See available options in the description.

Note

`persist` since 1.4.0

See Also

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach()``, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `group_by()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isLocal()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `nrow()`, `printSchema()`, `randomSplit()`, `rbind()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()`, `sample()`, `saveAsTable()`, `schema()`, `selectExpr()`, `select()`, `showDF()`, `show()`, `storageLevel()`, `str()`, `subset()`, `summary()`, `take()`, `toJSON()`, `unionByName()`, `union()`, `unpersist()`, `withColumn()`, `withWatermark()`, `with()`, `write.df()`, `write.jdbc()`, `write.json()`, `write.orc()`, `write.parquet()`, `write.stream()`, `write.text()`

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
persist(df, "MEMORY_AND_DISK")
```

## End(Not run)
pivot

Pivot a column of the GroupedData and perform the specified aggregation.

Description

Pivot a column of the GroupedData and perform the specified aggregation. There are two versions of pivot function: one that requires the caller to specify the list of distinct values to pivot on, and one that does not. The latter is more concise but less efficient, because Spark needs to first compute the list of distinct values internally.

Usage

```r
## S4 method for signature 'GroupedData,character'
pivot(x, colname, values = list())
```

Arguments

- `x`: a GroupedData object
- `colname`: A column name
- `values`: A value or a list/vector of distinct values for the output columns.

Value

GroupedData object

Note

pivot since 2.0.0

Examples

```r
## Not run:
df <- createDataFrame(data.frame(
    earnings = c(10000, 10000, 11000, 15000, 12000, 20000, 21000, 22000),
    period = c("1H", "1H", "2H", "2H", "1H", "1H", "2H", "2H"),
))
group_sum <- sum(pivot(groupBy(df, "year"), "course"), "earnings")
group_min <- min(pivot(groupBy(df, "year"), "course", "R"), "earnings")
group_max <- max(pivot(groupBy(df, "year"), "course", c("Python", "R")), "earnings")
group_mean <- mean(pivot(groupBy(df, "year"), "course", list("Python", "R")), "earnings")
```

## End(Not run)
**predict**  
*Makes predictions from a MLlib model*

**Description**

Makes predictions from a MLlib model. For more information, see the specific MLlib model below.

**Usage**

```r
predict(object, ...)
```

**Arguments**

- `object` a fitted ML model object.
- `...` additional argument(s) passed to the method.

**See Also**

`spark.als`, `spark.bisectingKmeans`, `spark.decisionTree`, `spark.gaussianMixture`, `spark.gbt`, `spark.glm`, `glm`, `spark.isoreg`, `spark.kmeans`, `spark.logit`, `spark.mlp`, `spark.naiveBayes`, `spark.randomForest`, `spark.survreg`, `spark.svmLinear`

---

**print.jobj**  
*Print a JVM object reference.*

**Description**

This function prints the type and id for an object stored in the SparkR JVM backend.

**Usage**

```r
## S3 method for class 'jobj'
print(x, ...)
```

**Arguments**

- `x` The JVM object reference
- `...` further arguments passed to or from other methods

**Note**

`print.jobj` since 1.4.0
print.structField  
*Print a Spark StructField.*

**Description**

This function prints the contents of a StructField returned from the SparkR JVM backend.

**Usage**

```r
## S3 method for class 'structField'
print(x, ...)
```

**Arguments**

- `x`  
  A StructField object
- `...`  
  further arguments passed to or from other methods

**Note**

print.structField since 1.4.0

---

print.structType  
*Print a Spark StructType.*

**Description**

This function prints the contents of a StructType returned from the SparkR JVM backend.

**Usage**

```r
## S3 method for class 'structType'
print(x, ...)
```

**Arguments**

- `x`  
  A StructType object
- `...`  
  further arguments passed to or from other methods

**Note**

print.structType since 1.4.0
printSchema

Print Schema of a SparkDataFrame

Description

Prints out the schema in tree format

Usage

printSchema(x)

## S4 method for signature 'SparkDataFrame'

printSchema(x)

Arguments

x

A SparkDataFrame

Note

printSchema since 1.4.0

See Also

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach()`, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `group_by()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isLocal()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `nrow()`, `persist()`, `randomSplit()`, `rbind()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()`, `sample()`, `saveAsTable()`, `schema()`, `selectExpr()`, `select()`, `showDF()`, `show()`, `storageLevel()`, `str()`, `subset()`, `summary()`, `take()`, `toJSON()`, `unionByName()`, `union()`, `unpersist()`, `withColumn()`, `withWatermark()`, `with()`, `write.df()`, `write.jdbc()`, `write.json()`, `write.orc()`, `write.parquet()`, `write.stream()`, `write.text()`

Examples

## Not run:

sparkR.session()

path <- "path/to/file.json"

df <- read.json(path)

printSchema(df)

## End(Not run)
queryName

Description

Returns the user-specified name of the query. This is specified in `write.stream(df, queryName = "query")`. This name, if set, must be unique across all active queries.

Usage

```r
queryName(x)
```

## S4 method for signature 'StreamingQuery'

```r
queryName(x)
```

Arguments

- `x`: a StreamingQuery.

Value

The name of the query, or NULL if not specified.

Note

`queryName(StreamingQuery)` since 2.2.0

experimental

See Also

- `write.stream`

Other StreamingQuery methods: `awaitTermination()`, `explain()`, `isActive()`, `lastProgress()`, `status()`, `stopQuery()`

Examples

```r
## Not run: queryName(sq)
```
RandomForestClassificationModel-class

*S4 class that represents a RandomForestClassificationModel*

---

**Description**

S4 class that represents a RandomForestClassificationModel

**Arguments**

- `jobj` a Java object reference to the backing Scala RandomForestClassificationModel

**Note**

RandomForestClassificationModel since 2.1.0

---

RandomForestRegressionModel-class

*S4 class that represents a RandomForestRegressionModel*

---

**Description**

S4 class that represents a RandomForestRegressionModel

**Arguments**

- `jobj` a Java object reference to the backing Scala RandomForestRegressionModel

**Note**

RandomForestRegressionModel since 2.1.0
randomSplit

Description

Return a list of randomly split dataframes with the provided weights.

Usage

randomSplit(x, weights, seed)

## S4 method for signature 'SparkDataFrame,numeric'
randomSplit(x, weights, seed)

Arguments

x
A SparkDataFrame

weights
A vector of weights for splits, will be normalized if they don’t sum to 1

seed
A seed to use for random split

Note

randomSplit since 2.0.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method,broadcast(),cache(),checkpoint(),coalesce(),collect(), colnames(),coltypes(),createOrReplaceTempView(),crossJoin(),cube(),dapplyCollect(),
dapply(),describe(),dim(),distinct(),dropDuplicates(),dropna(),drop(),dtypes(),
exceptAll(),except(),explain(),filter(),first(),gapplyCollect(),gapply(),getNumPartitions(),
group_by(),head(),hint(),histogram(),insertInto(),intersectAll(),intersect(),
isLocal(),isStreaming(),join(),limit(),localCheckpoint(),merge(),mutate(),ncol(),
nrow(),persist().printSchema(),rbind(),rename().repartitionByRange(),repartition(),
rollup(),sample().saveAsTable(),schema().selectExpr().select().showDF(),show(),
storageLevel().str().subset().summary().take().toJSON(),unionByName(),union(),
unpersist().withColumn(),withWatermark(),with(),write.df(),write.jdbc(),write.json(),
write.orc(),write.parquet(),write.stream(),write.text()
rangeBetween

## End(Not run)

---

rangeBetween  rangeBetween

**Description**

Defines the frame boundaries, from start (inclusive) to end (inclusive).

**Usage**

rangeBetween(x, start, end)

## S4 method for signature 'WindowSpec,numeric,numeric'

rangeBetween(x, start, end)

**Arguments**

- **x**  
  a WindowSpec
- **start**  
  boundary start, inclusive. The frame is unbounded if this is the minimum long value.
- **end**  
  boundary end, inclusive. The frame is unbounded if this is the maximum long value.

**Details**

Both start and end are relative from the current row. For example, "0" means "current row", while 
"-1" means one off before the current row, and "5" means the five off after the current row.

**Value**

a WindowSpec

**Note**

rangeBetween since 2.0.0

**See Also**

Other windowspec_method: orderBy(), partitionBy(), rowsBetween()

**Examples**

## Not run:

rangeBetween(ws, 0, 3)

## End(Not run)
**Description**

Union two or more SparkDataFrames by row. As in R’s `rbind`, this method requires that the input SparkDataFrames have the same column names.

**Usage**

```
rbind(..., deparse.level = 1)
```

```
## S4 method for signature 'SparkDataFrame'
rbind(x, ..., deparse.level = 1)
```

**Arguments**

- `...` additional SparkDataFrame(s).
- `deparse.level` currently not used (put here to match the signature of the base implementation).
- `x` a SparkDataFrame.

**Details**

Note: This does not remove duplicate rows across the two SparkDataFrames.

**Value**

A SparkDataFrame containing the result of the union.

**Note**

`rbind` since 1.5.0

**See Also**

`union unionByName`

Other SparkDataFrame functions: `SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube().dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()`
Examples

```r
## Not run:
sparkR.session()
unions <- rbind(df, df2, df3, df4)
## End(Not run)
```

Description

Returns the dataset in a data source as a SparkDataFrame

Usage

```r
## Default S3 method:
read.df(path = NULL, source = NULL, schema = NULL, na.strings = "NA", ...)
```

```r
## Default S3 method:
loadDF(path = NULL, source = NULL, schema = NULL, ...)
```

Arguments

- `path` The path of files to load
- `source` The name of external data source
- `schema` The data schema defined in structType or a DDL-formatted string.
- `na.strings` Default string value for NA when source is "csv"
- `...` additional external data source specific named properties.

Details

The data source is specified by the `source` and a set of options(...). If `source` is not specified, the default data source configured by "spark.sql.sources.default" will be used.

Similar to R `read.csv`, when source is "csv", by default, a value of "NA" will be interpreted as NA.

Value

SparkDataFrame

Note

- `read.df` since 1.4.0
- `loadDF` since 1.6.0
See Also

read.json

Examples

```r
## Not run:
sparkR.session()
df1 <- read.df("path/to/file.json", source = "json")
schema <- structType(structField("name", "string"),
                      structField("info", "map<string,double>")
)df2 <- read.df(mapTypeJsonPath, "json", schema, multiline = TRUE)
df3 <- loadDF("data/test_table", "parquet", mergeSchema = "true")
stringSchema <- "name STRING, info MAP<STRING, DOUBLE>"
df4 <- read.df(mapTypeJsonPath, "json", stringSchema, multiline = TRUE)
## End(Not run)
```

---

**read.jdbc**

Create a SparkDataFrame representing the database table accessible via JDBC URL

**Description**

Additional JDBC database connection properties can be set (...)

**Usage**

```r
read.jdbc(
  url,  # JDBC database url of the form jdbc:subprotocol:subname
  tableName,  # the name of the table in the external database
  partitionColumn = NULL,  # the name of a column of numeric, date, or timestamp type that will be used for partitioning.
  lowerBound = NULL,  # the minimum value of partitionColumn used to decide partition stride
  upperBound = NULL,  # the maximum value of partitionColumn used to decide partition stride
  numPartitions = 0L,
  predicates = list(), ...
)
```

**Arguments**

- `url`  
  JDBC database url of the form jdbc:subprotocol:subname
- `tableName`  
  the name of the table in the external database
- `partitionColumn`  
  the name of a column of numeric, date, or timestamp type that will be used for partitioning.
- `lowerBound`  
  the minimum value of `partitionColumn` used to decide partition stride
- `upperBound`  
  the maximum value of `partitionColumn` used to decide partition stride
numPartitions: The number of partitions. This, along with lowerBound (inclusive), upperBound (exclusive), form partition strides for generated WHERE clause expressions used to split the column partitionColumn evenly. This defaults to SparkContext.defaultParallelism when unset.

predicates: A list of conditions in the where clause; each one defines one partition

... Additional JDBC database connection named properties.

Details:

Only one of partitionColumn or predicates should be set. Partitions of the table will be retrieved in parallel based on the numPartitions or by the predicates.

Don’t create too many partitions in parallel on a large cluster; otherwise Spark might crash your external database systems.

Value:

SparkDataFrame

Note:

read.jdbc since 2.0.0

Examples:

```r
## Not run:
sparkR.session()
jdbcUrl <- "jdbc:mysql://localhost:3306/databasename"
df <- read.jdbc(jdbcUrl, "table", predicates = list("field<=123"), user = "username")
df2 <- read.jdbc(jdbcUrl, "table2", partitionColumn = "index", lowerBound = 0,
                   upperBound = 10000, user = "username", password = "password")
```

## End(Not run)

Description:

Loads a JSON file, returning the result as a SparkDataFrame. By default, (JSON Lines text format or newline-delimited JSON) is supported. For JSON (one record per file), set a named property multiLine to TRUE. It goes through the entire dataset once to determine the schema.

Usage:

```r
## Default S3 method:
read.json(path, ...)

## Default S3 method:
jsonFile(path)
```
Arguments

path Path of file to read. A vector of multiple paths is allowed.

Value

SparkDataFrame

Note

read.json since 1.6.0
jsonFile since 1.4.0

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
df <- read.json(path, multiLine = TRUE)
df <- jsonFile(path)
## End(Not run)
```

---

**read.ml**

*Load a fitted MLlib model from the input path.*

Description

Load a fitted MLlib model from the input path.

Usage

```r
read.ml(path)
```

Arguments

path path of the model to read.

Value

A fitted MLlib model.

Note

read.ml since 2.0.0
read.orc

See Also

write.ml

Examples

```r
## Not run:
path <- "path/to/model"
model <- read.ml(path)

## End(Not run)
```

---

read.orc

Create a SparkDataFrame from an ORC file.

Description

Loads an ORC file, returning the result as a SparkDataFrame.

Usage

`read.orc(path, ...)`

Arguments

- `path`  
  Path of file to read.
- `...`  
  Additional external data source specific named properties.

Value

SparkDataFrame

Note

read.orc since 2.0.0
**read.parquet**

*Create a SparkDataFrame from a Parquet file.*

**Description**

Loads a Parquet file, returning the result as a SparkDataFrame.

**Usage**

```r
## Default S3 method:
read.parquet(path, ...)
```

```r
## Default S3 method:
parquetFile(...)  # read.parquet since 1.6.0
```

**Arguments**

- `path`: path of file to read. A vector of multiple paths is allowed.
- `...`: argument(s) passed to the method.

**Value**

SparkDataFrame

**Note**

read.parquet since 1.6.0
parquetFile since 1.4.0

---

**read.stream**

*Load a streaming SparkDataFrame*

**Description**

Returns the dataset in a data source as a SparkDataFrame

**Usage**

```r
read.stream(source = NULL, schema = NULL, ...)  # read.stream since 2.0.0
```

---


read.stream

Arguments

source  The name of external data source

schema  The data schema defined in structType or a DDL-formatted string, this is required for file-based streaming data source

...  additional external data source specific named options, for instance path for file-based streaming data source. timeZone to indicate a timezone to be used to parse timestamps in the JSON/CSV data sources or partition values; If it isn’t set, it uses the default value, session local timezone.

Details

The data source is specified by the source and a set of options(...). If source is not specified, the default data source configured by "spark.sql.sources.default" will be used.

Value

SparkDataFrame

Note

read.stream since 2.2.0

experimental

See Also

write.stream

Examples

```r
## Not run:
sparkR.session()
df <- read.stream("socket", host = "localhost", port = 9999)
q <- write.stream(df, "text", path = "/home/user/out", checkpointLocation = "/user/cp")
df <- read.stream("json", path = jsonDir, schema = schema, maxFilesPerTrigger = 1)
stringSchema <- "name STRING, info MAP<STRING, DOUBLE>"
df1 <- read.stream("json", path = jsonDir, schema = stringSchema, maxFilesPerTrigger = 1)
## End(Not run)
```
Create a SparkDataFrame from a text file.

Description

Loads text files and returns a SparkDataFrame whose schema starts with a string column named "value", and followed by partitioned columns if there are any.

Usage

```r
## Default S3 method:
read.text(path, ...)  
```

Arguments

- `path` Path of file to read. A vector of multiple paths is allowed.
- `...` additional external data source specific named properties.

Details

Each line in the text file is a new row in the resulting SparkDataFrame.

Value

SparkDataFrame

Note

read.text since 1.6.1

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.txt"
df <- read.text(path)

## End(Not run)
```
recoverPartitions

Recover all the partitions in the directory of a table and update the catalog.

**Description**

Recovers all the partitions in the directory of a table and update the catalog. The name should reference a partitioned table, and not a view.

**Usage**

```r
recoverPartitions(tableName)
```

**Arguments**

- `tableName` the qualified or unqualified name that designates a table. If no database identifier is provided, it refers to a table in the current database.

**Note**

since 2.2.0

**Examples**

```r
## Not run: sparkR.session() recoverPartitions("myTable") ## End(Not run)
```

refreshByPath

Invalidates and refreshes all the cached data and metadata for SparkDataFrame containing path

**Description**

Invalidates and refreshes all the cached data (and the associated metadata) for any SparkDataFrame that contains the given data source path. Path matching is by prefix, i.e. "/" would invalidate everything that is cached.

**Usage**

```r
refreshByPath(path)
```

**Arguments**

- `path` the path of the data source.
refreshTable

Description

Invalidates and refreshes all the cached data and metadata of the given table. For performance reasons, Spark SQL or the external data source library it uses might cache certain metadata about a table, such as the location of blocks. When those change outside of Spark SQL, users should call this function to invalidate the cache.

Usage

refreshTable(tableName)

Arguments

tableName the qualified or unqualified name that designates a table. If no database identifier is provided, it refers to a table in the current database.

Details

If this table is cached as an InMemoryRelation, drop the original cached version and make the new version cached lazily.

Note

since 2.2.0

Examples

## Not run:
sparkR.session()
refreshByPath("/path")

## End(Not run)
registerTempTable  (Deprecated) Register Temporary Table

Description

 Registers a SparkDataFrame as a Temporary Table in the SparkSession

Usage

 registerTempTable(x, tableName)

## S4 method for signature 'SparkDataFrame,character'
 registerTempTable(x, tableName)

Arguments

  x              A SparkDataFrame
  tableName      A character vector containing the name of the table

Note

 registerTempTable since 1.4.0

See Also

 createOrReplaceTempView

Examples

  ## Not run:
  sparkR.session()
  path <- "path/to/file.json"
  df <- read.json(path)
  registerTempTable(df, "json_df")
  new_df <- sql("SELECT * FROM json_df")

  ## End(Not run)
rename

Description

Rename an existing column in a SparkDataFrame.

Usage

rename(x, ...)

withColumnRenamed(x, existingCol, newCol)

## S4 method for signature 'SparkDataFrame,character,character'
withColumnRenamed(x, existingCol, newCol)

## S4 method for signature 'SparkDataFrame'
rename(x, ...)

Arguments

x  A SparkDataFrame

... A named pair of the form new_column_name = existing_column

existingCol  The name of the column you want to change.

newCol  The new column name.

Value

A SparkDataFrame with the column name changed.

Note

withColumnRenamed since 1.4.0

rename since 1.4.0

See Also

mutate

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method,broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except().explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), repartitionByRange(), repartition(),...
repartition

rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
newDF <- withColumnRenamed(df, "col1", "newCol1")
## End(Not run)

## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
newDF <- rename(df, col1 = df$newCol1)
## End(Not run)
```

repartition

Repartition

Description

The following options for repartition are possible:

- 1. Return a new SparkDataFrame that has exactly numPartitions.
- 2. Return a new SparkDataFrame hash partitioned by the given columns into numPartitions.
- 3. Return a new SparkDataFrame hash partitioned by the given column(s), using spark.sql.shuffle.partitions as number of partitions.

Usage

```r
repartition(x, ...)
```

## S4 method for signature 'SparkDataFrame'
```r
repartition(x, numPartitions = NULL, col = NULL, ...)
```

Arguments

- `x`: a SparkDataFrame.
- `...`: additional column(s) to be used in the partitioning.
- `numPartitions`: the number of partitions to use.
- `col`: the column by which the partitioning will be performed.
Note

repartition since 1.4.0

See Also

colnames, coltypes, createOrReplaceTempView, crossJoin, cube, dapplyCollect, describe, dim, distinct, dropDuplicates, dropna, drop, dtypes, exceptAll, except, explain, filter, first, gapplyCollect, gapply, getNumPartitions, group_by, head, hint, histogram, insertInto, intersectAll, intersect, isLocal, isStreaming, join, limit, localCheckpoint, merge, mutate, ncol, nrow, persist, printSchema, randomSplit, rbind, rename, repartitionByRange, rollup, sample, saveAsTable, schema, selectExpr, select, showDF, show, storageLevel, str, subset, summary, take, toJSON, unionByName, union, unpersist, withColumn, withWatermark, with, write.df, write.jdbc, write.json, write.orc, write.parquet, write.stream, write.text

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
newDF <- repartition(df, 2L)
newDF <- repartition(df, numPartitions = 2L)
newDF <- repartition(df, col = df$"col1", df$"col2")
newDF <- repartition(df, 3L, col = df$"col1", df$"col2")
```

### Description

The following options for repartition by range are possible:

- 1. Return a new SparkDataFrame range partitioned by the given columns into `numPartitions`.
- 2. Return a new SparkDataFrame range partitioned by the given column(s), using `spark.sql.shuffle.partitions` as number of partitions.

### Usage

```
repartitionByRange(x, ...)  
```

## S4 method for signature 'SparkDataFrame'
```
repartitionByRange(x, numPartitions = NULL, col = NULL, ...)  
```
Arguments

x  a SparkDataFrame.

...  additional column(s) to be used in the range partitioning.

numPartitions  the number of partitions to use.

col  the column by which the range partitioning will be performed.

Note

repartitionByRange since 2.4.0

See Also

repartition, coalesce

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
newDF <- repartitionByRange(df, col = df$col1, df$col2)
newDF <- repartitionByRange(df, 3L, col = df$col1, df$col2)
## End(Not run)
```

Description

Create a multi-dimensional rollup for the SparkDataFrame using the specified columns.
Usage

rollup(x, ...)

## S4 method for signature 'SparkDataFrame'
rollup(x, ...)

Arguments

x:
a SparkDataFrame.

...:
character name(s) or Column(s) to group on.

Details

If grouping expression is missing rollup creates a single global aggregate and is equivalent to direct application of agg.

Value

A GroupedData.

Note

rollup since 2.3.0

See Also

agg, cube, groupBy

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

## Not run:
df <- createDataFrame(mtcars)
mean(rollup(df, "cyl", "gear", "am"), "mpg")

# Following calls are equivalent
agg(rollup(df), mean(df$mpg))
agg(df, mean(df$mpg))
## rowsBetween

### Description

Defines the frame boundaries, from start (inclusive) to end (inclusive).

### Usage

```r
rowsBetween(x, start, end)
```

```r
## S4 method for signature 'WindowSpec,numeric,numeric'
rowsBetween(x, start, end)
```

### Arguments

- `x` a WindowSpec
- `start` boundary start, inclusive. The frame is unbounded if this is the minimum long value.
- `end` boundary end, inclusive. The frame is unbounded if this is the maximum long value.

### Details

Both `start` and `end` are relative positions from the current row. For example, "0" means "current row", while "-1" means the row before the current row, and "5" means the fifth row after the current row.

### Value

a WindowSpec

### Note

`rowsBetween` since 2.0.0

### See Also

Other windowspec_method: `orderBy()`, `partitionBy()`, `rangeBetween()`

### Examples

```r
## Not run:
rowsBetween(ws, 0, 3)
```

```r
## End(Not run)
```
Description

Return a sampled subset of this SparkDataFrame using a random seed. Note: this is not guaranteed to provide exactly the fraction specified of the total count of the given SparkDataFrame.

Usage

```r
sample(x, withReplacement = FALSE, fraction, seed)
```

```r
sample_frac(x, withReplacement = FALSE, fraction, seed)
```

## S4 method for signature 'SparkDataFrame'

```r
sample(x, withReplacement = FALSE, fraction, seed)
```

## S4 method for signature 'SparkDataFrame'

```r
sample_frac(x, withReplacement = FALSE, fraction, seed)
```

Arguments

- **x**: A SparkDataFrame
- **withReplacement**: Sampling with replacement or not
- **fraction**: The (rough) sample target fraction
- **seed**: Randomness seed value. Default is a random seed.

Note

- sample since 1.4.0
- sample_frac since 1.4.0

See Also

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach()`, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `group_by()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isLocal()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `ncol()`, `nrow()`, `persist()`, `printSchema()`, `randomSplit()`, `rbind()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()`, `saveAsTable()`, `schema()`, `selectExpr()`, `select()`, `showDF()`, `show()`, `storageLevel()`, `str()`, `subset()`, `summary()`, `take()`, `toJSON()`, `unionByName()`, `union()`, ` unpersist()`, `withColumn()`, `withWatermark()`, `with()`, `write.df()`, `write.jdbc()`, `write.json()`, `write.orc()`, `write.parquet()`, `write.stream()`, `write.text()`
sampleBy

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
collect(sample(df, fraction = 0.5))
collect(sample(df, FALSE, 0.5))
collect(sample(df, TRUE, 0.5, seed = 3))

## End(Not run)
```

---

**sampleBy**  
*Returns a stratified sample without replacement*

### Description

Returns a stratified sample without replacement based on the fraction given on each stratum.

### Usage

```r
sampleBy(x, col, fractions, seed)
```

---

**Arguments**

- `x`  
  A SparkDataFrame

- `col`  
  column that defines strata

- `fractions`  
  A named list giving sampling fraction for each stratum. If a stratum is not specified, we treat its fraction as zero.

- `seed`  
  random seed

### Value

A new SparkDataFrame that represents the stratified sample

### Note

sampleBy since 1.6.0

### See Also

Other stat functions: `approxQuantile()`, `corr()`, `cov()`, `crosstab()`, `freqItems()`
## saveAsTable

### Description

The data source is specified by the `source` and a set of options (...). If `source` is not specified, the default data source configured by `spark.sql.sources.default` will be used.

### Usage

```r
saveAsTable(df, tableName, source = NULL, mode = "error", ...)
```

### Arguments

- `df`: a SparkDataFrame.
- `tableName`: a name for the table.
- `source`: a name for external data source.
- `mode`: one of 'append', 'overwrite', 'error', 'errorifexists', 'ignore' save mode (it is 'error' by default)
- `...`: additional option(s) passed to the method.

### Details

Additionally, `mode` is used to specify the behavior of the save operation when data already exists in the data source. There are four modes:

- 'append': Contents of this SparkDataFrame are expected to be appended to existing data.
- 'overwrite': Existing data is expected to be overwritten by the contents of this SparkDataFrame.
- 'error' or 'errorifexists': An exception is expected to be thrown.
- 'ignore': The save operation is expected to not save the contents of the SparkDataFrame and to not change the existing data.

### Note

`saveAsTable` since 1.4.0

---

## Examples

```r
## Not run:
df <- read.json("/path/to/file.json")
sample <- sampleBy(df, "key", fractions, 36)

## End(Not run)
```
See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(),attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(). explain(), filter(), first(). gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(). hint(). histogram(). insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(). printSchema(), randomSplit(). rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(). schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(). take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()
See Also

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach()`, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `columtypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `group_by()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isLocal()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `nrow()`, `persist()`, `printSchema()`, `randomSplit()`, `rbind()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()`, `sample()`, `saveAsTable()`, `selectExpr()`, `select()`, `showDF()`, `show()`, `storageLevel()`, `str()`, `subset()`, `summary()`, `take()`, `toJSON()`, `unionByName()`, `union()`, `unpersist()`, `withColumn()`, `withWatermark()`, `with()`, `write.df()`, `write.jdbc()`, `write.json()`, `write.orc()`, `write.parquet()`, `write.stream()`, `write.text()`

Examples

```r
## Not run:
sp1R.session()
path <- "path/to/file.json"
df <- read.json(path)
dfSchema <- schema(df)
## End(Not run)
```

<table>
<thead>
<tr>
<th>select</th>
<th>Select</th>
</tr>
</thead>
</table>

Description

Selects a set of columns with names or Column expressions.

Usage

```r
select(x, col, ...)
```

```r
## S4 method for signature 'SparkDataFrame'
x$name
```

```r
## S4 replacement method for signature 'SparkDataFrame'
x$name <- value
```

```r
## S4 method for signature 'SparkDataFrame,character'
select(x, col, ...)
```

```r
## S4 method for signature 'SparkDataFrame,Column'
select(x, col, ...)
```

```r
## S4 method for signature 'SparkDataFrame,list'
select(x, col)
```
select

Arguments

x  a SparkDataFrame.

col  a list of columns or single Column or name.

... additional column(s) if only one column is specified in col. If more than one column is assigned in col, ... should be left empty.

name  name of a Column (without being wrapped by "").

value  a Column or an atomic vector in the length of 1 as literal value, or NULL. If NULL, the specified Column is dropped.

Value

A new SparkDataFrame with selected columns.

Note

$ since 1.4.0
$<- since 1.4.0
select(SparkDataFrame, character) since 1.4.0
select(SparkDataFrame, Column) since 1.4.0
select(SparkDataFrame, list) since 1.4.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Other subsetting functions: filter(), subset()

Examples

## Not run:
select(df, "x")
select(df, "col1", "col2")
select(df, df$name, df$age + 1)
select(df, c("col1", "col2"))
select(df, list(df$name, df$age + 1))
# Similar to R data frames columns can also be selected using $
df[,df$age]
selectExpr

Description

Select from a SparkDataFrame using a set of SQL expressions.

Usage

```r
selectExpr(x, expr, ...)
```

Arguments

- `x` A SparkDataFrame to be selected from.
- `expr` A string containing a SQL expression
- `...` Additional expressions

Value

A SparkDataFrame

Note

selectExpr since 1.4.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()
Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
selectExpr(df, "col1", "(col2 * 5) as newCol")

## End(Not run)
```

Description

Set the directory under which SparkDataFrame are going to be checkpointed. The directory must be an HDFS path if running on a cluster.

Usage

```r
setCheckpointDir(directory)
```

Arguments

- `directory` Directory path to checkpoint to

Note

`setCheckpointDir` since 2.2.0

See Also

- `checkpoint`

Examples

```r
## Not run:
setCheckpointDir("/checkpoint")

## End(Not run)
```
setCurrentDatabase  

Sets the current default database.

Usage

setCurrentDatabase(databaseName)

Arguments

databaseName  
name of the database

Note

since 2.2.0

Examples

## Not run:
sparkR.session()
setCurrentDatabase("default")

## End(Not run)

setJobDescription  

Set a human readable description of the current job.

Description

Set a description that is shown as a job description in UI.

Usage

setJobDescription(value)

Arguments

value  
The job description of the current job.

Note

setJobDescription since 2.3.0
setJobGroup

Examples

```r
## Not run:
setJobDescription("This is an example job.")

## End(Not run)
```

| setJobGroup | Assigns a group ID to all the jobs started by this thread until the group ID is set to a different value or cleared. |

Description

Assigns a group ID to all the jobs started by this thread until the group ID is set to a different value or cleared.

Usage

```r
## Default S3 method:
setJobGroup(groupId, description, interruptOnCancel)
```

Arguments

- `groupId` the ID to be assigned to job groups.
- `description` description for the job group ID.
- `interruptOnCancel` flag to indicate if the job is interrupted on job cancellation.

Note

setJobGroup since 1.5.0

Examples

```r
## Not run:
sparkR.session()
setJobGroup("myJobGroup", "My job group description", TRUE)

## End(Not run)
```
**setLocalProperty**  
*Set a local property that affects jobs submitted from this thread, such as the Spark fair scheduler pool.*

**Description**
Set a local property that affects jobs submitted from this thread, such as the Spark fair scheduler pool.

**Usage**
```
setLocalProperty(key, value)
```

**Arguments**
- **key**  
The key for a local property.
- **value**  
The value for a local property.

**Note**
- `setLocalProperty` since 2.3.0

**Examples**
```
## Not run:
setLocalProperty("spark.scheduler.pool", "poolA")
## End(Not run)
```

---

**setLogLevel**  
*Set new log level*

**Description**
Set new log level: "ALL", "DEBUG", "ERROR", "FATAL", "INFO", "OFF", "TRACE", "WARN"

**Usage**
```
setLogLevel(level)
```

**Arguments**
- **level**  
New log level

**Note**
- `setLogLevel` since 2.0.0
Examples

```r
## Not run:
setLogLevel("ERROR")

## End(Not run)
```

Description

Print class and type information of a Spark object.

Usage

```r
## S4 method for signature 'Column'
show(object)

## S4 method for signature 'GroupedData'
show(object)

## S4 method for signature 'SparkDataFrame'
show(object)

## S4 method for signature 'WindowSpec'
show(object)

## S4 method for signature 'StreamingQuery'
show(object)
```

Arguments

- `object` a Spark object. Can be a SparkDataFrame, Column, GroupedData, WindowSpec.

Note

- `show(Column)` since 1.4.0
- `show(GroupedData)` since 1.4.0
- `show(SparkDataFrame)` since 1.4.0
- `show(WindowSpec)` since 2.0.0
- `show(StreamingQuery)` since 2.2.0
See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
p <- "path/to/file.json"
df <- read.json(p)
show(df)
## End(Not run)
```

Description

Print the first numRows rows of a SparkDataFrame

Usage

```r
showDF(x, ...)
```

Arguments

- `x`: a SparkDataFrame.
- `...`: further arguments to be passed to or from other methods.
- `numRows`: the number of rows to print. Defaults to 20.
- `truncate`: whether truncate long strings. If TRUE, strings more than 20 characters will be truncated. However, if set greater than zero, truncates strings longer than truncate characters and all cells will be aligned right.
- `vertical`: whether print output rows vertically (one line per column value).
spark.addFile

Add a file or directory to be downloaded with this Spark job on every node.

Description

The path passed can be either a local file, a file in HDFS (or other Hadoop-supported filesystems), or an HTTP, HTTPS or FTP URI. To access the file in Spark jobs, use spark.getSparkFiles(fileName) to find its download location.

Usage

spark.addFile(path, recursive = FALSE)

Arguments

  path          The path of the file to be added
  recursive     Whether to add files recursively from the path. Default is FALSE.
spark.als

**Details**

A directory can be given if the recursive option is set to true. Currently directories are only supported for Hadoop-supported filesystems. Refer Hadoop-supported filesystems at [https://cwiki.apache.org/confluence/display/HADOOP2/HCFS](https://cwiki.apache.org/confluence/display/HADOOP2/HCFS).

Note: A path can be added only once. Subsequent additions of the same path are ignored.

**Note**

spark.addFile since 2.1.0

**Examples**

```r
## Not run:
spark.addFile("~/myfile")

## End(Not run)
```

---

**Description**

spark.als learns latent factors in collaborative filtering via alternating least squares. Users can call summary to obtain fitted latent factors, predict to make predictions on new data, and write.ml/read.ml to save/load fitted models.

**Usage**

```r
spark.als(data, ...)

## S4 method for signature 'SparkDataFrame'
spark.als(
    data,
    ratingCol = "rating",
    userCol = "user",
    itemCol = "item",
    rank = 10,
    regParam = 0.1,
    maxIter = 10,
    nonnegative = FALSE,
    implicitPrefs = FALSE,
    alpha = 1,
    numUserBlocks = 10,
    numItemBlocks = 10,
    checkpointInterval = 10,
    seed = 0
)
```
## S4 method for signature 'ALSModel'
summary(object)

## S4 method for signature 'ALSModel'
predict(object, newData)

## S4 method for signature 'ALSModel,character'
write.ml(object, path, overwrite = FALSE)

### Arguments

data | a SparkDataFrame for training.
--- | additional argument(s) passed to the method.
ratingCol | column name for ratings.
userCol | column name for user ids. Ids must be (or can be coerced into) integers.
itemCol | column name for item ids. Ids must be (or can be coerced into) integers.
rank | rank of the matrix factorization (> 0).
regParam | regularization parameter (>= 0).
maxIter | maximum number of iterations (>= 0).
nonnegative | logical value indicating whether to apply nonnegativity constraints.
implicitPrefs | logical value indicating whether to use implicit preference.
alpha | alpha parameter in the implicit preference formulation (>= 0).
numUser Blocks | number of user blocks used to parallelize computation (> 0).
numItemBlocks | number of item blocks used to parallelize computation (> 0).
checkpointInterval | number of checkpoint intervals (>= 1) or disable checkpoint (-1). Note: this setting will be ignored if the checkpoint directory is not set.
seed | integer seed for random number generation.
object | a fitted ALS model.
newData | a SparkDataFrame for testing.
path | the directory where the model is saved.
overwrite | logical value indicating whether to overwrite if the output path already exists. Default is FALSE which means throw exception if the output path exists.

### Details

For more details, see **MLlib: Collaborative Filtering**.
spark.als returns a fitted ALS model.

summary returns summary information of the fitted model, which is a list. The list includes user (the names of the user column), item (the item column), rating (the rating column), userFactors (the estimated user factors), itemFactors (the estimated item factors), and rank (rank of the matrix factorization model).

predict returns a SparkDataFrame containing predicted values.

Note

spark.als since 2.1.0
summary(ALSModel) since 2.1.0
predict(ALSModel) since 2.1.0
write.ml(ALSModel, character) since 2.1.0

See Also

read.ml

Examples

```r
## Not run:
ratings <- list(list(0, 0, 4.0), list(0, 1, 2.0), list(1, 1, 3.0), list(1, 2, 4.0),
                 list(2, 1, 1.0), list(2, 2, 5.0))
df <- createDataFrame(ratings, c("user", "item", "rating"))
model <- spark.als(df, "rating", "user", "item")

# extract latent factors
stats <- summary(model)
userFactors <- stats$userFactors
itemFactors <- stats$itemFactors

# make predictions
predicted <- predict(model, df)
showDF(predicted)

# save and load the model
path <- "path/to/model"
write.ml(model, path)
savedModel <- read.ml(path)
summary(savedModel)

# set other arguments
modelS <- spark.als(df, "rating", "user", "item", rank = 20,
                    regParam = 0.1, nonnegative = TRUE)
statsS <- summary(modelS)

## End(Not run)
```
spark.bisectingKmeans  Bisecting K-Means Clustering Model

Description

Fits a bisecting k-means clustering model against a SparkDataFrame. Users can call `summary` to print a summary of the fitted model, `predict` to make predictions on new data, and `write.ml/read.ml` to save/load fitted models.

Get fitted result from a bisecting k-means model. Note: A saved-loaded model does not support this method.

Usage

```r
spark.bisectingKmeans(data, formula, ...)  
## S4 method for signature 'SparkDataFrame,formula'
spark.bisectingKmeans(
  data,
  formula,
  k = 4,
  maxIter = 20,
  seed = NULL,
  minDivisibleClusterSize = 1
)
```

```r
## S4 method for signature 'BisectingKMeansModel'
summary(object)
```

```r
## S4 method for signature 'BisectingKMeansModel'
predict(object, newData)
```

```r
## S4 method for signature 'BisectingKMeansModel'
fitted(object, method = c("centers", "classes"))
```

```r
## S4 method for signature 'BisectingKMeansModel,character'
write.ml(object, path, overwrite = FALSE)
```

Arguments

- **data**: a SparkDataFrame for training.
- **formula**: a symbolic description of the model to be fitted. Currently only a few formula operators are supported, including `~`, `;`, `:`, `+`, and `-`. Note that the response variable of formula is empty in `spark.bisectingKmeans`.
- **...**: additional argument(s) passed to the method.
- **k**: the desired number of leaf clusters. Must be > 1. The actual number could be smaller if there are no divisible leaf clusters.
maxIter maximum iteration number.
seed the random seed.
minDivisibleClusterSize
The minimum number of points (if greater than or equal to 1.0) or the minimum proportion of points (if less than 1.0) of a divisible cluster. Note that it is an expert parameter. The default value should be good enough for most cases.

object a fitted bisecting k-means model.
newData a SparkDataFrame for testing.
method type of fitted results, "centers" for cluster centers or "classes" for assigned classes.
path the directory where the model is saved.
overwrite overwrites or not if the output path already exists. Default is FALSE which means throw exception if the output path exists.

Value

spark.bisectingKmeans returns a fitted bisecting k-means model.
summary returns summary information of the fitted model, which is a list. The list includes the model's k (number of cluster centers), coefficients (model cluster centers), size (number of data points in each cluster), cluster (cluster centers of the transformed data; cluster is NULL if is.loaded is TRUE), and is.loaded (whether the model is loaded from a saved file).
predict returns the predicted values based on a bisecting k-means model.
fitted returns a SparkDataFrame containing fitted values.

Note

spark.bisectingKmeans since 2.2.0
summary(BisectingKMeansModel) since 2.2.0
predict(BisectingKMeansModel) since 2.2.0
fitted since 2.2.0
write.ml(BisectingKMeansModel, character) since 2.2.0

See Also

predict, read.ml, write.ml

Examples

```r
## Not run:
sparkR.session()
t <- as.data.frame(Titanic)
df <- createDataFrame(t)
model <- spark.bisectingKmeans(df, Class ~ Survived, k = 4)
summary(model)

# get fitted result from a bisecting k-means model
```
fitted.model <- fitted(model, "centers")
showDF(fitted.model)

# fitted values on training data
fitted <- predict(model, df)
head(select(fitted, "Class", "prediction"))

# save fitted model to input path
path <- "path/to/model"
write.ml(model, path)

# can also read back the saved model and print
savedModel <- read.ml(path)
summary(savedModel)

## End(Not run)

---

**spark.decisionTree**  
*Decision Tree Model for Regression and Classification*

**Description**

`spark.decisionTree` fits a Decision Tree Regression model or Classification model on a SparkDataFrame. Users can call `summary` to get a summary of the fitted Decision Tree model, `predict` to make predictions on new data, and `write.ml`/`read.ml` to save/load fitted models. For more details, see Decision Tree Regression and Decision Tree Classification.

**Usage**

```r
spark.decisionTree(data, formula, ...)  
## S4 method for signature 'SparkDataFrame,formula'
spark.decisionTree(  
data,  
formula,  
type = c("regression", "classification"),  
maxDepth = 5,  
maxBins = 32,  
impurity = NULL,  
seed = NULL,  
minInstancesPerNode = 1,  
minInfoGain = 0,  
checkpointInterval = 10,  
maxMemoryInMB = 256,  
cacheNodeIds = FALSE,  
handleInvalid = c("error", "keep", "skip")
)
```
## S4 method for signature 'DecisionTreeRegressionModel'
summary(object)

## S3 method for class 'summary.DecisionTreeRegressionModel'
print(x, ...)

## S4 method for signature 'DecisionTreeClassificationModel'
summary(object)

## S3 method for class 'summary.DecisionTreeClassificationModel'
print(x, ...)

## S4 method for signature 'DecisionTreeRegressionModel'
predict(object, newData)

## S4 method for signature 'DecisionTreeClassificationModel'
predict(object, newData)

## S4 method for signature 'DecisionTreeRegressionModel,character'
write.ml(object, path, overwrite = FALSE)

## S4 method for signature 'DecisionTreeClassificationModel,character'
write.ml(object, path, overwrite = FALSE)

### Arguments

- **data**: a SparkDataFrame for training.
- **formula**: a symbolic description of the model to be fitted. Currently only a few formula operators are supported, including `~`, `:`, `+`, and `-`.
- **...**: additional arguments passed to the method.
- **type**: type of model, one of "regression" or "classification", to fit
- **maxDepth**: Maximum depth of the tree (>= 0).
- **maxBins**: 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. Must be >= 2 and >= number of categories in any categorical feature.
- **impurity**: Criterion used for information gain calculation. For regression, must be "variance". For classification, must be one of "entropy" and "gini", default is "gini".
- **seed**: integer seed for random number generation.
- **minInstancesPerNode**: Minimum number of instances each child must have after split.
- **minInfoGain**: Minimum information gain for a split to be considered at a tree node.
- **checkpointInterval**: Param for set checkpoint interval (>= 1) or disable checkpoint (-1). Note: this setting will be ignored if the checkpoint directory is not set.
- **maxMemoryInMB**: Maximum memory in MB allocated to histogram aggregation.
cacheNodeIds: 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. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.

handleInvalid: How to handle invalid data (unseen labels or NULL values) in features and label column of string type in classification model. Supported options: "skip" (filter out rows with invalid data), "error" (throw an error), "keep" (put invalid data in a special additional bucket, at index numLabels). Default is "error".

object: A fitted Decision Tree regression model or classification model.

x: summary object of Decision Tree regression model or classification model returned by summary.

newData: a SparkDataFrame for testing.

path: The directory where the model is saved.

overwrite: Overwrites or not if the output path already exists. Default is FALSE which means throw exception if the output path exists.

Value

spark.decisionTree returns a fitted Decision Tree model.
summary returns summary information of the fitted model, which is a list. The list of components includes formula (formula), numFeatures (number of features), features (list of features), featureImportances (feature importances), and maxDepth (max depth of trees).
predict returns a SparkDataFrame containing predicted labeled in a column named "prediction".

Note

spark.decisionTree since 2.3.0
summary(DecisionTreeRegressionModel) since 2.3.0
print.summary.DecisionTreeRegressionModel since 2.3.0
summary(DecisionTreeClassificationModel) since 2.3.0
print.summary.DecisionTreeClassificationModel since 2.3.0
predict(DecisionTreeRegressionModel) since 2.3.0
predict(DecisionTreeClassificationModel) since 2.3.0
write.ml(DecisionTreeRegressionModel, character) since 2.3.0
write.ml(DecisionTreeClassificationModel, character) since 2.3.0

Examples

```r
## Not run:
# fit a Decision Tree Regression Model
df <- createDataFrame(longley)
model <- spark.decisionTree(df, Employed ~ ., type = "regression", maxDepth = 5, maxBins = 16)

# get the summary of the model
summary(model)
```
spark.fpGrowth

# make predictions
predictions <- predict(model, df)

# save and load the model
path <- "path/to/model"
write.ml(model, path)
savedModel <- read.ml(path)
summary(savedModel)

# fit a Decision Tree Classification Model
t <- as.data.frame(Titanic)
df <- createDataFrame(t)
model <- spark.decisionTree(df, Survived ~ Freq + Age, "classification")

## End(Not run)

---

**Description**

A parallel FP-growth algorithm to mine frequent itemsets. `spark.fpGrowth` fits a FP-growth model on a SparkDataFrame. Users can `spark.freqItemsets` to get frequent itemsets, `spark.associationRules` to get association rules, `predict` to make predictions on new data based on generated association rules, and `write.ml/read.ml` to save/load fitted models. For more details, see FP-growth.

**Usage**

```r
spark.fpGrowth(data, ...)  
spark.freqItemsets(object)  
spark.associationRules(object)
```

```r
## S4 method for signature 'SparkDataFrame'
spark.fpGrowth(  
data,  
minSupport = 0.3,  
minConfidence = 0.8,  
itemsCol = "items",  
numPartitions = NULL
)
```

```r
## S4 method for signature 'FPGrowthModel'
spark.freqItemsets(object)

## S4 method for signature 'FPGrowthModel'
```
spark.fpGrowth

spark.associationRules(object)

## S4 method for signature 'FP_growth'
predict(object, newData)

## S4 method for signature 'FP_growth,character'
write.ml(object, path, overwrite = FALSE)

Arguments

data A SparkDataFrame for training.

object a fitted FP_growth model.

minSupport Minimal support level.

minConfidence Minimal confidence level.

itemsCol Features column name.

numPartitions Number of partitions used for fitting.

newData a SparkDataFrame for testing.

path the directory where the model is saved.

overwrite logical value indicating whether to overwrite if the output path already exists. Default is FALSE which means throw exception if the output path exists.

Value

spark.fpGrowth returns a fitted FP_growth model.

A SparkDataFrame with frequent itemsets. The SparkDataFrame contains two columns: items (an array of the same type as the input column) and freq (frequency of the itemset).

A SparkDataFrame with association rules. The SparkDataFrame contains four columns: antecedent (an array of the same type as the input column), consequent (an array of the same type as the input column), condfidence (confidence for the rule) and lift (lift for the rule)

predict returns a SparkDataFrame containing predicted values.

Note

spark.fpGrowth since 2.2.0
spark.freqItemsets(FP_growth) since 2.2.0
spark.associationRules(FP_growth) since 2.2.0
predict(FP_growth) since 2.2.0
write.ml(FP_growth, character) since 2.2.0

See Also

read.ml
spark.gaussianMixture

Multivariate Gaussian Mixture Model (GMM)

Description

Fits multivariate gaussian mixture model against a SparkDataFrame, similarly to R’s mvnormalmixEM(). Users can call summary to print a summary of the fitted model, predict to make predictions on new data, and write.ml/read.ml to save/load fitted models.

Usage

spark.gaussianMixture(data, formula, ...)

## S4 method for signature 'SparkDataFrame,formula'
spark.gaussianMixture

spark.gaussianMixture(data, formula, k = 2, maxIter = 100, tol = 0.01)

## S4 method for signature 'GaussianMixtureModel'
summary(object)

## S4 method for signature 'GaussianMixtureModel'
predict(object, newData)

## S4 method for signature 'GaussianMixtureModel,character'
write.ml(object, path, overwrite = FALSE)

Arguments

data a SparkDataFrame for training.
formulaa symbolic description of the model to be fitted. Currently only a few formula
operators are supported, including ‘‘’, ‘.’, ‘:’, ‘+’, and ‘-’. Note that the response
variable of formula is empty in spark.gaussianMixture.

... additional arguments passed to the method.
k number of independent Gaussians in the mixture model.
maxIter maximum iteration number.
tol the convergence tolerance.
object a fitted gaussian mixture model.
newData a SparkDataFrame for testing.
path the directory where the model is saved.
overwrite overwrites or not if the output path already exists. Default is FALSE which
means throw exception if the output path exists.

Value

spark.gaussianMixture returns a fitted multivariate gaussian mixture model.
summary returns summary of the fitted model, which is a list. The list includes the model's lambda
(lambda), mu (mu), sigma (sigma), loglik (loglik), and posterior (posterior).
predict returns a SparkDataFrame containing predicted labels in a column named "prediction".

Note

spark.gaussianMixture since 2.1.0
summary(GaussianMixtureModel) since 2.1.0
predict(GaussianMixtureModel) since 2.1.0
write.ml(GaussianMixtureModel, character) since 2.1.0

See Also

mixtools: https://cran.r-project.org/package=mixtools
predict, read.ml, write.ml
Examples

```r
## Not run:
sparkR.session()
library(mvtnorm)
set.seed(100)
a <- rmvnorm(4, c(0, 0))
b <- rmvnorm(6, c(3, 4))
data <- rbind(a, b)
df <- createDataFrame(as.data.frame(data))
model <- spark.gaussianMixture(df, ~ V1 + V2, k = 2)
summary(model)

# fitted values on training data
fitted <- predict(model, df)
head(select(fitted, "V1", "prediction"))

# save fitted model to input path
path <- "path/to/model"
write.ml(model, path)

# can also read back the saved model and print
savedModel <- read.ml(path)
summary(savedModel)

## End(Not run)
```

---

**spark.gbt**  
*Gradient Boosted Tree Model for Regression and Classification*

**Description**

`spark.gbt` fits a Gradient Boosted Tree Regression model or Classification model on a Spark DataFrame. Users can call `summary` to get a summary of the fitted Gradient Boosted Tree model, `predict` to make predictions on new data, and `write.ml/read.ml` to save/load fitted models. For more details, see [GBT Regression](#) and [GBT Classification](#)

**Usage**

```r
spark.gbt(data, formula, ...)
```

## S4 method for signature 'SparkDataFrame,formula'

```r
spark.gbt(
data,
formula,
type = c("regression", "classification"),
maxDepth = 5,
maxBins = 32,
maxIter = 20,
```

```r
```
stepSize = 0.1,
lossType = NULL,
seed = NULL,
subsamplingRate = 1,
minInstancesPerNode = 1,
minInfoGain = 0,
checkpointInterval = 10,
maxMemoryInMB = 256,
cacheNodeIds = FALSE,
handleInvalid = c("error", "keep", "skip")
)

## S4 method for signature 'GBTRegressionModel'
summary(object)

## S3 method for class 'summary.GBTRegressionModel'
print(x, ...)

## S4 method for signature 'GBTClassificationModel'
summary(object)

## S3 method for class 'summary.GBTClassificationModel'
print(x, ...)

## S4 method for signature 'GBTRegressionModel'
predict(object, newData)

## S4 method for signature 'GBTClassificationModel'
predict(object, newData)

## S4 method for signature 'GBTRegressionModel,character'
write.ml(object, path, overwrite = FALSE)

## S4 method for signature 'GBTClassificationModel,character'
write.ml(object, path, overwrite = FALSE)

### Arguments

- **data**: a SparkDataFrame for training.
- **formula**: a symbolic description of the model to be fitted. Currently only a few formula operators are supported, including `~`, `+`, `+`, and `-`.
- **...**: additional arguments passed to the method.
- **type**: type of model, one of "regression" or "classification", to fit
- **maxDepth**: Maximum depth of the tree (>= 0).
- **maxBins**: 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. Must be >= 2 and >= number of categories in any categorical feature.
maxIter Param for maximum number of iterations (>= 0).
stepSize Param for Step size to be used for each iteration of optimization.
lossType Loss function which GBT tries to minimize. For classification, must be "logistic". For regression, must be one of "squared" (L2) and "absolute" (L1), default is "squared".
seed integer seed for random number generation.
subsamplingRate Fraction of the training data used for learning each decision tree, in range (0, 1].
minInstancesPerNode Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.
minInfoGain Minimum information gain for a split to be considered at a tree node.
checkpointInterval Param for set checkpoint interval (>= 1) or disable checkpoint (-1). Note: this setting will be ignored if the checkpoint directory is not set.
maxMemoryInMB Maximum memory in MB allocated to histogram aggregation.
cacheNodeIds 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. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.
handleInvalid How to handle invalid data (unseen labels or NULL values) in features and label column of string type in classification model. Supported options: "skip" (filter out rows with invalid data), "error" (throw an error), "keep" (put invalid data in a special additional bucket, at index numLabels). Default is "error".
object A fitted Gradient Boosted Tree regression model or classification model.
x summary object of Gradient Boosted Tree regression model or classification model returned by summary.
newData a SparkDataFrame for testing.
path The directory where the model is saved.
overwrite Overwrites or not if the output path already exists. Default is FALSE which means throw exception if the output path exists.

Value

spark.gbt returns a fitted Gradient Boosted Tree model.
summary returns summary information of the fitted model, which is a list. The list of components includes formula (formula), numFeatures (number of features), features (list of features), featureImportances (feature importances), maxDepth (max depth of trees), numTrees (number of trees), and treeWeights (tree weights).
predict returns a SparkDataFrame containing predicted labeled in a column named "prediction".
Note

spark.gbt since 2.1.0
summary(GBTRegressionModel) since 2.1.0
print.summary.GBTRegressionModel since 2.1.0
summary(GBTClassificationModel) since 2.1.0
print.summary.GBTClassificationModel since 2.1.0
predict(GBTRegressionModel) since 2.1.0
predict(GBTClassificationModel) since 2.1.0
write.ml(GBTRegressionModel, character) since 2.1.0
write.ml(GBTClassificationModel, character) since 2.1.0

Examples

## Not run:
# fit a Gradient Boosted Tree Regression Model
df <- createDataFrame(longley)
model <- spark.gbt(df, Employed ~ ., type = "regression", maxDepth = 5, maxBins = 16)

# get the summary of the model
summary(model)

# make predictions
predictions <- predict(model, df)

# save and load the model
path <- "path/to/model"
write.ml(model, path)
savedModel <- read.ml(path)
summary(savedModel)

# fit a Gradient Boosted Tree Classification Model
# label must be binary - Only binary classification is supported for GBT.
t <- as.data.frame(Titanic)
df <- createDataFrame(t)
model <- spark.gbt(df, Survived ~ Age + Freq, "classification")

# numeric label is also supported
t2 <- as.data.frame(Titanic)
t2$NumericGender <- ifelse(t2$Sex == "Male", 0, 1)
df <- createDataFrame(t2)
model <- spark.gbt(df, NumericGender ~ ., type = "classification")

## End(Not run)
spark.getSparkFiles  Get the absolute path of a file added through spark.addFile.

**Description**

Get the absolute path of a file added through spark.addFile.

**Usage**

```
spark.getSparkFiles(fileName)
```

**Arguments**

- **fileName** The name of the file added through spark.addFile

**Value**

the absolute path of a file added through spark.addFile.

**Note**

spark.getSparkFiles since 2.1.0

**Examples**

```
## Not run:
spark.getSparkFiles("myfile")

## End(Not run)
```

_____________________________

spark.getSparkFilesRootDirectory  Get the root directory that contains files added through spark.addFile.

**Description**

Get the root directory that contains files added through spark.addFile.

**Usage**

```
spark.getSparkFilesRootDirectory()
```

**Value**

the root directory that contains files added through spark.addFile
**spark.glm**

**Note**

spark.getSparkFilesRootDirectory since 2.1.0

**Examples**

```r
## Not run:
spark.getSparkFilesRootDirectory()

## End(Not run)
```

---

**spark.glm**

*Generalized Linear Models*

**Description**

Fits generalized linear model against a SparkDataFrame. Users can call `summary` to print a summary of the fitted model, `predict` to make predictions on new data, and `write.ml/read.ml` to save/load fitted models.

**Usage**

```r
spark.glm(data, formula, ...)
```

```r
## S4 method for signature 'SparkDataFrame,formula'
spark.glm(
  data,
  formula,
  family = gaussian,
  tol = 1e-06,
  maxIter = 25,
  weightCol = NULL,
  regParam = 0,
  var.power = 0,
  link.power = 1 - var.power,
  stringIndexerOrderType = c("frequencyDesc", "frequencyAsc", "alphabetDesc",
                             "alphabetAsc"),
  offsetCol = NULL
)
```

```r
## S4 method for signature 'GeneralizedLinearRegressionModel'
summary(object)
```

```r
## S3 method for class 'summary.GeneralizedLinearRegressionModel'
print(x, ...)
```

```r
## S4 method for signature 'GeneralizedLinearRegressionModel'
predict(object, newData)
```
write.ml(object, path, overwrite = FALSE)

Arguments

data   a SparkDataFrame for training.
formula a symbolic description of the model to be fitted. Currently only a few formula operators are supported, including '~', ':', '+', and '-'.
...     additional arguments passed to the method.
family  a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. Refer R family at https://stat.ethz.ch/R-manual/R-devel/library/stats/html/family.html. Currently these families are supported: binomial, gaussian, Gamma, poisson and tweedie. Note that there are two ways to specify the tweedie family.

  • Set family = "tweedie" and specify the var.power and link.power;
  • When package statmod is loaded, the tweedie family is specified using the family definition therein, i.e., tweedie(var.power, link.power).
tol     positive convergence tolerance of iterations.
maxIter integer giving the maximal number of IRLS iterations.
weightCol the weight column name. If this is not set or NULL, we treat all instance weights as 1.0.
regParam regularization parameter for L2 regularization.
var.power the power in the variance function of the Tweedie distribution which provides the relationship between the variance and mean of the distribution. Only applicable to the Tweedie family.
link.power the index in the power link function. Only applicable to the Tweedie family.
stringIndexerOrderType how to order categories of a string feature column. This is used to decide the base level of a string feature as the last category after ordering is dropped when encoding strings. Supported options are "frequencyDesc", "frequencyAsc", "alphabetDesc", and "alphabetAsc". The default value is "frequencyDesc". When the ordering is set to "alphabetDesc", this drops the same category as R when encoding strings.
offsetCol the offset column name. If this is not set or empty, we treat all instance offsets as 0.0. The feature specified as offset has a constant coefficient of 1.0.
object a fitted generalized linear model.
x summary object of fitted generalized linear model returned by summary function.
newData a SparkDataFrame for testing.
path the directory where the model is saved.
overwrite overwrites or not if the output path already exists. Default is FALSE which means throw exception if the output path exists.
spark.glm returns a fitted generalized linear model.

`summary` returns summary information of the fitted model, which is a list. The list of components includes at least the coefficients (coefficients matrix, which includes coefficients, standard error of coefficients, t value and p value), null.deviance (null/residual degrees of freedom), aic (AIC) and iter (number of iterations IRLS takes). If there are collinear columns in the data, the coefficients matrix only provides coefficients.

`predict` returns a SparkDataFrame containing predicted labels in a column named "prediction".

Note

spark.glm since 2.0.0
summary(GeneralizedLinearRegressionModel) since 2.0.0
print.summary.GeneralizedLinearRegressionModel since 2.0.0
predict(GeneralizedLinearRegressionModel) since 1.5.0
write.ml(GeneralizedLinearRegressionModel, character) since 2.0.0

See Also

glm, read.ml

Examples

```r
## Not run:
sparkR.session()
t <- as.data.frame(Titanic, stringsAsFactors = FALSE)
df <- createDataFrame(t)
model <- spark.glm(df, Freq ~ Sex + Age, family = "gaussian")
summary(model)

# fitted values on training data
fitted <- predict(model, df)
head(select(fitted, "Freq", "prediction"))

# save fitted model to input path
path <- "path/to/model"
write.ml(model, path)

# can also read back the saved model and print
savedModel <- read.ml(path)
summary(savedModel)

# note that the default string encoding is different from R's glm
model2 <- glm(Freq ~ Sex + Age, family = "gaussian", data = t)
summary(model2)

# use stringIndexerOrderType = "alphabetDesc" to force string encoding
to be consistent with R
model3 <- spark.glm(df, Freq ~ Sex + Age, family = "gaussian",
                   stringIndexerOrderType = "alphabetDesc")
```
summary(model)
# fit tweedie model
model <- spark.glm(df, Freq ~ Sex + Age, family = "tweedie",
                   var.power = 1.2, link.power = 0)
summary(model)

# use the tweedie family from statmod
library(statmod)
model <- spark.glm(df, Freq ~ Sex + Age, family = tweedie(1.2, 0))
summary(model)

## End(Not run)

---

**spark.isoreg**

*Isotonic Regression Model*

**Description**

Fits an Isotonic Regression model against a SparkDataFrame, similarly to R's isoreg(). Users can print, make predictions on the produced model and save the model to the input path.

**Usage**

```r
spark.isoreg(data, formula, ...)
```

## S4 method for signature 'SparkDataFrame,formula'
spark.isoreg(
  data,
  formula,
  isotonic = TRUE,
  featureIndex = 0,
  weightCol = NULL
)

## S4 method for signature 'IsotonicRegressionModel'
summary(object)

## S4 method for signature 'IsotonicRegressionModel'
predict(object, newData)

## S4 method for signature 'IsotonicRegressionModel,character'
write.ml(object, path, overwrite = FALSE)

**Arguments**

data SparkDataFrame for training.
spark.isoreg

formula  A symbolic description of the model to be fitted. Currently only a few formula operators are supported, including `~`, `::`, `+`, and `-`.

... additional arguments passed to the method.

isotonic  Whether the output sequence should be isotonic/increasing (TRUE) or anti-tonic/decreasing (FALSE).

featureIndex  The index of the feature if `featuresCol` is a vector column (default: 0), no effect otherwise.

weightCol  The weight column name.

object  a fitted IsotonicRegressionModel.

newData  SparkDataFrame for testing.

path  The directory where the model is saved.

overwrite  Overwrites or not if the output path already exists. Default is FALSE which means throw exception if the output path exists.

Value

spark.isoreg returns a fitted Isotonic Regression model.

summary returns summary information of the fitted model, which is a list. The list includes model’s boundaries (boundaries in increasing order) and predictions (predictions associated with the boundaries at the same index).

predict returns a SparkDataFrame containing predicted values.

Note

spark.isoreg since 2.1.0

summary(IsotonicRegressionModel) since 2.1.0

predict(IsotonicRegressionModel) since 2.1.0

write.ml(IsotonicRegression, character) since 2.1.0

Examples

```r
## Not run:
sparkR.session()
data <- list(list(7.0, 0.0), list(5.0, 1.0), list(3.0, 2.0),
    list(5.0, 3.0), list(1.0, 4.0))
df <- createDataFrame(data, c("label", "feature"))
model <- spark.isoreg(df, label ~ feature, isotonic = FALSE)
# return model boundaries and prediction as lists
result <- summary(model, df)
# prediction based on fitted model
predict_data <- list(list(-2.0), list(-1.0), list(0.5),
    list(0.75), list(1.0), list(2.0), list(9.0))
predict_df <- createDataFrame(predict_data, c("feature"))
# get prediction column
predict_result <- collect(select(predict(model, predict_df), "prediction"))
```
## Description

Fits a k-means clustering model against a SparkDataFrame, similarly to R’s kmeans(). Users can call `summary` to print a summary of the fitted model, `predict` to make predictions on new data, and `write.ml`/`read.ml` to save/load fitted models.

## Usage

```r
spark.kmeans(data, formula, ...)  
## S4 method for signature 'SparkDataFrame,formula'
spark.kmeans(
  data,
  formula,
  k = 2,
  maxIter = 20,
  initMode = c("k-means||", "random"),
  seed = NULL,
  initSteps = 2,
  tol = 1e-04
)
```

```r
## S4 method for signature 'KMeansModel'
summary(object)
```

```r
## S4 method for signature 'KMeansModel'
predict(object, newData)
```

```r
## S4 method for signature 'KMeansModel,character'
write.ml(object, path, overwrite = FALSE)
```

## Arguments

- **data**: a SparkDataFrame for training.
spark.kmeans

formula a symbolic description of the model to be fitted. Currently only a few formula operators are supported, including `~`, `:`, `+`, and `-`. Note that the response variable of formula is empty in `spark.kmeans`.

... additional argument(s) passed to the method.

k number of centers.

maxIter maximum iteration number.

initMode the initialization algorithm chosen to fit the model.

seed the random seed for cluster initialization.

initSteps the number of steps for the k-means|| initialization mode. This is an advanced setting, the default of 2 is almost always enough. Must be > 0.

tol convergence tolerance of iterations.

object a fitted k-means model.

newData a SparkDataFrame for testing.

path the directory where the model is saved.

overwrite overwrites or not if the output path already exists. Default is FALSE which means throw exception if the output path exists.

Value

`spark.kmeans` returns a fitted k-means model.

`summary` returns summary information of the fitted model, which is a list. The list includes the model's k (the configured number of cluster centers), coefficients (model cluster centers), size (number of data points in each cluster), cluster (cluster centers of the transformed data), is.loaded (whether the model is loaded from a saved file), and clusterSize (the actual number of cluster centers. When using `initMode = "random"`, `clusterSize` may not equal to k).

`predict` returns the predicted values based on a k-means model.

Note

spark.kmeans since 2.0.0

summary(KMeansModel) since 2.0.0

predict(KMeansModel) since 2.0.0

write.ml(KMeansModel, character) since 2.0.0

See Also

predict, read.ml, write.ml

Examples

```r
## Not run:
sparkR.session()
t <- as.data.frame(Titanic)
df <- createDataFrame(t)
model <- spark.kmeans(df, Class ~ Survived, k = 4, initMode = "random")
```
summary(model)

# fitted values on training data
fitted <- predict(model, df)
head(select(fitted, "Class", "prediction"))

# save fitted model to input path
path <- "path/to/model"
write.ml(model, path)

# can also read back the saved model and print
savedModel <- read.ml(path)
summary(savedModel)

## End(Not run)

---

**spark.kstest**  
*(One-Sample) Kolmogorov-Smirnov Test*

**Description**

spark.kstest Conduct the two-sided Kolmogorov-Smirnov (KS) test for data sampled from a continuous distribution.

By comparing the largest difference between the empirical cumulative distribution of the sample data and the theoretical distribution we can provide a test for the the null hypothesis that the sample data comes from that theoretical distribution.

Users can call summary to obtain a summary of the test, and print.summary.KSTest to print out a summary result.

**Usage**

```r
spark.kstest(data, ...)
```

## S4 method for signature 'SparkDataFrame'

```r
spark.kstest(
  data,
  testCol = "test",
  nullHypothesis = c("norm"),
  distParams = c(0, 1)
)
```

## S4 method for signature 'KSTest'

```r
summary(object)
```

## S3 method for class 'summary.KSTest'

```r
print(x, ...)
```
Arguments

data a SparkDataFrame of user data.
... additional argument(s) passed to the method.
testCol column name where the test data is from. It should be a column of double type.
nullHypothesis name of the theoretical distribution tested against. Currently only "norm" for normal distribution is supported.
distParams parameters(s) of the distribution. For nullHypothesis = "norm", we can provide as a vector the mean and standard deviation of the distribution. If none is provided, then standard normal will be used. If only one is provided, then the standard deviation will be set to be one.
object test result object of KSTest by spark.ktest.
x summary object of KSTest returned by summary.

Value

spark.ktest returns a test result object.

summary returns summary information of KSTest object, which is a list. The list includes the p.value (p-value), statistic (test statistic computed for the test), nullHypothesis (the null hypothesis with its parameters tested against) and degreesOfFreedom (degrees of freedom of the test).

Note

spark.ktest since 2.1.0
summary(KSTest) since 2.1.0
print.summary.KSTest since 2.1.0

See Also

MLlib: Hypothesis Testing

Examples

```r
## Not run:
data <- data.frame(test = c(0.1, 0.15, 0.2, 0.3, 0.25))
df <- createDataFrame(data)
test <- spark.ktest(df, "test", "norm", c(0, 1))

# get a summary of the test result
testSummary <- summary(test)
testSummary

# print out the summary in an organized way
print.summary.KSTest(testSummary)

## End(Not run)
```
spark.lapply  

Run a function over a list of elements, distributing the computations with Spark

Description

Run a function over a list of elements, distributing the computations with Spark. Applies a function in a manner that is similar to doParallel or lapply to elements of a list. The computations are distributed using Spark. It is conceptually the same as the following code: lapply(list, func)

Usage

spark.lapply(list, func)

Arguments

list the list of elements
func a function that takes one argument.

Details

Known limitations:

- variable scoping and capture: compared to R’s rich support for variable resolutions, the distributed nature of SparkR limits how variables are resolved at runtime. All the variables that are available through lexical scoping are embedded in the closure of the function and available as read-only variables within the function. The environment variables should be stored into temporary variables outside the function, and not directly accessed within the function.
- loading external packages: In order to use a package, you need to load it inside the closure. For example, if you rely on the MASS module, here is how you would use it:

  ```r
  train <- function(hyperparam) {
    library(MASS)
    lm.ridge("y ~ x+z", data, lambda=hyperparam)
    model
  }
  ```

Value

a list of results (the exact type being determined by the function)

Note

spark.lapply since 2.0.0
spark.lda

**Examples**

```r
## Not run:
sparkR.session()
doubled <- spark.lapply(1:10, function(x) {2 * x})
## End(Not run)
```

---

**spark.lda**

**Latent Dirichlet Allocation**

**Description**

`spark.lda` fits a Latent Dirichlet Allocation model on a SparkDataFrame. Users can call `summary` to get a summary of the fitted LDA model, `spark.posterior` to compute posterior probabilities on new data, `spark.perplexity` to compute log perplexity on new data and `write.ml/read.ml` to save/load fitted models.

**Usage**

```r
spark.lda(data, ...)
spark.posterior(object, newData)
spark.perplexity(object, data)
```

```r
## S4 method for signature 'SparkDataFrame'
spark.lda(  
data,  
features = "features",  
k = 10,  
maxIter = 20,  
optimizer = c("online", "em"),  
subsamplingRate = 0.05,  
topicConcentration = -1,  
docConcentration = -1,  
customizedStopWords = "",  
maxVocabSize = bitwShiftL(1, 18)  
)
```

```r
## S4 method for signature 'LDAModel'
summary(object, maxTermsPerTopic)
```

```r
## S4 method for signature 'LDAModel,SparkDataFrame'
spark.perplexity(object, data)
```

```r
## S4 method for signature 'LDAModel,SparkDataFrame'
spark.posterior(object, newData)
```
## S4 method for signature 'LDAModel,character'
write.ml(object, path, overwrite = FALSE)

### Arguments

- **data**
  - A SparkDataFrame for training.
- **...**
  - additional argument(s) passed to the method.
- **object**
  - A Latent Dirichlet Allocation model fitted by spark.lda.
- **newData**
  - A SparkDataFrame for testing.
- **features**
  - Features column name. Either libSVM-format column or character-format column is valid.
- **k**
  - Number of topics.
- **maxIter**
  - Maximum iterations.
- **optimizer**
  - Optimizer to train an LDA model, "online" or "em", default is "online".
- **subsamplingRate**
  - (For online optimizer) Fraction of the corpus to be sampled and used in each iteration of mini-batch gradient descent, in range (0, 1].
- **topicConcentration**
  - concentration parameter (commonly named beta or eta) for the prior placed on topic distributions over terms, default -1 to set automatically on the Spark side. Use summary to retrieve the effective topicConcentration. Only 1-size numeric is accepted.
- **docConcentration**
  - concentration parameter (commonly named alpha) for the prior placed on documents distributions over topics (theta), default -1 to set automatically on the Spark side. Use summary to retrieve the effective docConcentration. Only 1-size or k-size numeric is accepted.
- **customizedStopWords**
  - stopwords that need to be removed from the given corpus. Ignore the parameter if libSVM-format column is used as the features column.
- **maxVocabSize**
  - maximum vocabulary size, default $1 \times 18$
- **maxTermsPerTopic**
  - Maximum number of terms to collect for each topic. Default value of 10.
- **path**
  - The directory where the model is saved.
- **overwrite**
  - Overwrites or not if the output path already exists. Default is FALSE which means throw exception if the output path exists.

### Value

- **spark.lda** returns a fitted Latent Dirichlet Allocation model.
- **summary** returns summary information of the fitted model, which is a list. The list includes
  - **docConcentration**
    - concentration parameter commonly named alpha for the prior placed on documents distributions over topics theta
spark.lda since 2.1.0
summary(LDAModel) since 2.1.0
spark.perplexity(LDAModel) since 2.1.0
spark.posterior(LDAModel) since 2.1.0
write.ml(LDAModel, character) since 2.1.0

See Also
topicmodels: https://cran.r-project.org/package=topicmodels
read.ml

Examples
## Not run:
text <- read.df("data/nlib/sample_lda_libsvm_data.txt", source = "libsvm")
model <- spark.lida(data = text, optimizer = "em")

# get a summary of the model
summary(model)

# compute posterior probabilities
posterior <- spark.posterior(model, text)
showDF(posterior)

# compute perplexity
perplexity <- spark.perplexity(model, text)

# save and load the model
path <- "path/to/model"
write.ml(model, path)
savedModel <- read.ml(path)
summary(savedModel)

## End(Not run)

---

**spark.logit**

*Logistic Regression Model*

**Description**

Fits an logistic regression model against a SparkDataFrame. It supports "binomial": Binary logistic regression with pivoting; "multinomial": Multinomial logistic (softmax) regression without pivoting, similar to glmnet. Users can print, make predictions on the produced model and save the model to the input path.

**Usage**

`spark.logit(data, formula, ...)`

## S4 method for signature 'SparkDataFrame,formula'
spark.logit(
  data,
  formula,
  regParam = 0,
  elasticNetParam = 0,
  maxIter = 100,
  tol = 1e-06,
  family = "auto",
  standardization = TRUE,
  thresholds = 0.5,
  weightCol = NULL,
  aggregationDepth = 2,
  lowerBoundsOnCoefficients = NULL,
  upperBoundsOnCoefficients = NULL,
  lowerBoundsOnIntercepts = NULL,
  upperBoundsOnIntercepts = NULL,
  handleInvalid = c("error", "keep", "skip")
)
### S4 method for signature 'LogisticRegressionModel'

**summary(object)**

**predict(object, newData)**

**write.ml(object, path, overwrite = FALSE)**

**Arguments**

- `data` SparkDataFrame for training.
- `formula` A symbolic description of the model to be fitted. Currently only a few formula operators are supported, including `~`, `.` , `|`, `+`, and `-`.
- `...` additional arguments passed to the method.
- `regParam` the regularization parameter.
- `elasticNetParam` the ElasticNet mixing parameter. For alpha = 0.0, the penalty is an L2 penalty. For alpha = 1.0, it is an L1 penalty. For 0.0 < alpha < 1.0, the penalty is a combination of L1 and L2. Default is 0.0 which is an L2 penalty.
- `maxIter` maximum iteration number.
- `tol` convergence tolerance of iterations.
- `family` the name of family which is a description of the label distribution to be used in the model. Supported options:
  - "auto": Automatically select the family based on the number of classes: If number of classes == 1 || number of classes == 2, set to "binomial". Else, set to "multinomial".
  - "binomial": Binary logistic regression with pivoting.
  - "multinomial": Multinomial logistic (softmax) regression without pivoting.
- `standardization` whether to standardize the training features before fitting the model. The coefficients of models will be always returned on the original scale, so it will be transparent for users. Note that with/without standardization, the models should be always converged to the same solution when no regularization is applied. Default is TRUE, same as glmnet.
- `thresholds` in binary classification, in range [0, 1]. If the estimated probability of class label 1 is > threshold, then predict 1, else 0. A high threshold encourages the model to predict 0 more often; a low threshold encourages the model to predict 1 more often. Note: Setting this with threshold p is equivalent to setting thresholds c(1-p, p). In multiclass (or binary) classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class’s threshold.
- `weightCol` The weight column name.
aggregationDepth
The depth for treeAggregate (greater than or equal to 2). If the dimensions of features or the number of partitions are large, this param could be adjusted to a larger size. This is an expert parameter. Default value should be good for most cases.

lowerBoundsOnCoefficients
The lower bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) for multinomial regression. It is a R matrix.

upperBoundsOnCoefficients
The upper bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) for multinomial regression. It is a R matrix.

lowerBoundsOnIntercepts
The lower bounds on intercepts if fitting under bound constrained optimization. The bounds vector size must be equal to 1 for binomial regression, or the number of classes for multinomial regression.

upperBoundsOnIntercepts
The upper bounds on intercepts if fitting under bound constrained optimization. The bound vector size must be equal to 1 for binomial regression, or the number of classes for multinomial regression.

handleInvalid
How to handle invalid data (unseen labels or NULL values) in features and label column of string type. Supported options: "skip" (filter out rows with invalid data), "error" (throw an error), "keep" (put invalid data in a special additional bucket, at index numLabels). Default is "error".

object
an LogisticRegressionModel fitted by spark.logit.

newData
a SparkDataFrame for testing.

path
The directory where the model is saved.

overwrite
Overwrites or not if the output path already exists. Default is FALSE which means throw exception if the output path exists.

Value

spark.logit returns a fitted logistic regression model.

summary returns summary information of the fitted model, which is a list. The list includes coefficients (coefficients matrix of the fitted model).

predict returns the predicted values based on an LogisticRegressionModel.

Note

spark.logit since 2.1.0

summary(LogisticRegressionModel) since 2.1.0

predict(LogisticRegressionModel) since 2.1.0

write.ml(LogisticRegression, character) since 2.1.0
spark.mlp

Examples

```r
## Not run:
sparkR.session()
# binary logistic regression
t <- as.data.frame(Titanic)
training <- createDataFrame(t)
model <- spark.logit(training, Survived ~ ., regParam = 0.5)
summary <- summary(model)

# fitted values on training data
fitted <- predict(model, training)

# save fitted model to input path
path <- "path/to/model"
write.ml(model, path)

# can also read back the saved model and predict
# Note that summary deos not work on loaded model
savedModel <- read.ml(path)
summary(savedModel)

# binary logistic regression against two classes with
# upperBoundsOnCoefficients and upperBoundsOnIntercepts
ubc <- matrix(c(1.0, 0.0, 1.0, 0.0), nrow = 1, ncol = 4)
model <- spark.logit(training, Species ~ .,
                      upperBoundsOnCoefficients = ubc,
                      upperBoundsOnIntercepts = 1.0)

# multinomial logistic regression
model <- spark.logit(training, Class ~ ., regParam = 0.5)
summary <- summary(model)

# multinomial logistic regression with
# lowerBoundsOnCoefficients and lowerBoundsOnIntercepts
lbc <- matrix(c(0.0, -1.0, 0.0, -1.0, 0.0, -1.0, 0.0, -1.0), nrow = 2, ncol = 4)
lbi <- as.array(c(0.0, 0.0))
model <- spark.logit(training, Species ~ ., family = "multinomial",
                     lowerBoundsOnCoefficients = lbc,
                     lowerBoundsOnIntercepts = lbi)

## End(Not run)
```

Description

spark.mlp fits a multi-layer perceptron neural network model against a SparkDataFrame. Users can call summary to print a summary of the fitted model, predict to make predictions on new data, and write.ml/read.ml to save/load fitted models. Only categorical data is supported. For more details, see Multilayer Perceptron
Usage

spark.mlp(data, formula, ...)

## S4 method for signature 'SparkDataFrame,formula'
spark.mlp(
  data,
  formula,
  layers,
  blockSize = 128,
  solver = "l-bfgs",
  maxIter = 100,
  tol = 1e-06,
  stepSize = 0.03,
  seed = NULL,
  initialWeights = NULL,
  handleInvalid = c("error", "keep", "skip")
)

## S4 method for signature 'MultilayerPerceptronClassificationModel'
summary(object)

## S4 method for signature 'MultilayerPerceptronClassificationModel'
predict(object, newData)

## S4 method for signature 'MultilayerPerceptronClassificationModel,character'
write.ml(object, path, overwrite = FALSE)

Arguments

data a SparkDataFrame of observations and labels for model fitting.
formula a symbolic description of the model to be fitted. Currently only a few formula
operators are supported, including '~', ':', '+', and '-'.
... additional arguments passed to the method.
layers integer vector containing the number of nodes for each layer.
blockSize blockSize parameter.
solver solver parameter, supported options: "gd" (minibatch gradient descent) or "l-
bfgs".
maxIter maximum iteration number.
tol convergence tolerance of iterations.
stepSize stepSize parameter.
seed seed parameter for weights initialization.
initialWeights initialWeights parameter for weights initialization, it should be a numeric vector.
handleInvalid How to handle invalid data (unseen labels or NULL values) in features and label
column of string type. Supported options: "skip" (filter out rows with invalid
data), "error" (throw an error), "keep" (put invalid data in a special additional
bucket, at index numLabels). Default is "error".
spark.mlp

object a Multilayer Perceptron Classification Model fitted by spark.mlp
newData a SparkDataFrame for testing.
path the directory where the model is saved.
overwrite overwrites or not if the output path already exists. Default is FALSE which means throw exception if the output path exists.

Value

spark.mlp returns a fitted Multilayer Perceptron Classification Model.
summary returns summary information of the fitted model, which is a list. The list includes numOfInputs (number of inputs), numOfOutputs (number of outputs), layers (array of layer sizes including input and output layers), and weights (the weights of layers). For weights, it is a numeric vector with length equal to the expected given the architecture (i.e., for 8-10-2 network, 112 connection weights).
predict returns a SparkDataFrame containing predicted labeled in a column named "prediction".

Note

spark.mlp since 2.1.0
summary(MultilayerPerceptronClassificationModel) since 2.1.0
predict(MultilayerPerceptronClassificationModel) since 2.1.0
write.ml(MultilayerPerceptronClassificationModel, character) since 2.1.0

See Also

read.ml
write.ml

Examples

```r
## Not run:
df <- read.df("data/mllib/sample_multiclass_classification_data.txt", source = "libsvm")

# fit a Multilayer Perceptron Classification Model
model <- spark.mlp(df, label ~ features, blockSize = 128, layers = c(4, 3), solver = "l-bfgs",
                   maxIter = 100, tol = 0.5, stepSize = 1, seed = 1,
                   initialWeights = c(0, 0, 0, 0, 0, 5, 5, 5, 5, 5, 9, 9, 9, 9, 9, 9))

# get the summary of the model
summary(model)

# make predictions
predictions <- predict(model, df)

# save and load the model
path <- "path/to/model"
write.ml(model, path)
savedModel <- read.ml(path)
```
Summary

## End(Not run)

---

**spark.naiveBayes**  
*Naive Bayes Models*

**Description**

`spark.naiveBayes` fits a Bernoulli naive Bayes model against a SparkDataFrame. Users can call `summary` to print a summary of the fitted model, `predict` to make predictions on new data, and `write.ml`/`read.ml` to save/load fitted models. Only categorical data is supported.

**Usage**

```r
spark.naiveBayes(data, formula, ...)  
## S4 method for signature 'SparkDataFrame,formula'
spark.naiveBayes(  
  data,  
  formula,  
  smoothing = 1,  
  handleInvalid = c("error", "keep", "skip")
)
```

```r
## S4 method for signature 'NaiveBayesModel'
summary(object)
```

```r
## S4 method for signature 'NaiveBayesModel'
predict(object, newData)
```

```r
## S4 method for signature 'NaiveBayesModel,character'
write.ml(object, path, overwrite = FALSE)
```

**Arguments**

- **data**  
  a SparkDataFrame of observations and labels for model fitting.

- **formula**  
  a symbolic description of the model to be fitted. Currently only a few formula operators are supported, including `~`, `.` `:`, `+`, and `-`.

- **...**  
  additional argument(s) passed to the method. Currently only smoothing.

- **smoothing**  
  smoothing parameter.

- **handleInvalid**  
  How to handle invalid data (unseen labels or NULL values) in features and label column of string type. Supported options: "skip" (filter out rows with invalid data), "error" (throw an error), "keep" (put invalid data in a special additional bucket, at index numLabels). Default is "error".

- **object**  
  a naive Bayes model fitted by `spark.naiveBayes`. 
newData a SparkDataFrame for testing.
path the directory where the model is saved.
overwrite overwrites or not if the output path already exists. Default is FALSE which means throw exception if the output path exists.

Value

spark.naiveBayes returns a fitted naive Bayes model.
summary returns summary information of the fitted model, which is a list. The list includes apriori (the label distribution) and tables (conditional probabilities given the target label).
predict returns a SparkDataFrame containing predicted labeled in a column named "prediction".

Note

spark.naiveBayes since 2.0.0
summary(NaiveBayesModel) since 2.0.0
predict(NaiveBayesModel) since 2.0.0
write.ml(NaiveBayesModel, character) since 2.0.0

See Also

e1071: https://cran.r-project.org/package=e1071
write.ml

Examples

```r
## Not run:
data <- as.data.frame(UCBAdmissions)
df <- createDataFrame(data)

# fit a Bernoulli naive Bayes model
model <- spark.naiveBayes(df, Admit ~ Gender + Dept, smoothing = 0)

# get the summary of the model
summary(model)

# make predictions
predictions <- predict(model, df)

# save and load the model
path <- "path/to/model"
write.ml(model, path)
savedModel <- read.ml(path)
summary(savedModel)

## End(Not run)
```
spark.randomForest  Random Forest Model for Regression and Classification

Description

spark.randomForest fits a Random Forest Regression model or Classification model on a Spark-DataFrame. Users can call summary to get a summary of the fitted Random Forest model, predict to make predictions on new data, and write.ml/read.ml to save/load fitted models. For more details, see Random Forest Regression and Random Forest Classification

Usage

spark.randomForest(data, formula, ...)

## S4 method for signature 'SparkDataFrame,formula'
spark.randomForest(
data,
formula,
type = c("regression", "classification"),
maxDepth = 5,
maxBins = 32,
numTrees = 20,
impurity = NULL,
featureSubsetStrategy = "auto",
seed = NULL,
subsamplingRate = 1,
minInstancesPerNode = 1,
minInfoGain = 0,
checkpointInterval = 10,
maxMemoryInMB = 256,
cacheNodeIds = FALSE,
handleInvalid = c("error", "keep", "skip")
)

## S4 method for signature 'RandomForestRegressionModel'
summary(object)

## S3 method for class 'summary.RandomForestRegressionModel'
print(x, ...)

## S4 method for signature 'RandomForestClassificationModel'
summary(object)

## S3 method for class 'summary.RandomForestClassificationModel'
print(x, ...)

## S4 method for signature 'RandomForestRegressionModel'
predict(object, newData)

## S4 method for signature 'RandomForestClassificationModel'
predict(object, newData)

## S4 method for signature 'RandomForestRegressionModel, character'
write.ml(object, path, overwrite = FALSE)

## S4 method for signature 'RandomForestClassificationModel, character'
write.ml(object, path, overwrite = FALSE)

Arguments

data a SparkDataFrame for training.

formula a symbolic description of the model to be fitted. Currently only a few formula
operators are supported, including `-`, `:`, `+`, and `-`

... additional arguments passed to the method.

type type of model, one of "regression" or "classification", to fit

maxDepth Maximum depth of the tree (>= 0).

maxBins Maximum number of bins used for discretizing continuous features and for
choosing how to split on features at each node. More bins give higher gran-
ularity. Must be >= 2 and >= number of categories in any categorical feature.

numTrees Number of trees to train (>= 1).

impurity Criterion used for information gain calculation. For regression, must be "variance". For classification, must be one of "entropy" and "gini", default is "gini".

featureSubsetStrategy The number of features to consider for splits at each tree node. Supported op-
tions: "auto" (choose automatically for task: If numTrees == 1, set to "all." If
numTrees > 1 (forest), set to "sqrt" for classification and to "onethird" for re-
gression), "all" (use all features), "onethird" (use 1/3 of the features), "sqrt" (use
sqrt(number of features)), "log2" (use log2(number of features)), "n": (when n
is in the range (0, 1.0], use n * number of features. When n is in the range (1,
number of features), use n features). Default is "auto".

seed integer seed for random number generation.

subsamplingRate Fraction of the training data used for learning each decision tree, in range (0, 1].

minInstancesPerNode Minimum number of instances each child must have after split.

minInfoGain Minimum information gain for a split to be considered at a tree node.

checkpointInterval Param for set checkpoint interval (>= 1) or disable checkpoint (-1). Note: this
setting will be ignored if the checkpoint directory is not set.

maxMemoryInMB Maximum memory in MB allocated to histogram aggregation.
cacheNodeIds
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. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.

handleInvalid
How to handle invalid data (unseen labels or NULL values) in features and label column of string type in classification model. Supported options: "skip" (filter out rows with invalid data), "error" (throw an error), "keep" (put invalid data in a special additional bucket, at index numLabels). Default is "error".

object
A fitted Random Forest regression model or classification model.

x
summary object of Random Forest regression model or classification model returned by summary.

newData
a SparkDataFrame for testing.

path
The directory where the model is saved.

overwrite
Overwrites or not if the output path already exists. Default is FALSE which means throw exception if the output path exists.

Value

spark.randomForest returns a fitted Random Forest model.

summary returns summary information of the fitted model, which is a list. The list of components includes formula (formula), numFeatures (number of features), features (list of features), featureImportances (feature importances), maxDepth (max depth of trees), numTrees (number of trees), and treeWeights (tree weights).

predict returns a SparkDataFrame containing predicted labeled in a column named "prediction".

Note

spark.randomForest since 2.1.0
summary(RandomForestRegressionModel) since 2.1.0
print.summary.RandomForestRegressionModel since 2.1.0
summary(RandomForestClassificationModel) since 2.1.0
print.summary.RandomForestClassificationModel since 2.1.0
predict(RandomForestClassificationModel) since 2.1.0
write.ml(RandomForestRegressionModel, character) since 2.1.0
write.ml(RandomForestClassificationModel, character) since 2.1.0

Examples

## Not run:
# fit a Random Forest Regression Model
df <- createDataFrame(longley)
model <- spark.randomForest(df, Employed ~ ., type = "regression", maxDepth = 5, maxBins = 16)

# get the summary of the model
summary(model)

# make predictions
predictions <- predict(model, df)

# save and load the model
path <- "path/to/model"
write.ml(model, path)
savedModel <- read.ml(path)
summary(savedModel)

# fit a Random Forest Classification Model
library(sparklyr)

t <- as.data.frame(Titanic)
df <- createDataFrame(t)
model <- spark.randomForest(df, Survived ~ Freq + Age, "classification")

## End(Not run)

---

**spark.survreg**

**Accelerated Failure Time (AFT) Survival Regression Model**

**Description**

spark.survreg fits an accelerated failure time (AFT) survival regression model on a SparkDataFrame. Users can call summary to get a summary of the fitted AFT model, predict to make predictions on new data, and write.ml/read.ml to save/load fitted models.

**Usage**

spark.survreg(data, formula, ...)

## S4 method for signature 'SparkDataFrame,formula'
spark.survreg(
  data,
  formula,
  aggregationDepth = 2,
  stringIndexerOrderType = c("frequencyDesc", "frequencyAsc", "alphabetDesc", "alphabetAsc")
)

## S4 method for signature 'AFTSurvivalRegressionModel'
summary(object)

## S4 method for signature 'AFTSurvivalRegressionModel'
predict(object, newData)

## S4 method for signature 'AFTSurvivalRegressionModel,character'
write.ml(object, path, overwrite = FALSE)
Arguments

- **data**: a SparkDataFrame for training.
- **formula**: a symbolic description of the model to be fitted. Currently only a few formula operators are supported, including `~`, `:`, `+`, and `-`. Note that operator `.` is not supported currently.
- ... additional arguments passed to the method.
- **aggregationDepth**: The depth for treeAggregate (greater than or equal to 2). If the dimensions of features or the number of partitions are large, this param could be adjusted to a larger size. This is an expert parameter. Default value should be good for most cases.
- **stringIndexerOrderType**: how to order categories of a string feature column. This is used to decide the base level of a string feature as the last category after ordering is dropped when encoding strings. Supported options are "frequencyDesc", "frequencyAsc", "alphabetDesc", and "alphabetAsc". The default value is "frequencyDesc". When the ordering is set to "alphabetDesc", this drops the same category as R when encoding strings.
- **object**: a fitted AFT survival regression model.
- **newData**: a SparkDataFrame for testing.
- **path**: the directory where the model is saved.
- **overwrite**: overwrites or not if the output path already exists. Default is FALSE which means throw exception if the output path exists.

Value

`spark.survreg` returns a fitted AFT survival regression model.

`summary` returns summary information of the fitted model, which is a list. The list includes the model's coefficients (features, coefficients, intercept and log(scale)).

`predict` returns a SparkDataFrame containing predicted values on the original scale of the data (mean predicted value at scale = 1.0).

Note

`spark.survreg` since 2.0.0
`summary(AFTSurvivalRegressionModel)` since 2.0.0
`predict(AFTSurvivalRegressionModel)` since 2.0.0
`write.ml(AFTSurvivalRegressionModel, character)` since 2.0.0

See Also

- survival: [https://cran.r-project.org/package=survival](https://cran.r-project.org/package=survival)
- `write.ml`
spark.svmLinear

## Examples

```r
### Not run:
df <- createDataFrame(ovarian)
model <- spark.survreg(df, Surv(futime, fustat) ~ ecog_ps + rx)

# get a summary of the model
summary(model)

# make predictions
predicted <- predict(model, df)
showDF(predicted)

# save and load the model
path <- "path/to/model"
write.ml(model, path)
savedModel <- read.ml(path)
summary(savedModel)

### End(Not run)
```

---

**spark.svmLinear**

### Linear SVM Model

**Description**

Fits a linear SVM model against a SparkDataFrame, similar to svm in e1071 package. Currently only supports binary classification model with linear kernel. Users can print, make predictions on the produced model and save the model to the input path.

**Usage**

```r
spark.svmLinear(data, formula, ...)
```

### S4 method for signature 'SparkDataFrame,formula'

```r
spark.svmLinear(
  data,
  formula,
  regParam = 0,
  maxIter = 100,
  tol = 1e-06,
  standardization = TRUE,
  threshold = 0,
  weightCol = NULL,
  aggregationDepth = 2,
  handleInvalid = c("error", "keep", "skip")
)
```

### S4 method for signature 'LinearSVCModel'
predict(object, newData)

## S4 method for signature 'LinearSVCModel'
summary(object)

## S4 method for signature 'LinearSVCModel,character'
write.ml(object, path, overwrite = FALSE)

Arguments

data  SparkDataFrame for training.
formula A symbolic description of the model to be fitted. Currently only a few formula
operators are supported, including '~', ':', '+', and '-'.
... additional arguments passed to the method.
regParam The regularization parameter. Only supports L2 regularization currently.
maxIter Maximum iteration number.
tol Convergence tolerance of iterations.
standardization Whether to standardize the training features before fitting the model. The co-
efficients of models will be always returned on the original scale, so it will be
transparent for users. Note that with/without standardization, the models should
be always converged to the same solution when no regularization is applied.
threshold The threshold in binary classification applied to the linear model prediction.
This threshold can be any real number, where Inf will make all predictions 0.0
and -Inf will make all predictions 1.0.
weightCol The weight column name.
aggregationDepth The depth for treeAggregate (greater than or equal to 2). If the dimensions of
features or the number of partitions are large, this param could be adjusted to a
larger size. This is an expert parameter. Default value should be good for most
cases.
handleInvalid How to handle invalid data (unseen labels or NULL values) in features and label
column of string type. Supported options: "skip" (filter out rows with invalid
data), "error" (throw an error), "keep" (put invalid data in a special additional
bucket, at index numLabels). Default is "error".
object a LinearSVCModel fitted by spark.svmLinear.
newData a SparkDataFrame for testing.
path The directory where the model is saved.
overwrite Overwrites or not if the output path already exists. Default is FALSE which
means throw exception if the output path exists.

Value

spark.svmLinear returns a fitted linear SVM model.
predict returns the predicted values based on a LinearSVCModel.
summary returns summary information of the fitted model, which is a list. The list includes coefficients (coefficients of the fitted model), numClasses (number of classes), numFeatures (number of features).

Note

spark.svmLinear since 2.2.0
predict(LinearSVCModel) since 2.2.0
summary(LinearSVCModel) since 2.2.0
write.ml(LogisticRegression, character) since 2.2.0

Examples

```r
## Not run:
sparkR.session()
t <- as.data.frame(Titanic)
training <- createDataFrame(t)
model <- spark.svmLinear(training, Survived ~ ., regParam = 0.5)
summary <- summary(model)

# fitted values on training data
fitted <- predict(model, training)

# save fitted model to input path
path <- "path/to/model"
write.ml(model, path)

# can also read back the saved model and predict
# Note that summary deos not work on loaded model
savedModel <- read.ml(path)
summary(savedModel)

## End(Not run)
```

---

**SparkDataFrame-class**  
*S4 class that represents a SparkDataFrame*

**Description**

SparkDataFrames can be created using functions like `createDataFrame`, `read.json`, `table` etc.

**Slots**

- `env`  
  An R environment that stores bookkeeping states of the SparkDataFrame  
- `sdf`  
  A Java object reference to the backing Scala DataFrame

**Note**

SparkDataFrame since 2.0.0
See Also

createDataFrame, read.json, table

https://spark.apache.org/docs/latest/sparkr.html#sparkr-dataframes

Other SparkDataFrame functions: agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()
Details

This is a low level function to access the JVM directly and should only be used for advanced use cases. The arguments and return values that are primitive R types (like integer, numeric, character, lists) are automatically translated to/from Java types (like Integer, Double, String, Array). A full list can be found in serialize.R and deserialize.R in the Apache Spark code base.

Value

the return value of the Java method. Either returned as a R object if it can be deserialized or returned as a "jobj". See details section for more.

Note

sparkR.callJMethod since 2.0.1

See Also

sparkR.callJStatic, sparkR.newJObject

Examples

## Not run:
sparkR.session() # Need to have a Spark JVM running before calling newJObject
# Create a Java ArrayList and populate it
jarray <- sparkR.newJObject("java.util.ArrayList")
sparkR.callJMethod(jarray, "add", 42L)
sparkR.callJMethod(jarray, "get", 0L) # Will print 42

## End(Not run)
Details

This is a low level function to access the JVM directly and should only be used for advanced use cases. The arguments and return values that are primitive R types (like integer, numeric, character, lists) are automatically translated to/from Java types (like Integer, Double, String, Array). A full list can be found in serialize.R and deserialize.R in the Apache Spark code base.

Value

the return value of the Java method. Either returned as a R object if it can be deserialized or returned as a "jobj". See details section for more.

Note

sparkR.callJStatic since 2.0.1

See Also

sparkR.callJMethod, sparkR.newJObject

Examples

```
## Not run:
sparkR.session() # Need to have a Spark JVM running before calling callJStatic
sparkR.callJStatic("java.lang.System", "currentTimeMillis")
sparkR.callJStatic("java.lang.System", "getProperty", "java.home")
## End(Not run)
```

sparkR.conf
Get Runtime Config from the current active SparkSession

Description

Get Runtime Config from the current active SparkSession. To change SparkSession Runtime Config, please see sparkR.session().

Usage

sparkR.conf(key, defaultValue)

Arguments

key (optional) The key of the config to get, if omitted, all config is returned
defaultValue (optional) The default value of the config to return if they config is not set, if omitted, the call fails if the config key is not set
Value

a list of config values with keys as their names

Note

sparkR.conf since 2.0.0

Examples

```r
## Not run:
sparkR.session()
allConfigs <- sparkR.conf()
masterValue <- unlist(sparkR.conf("spark.master"))
namedConfig <- sparkR.conf("spark.executor.memory", "0g")

## End(Not run)
```

sparkR.init  (Deprecated) Initialize a new Spark Context

Description

This function initializes a new SparkContext.

Usage

```r
sparkR.init(
  master = "", 
  appName = "SparkR", 
  sparkHome = Sys.getenv("SPARK_HOME"), 
  sparkEnvir = list(), 
  sparkExecutorEnv = list(), 
  sparkJars = "", 
  sparkPackages = ""
)
```

Arguments

- **master**  The Spark master URL
- **appName**  Application name to register with cluster manager
- **sparkHome**  Spark Home directory
- **sparkEnvir**  Named list of environment variables to set on worker nodes
- **sparkExecutorEnv**  Named list of environment variables to be used when launching executors
- **sparkJars**  Character vector of jar files to pass to the worker nodes
- **sparkPackages**  Character vector of package coordinates
sparkR.newJObject

Create Java Objects

Description

Create a new Java object in the JVM running the Spark driver. The return value is automatically converted to an R object for simple objects. Other values are returned as a "jobj" which is a reference to an object on JVM.

Usage

sparkR.newJObject(x, ...)

Arguments

x

fully qualified Java class name.

... arguments to be passed to the constructor.

Details

This is a low level function to access the JVM directly and should only be used for advanced use cases. The arguments and return values that are primitive R types (like integer, numeric, character, lists) are automatically translated to/from Java types (like Integer, Double, String, Array). A full list can be found in serialize.R and deserialize.R in the Apache Spark code base.
Value
the object created. Either returned as a R object if it can be deserialized or returned as a "jobb". See
details section for more.

Note
sparkR.newJObject since 2.0.1

See Also
sparkR.callJMethod, sparkR.callJStatic

Examples
```r
## Not run:
sparkR.session() # Need to have a Spark JVM running before calling newJObject
# Create a Java ArrayList and populate it
jarray <- sparkR.newJObject("java.util.ArrayList")
sparkR.callJMethod(jarray, "add", 42L)
sparkR.callJMethod(jarray, "get", 0L) # Will print 42
## End(Not run)
```

---

**sparkR.session**

Get the existing SparkSession or initialize a new SparkSession.

**Description**

SparkSession is the entry point into SparkR. `sparkR.session` gets the existing SparkSession or
initializes a new SparkSession. Additional Spark properties can be set in . . . , and these named
parameters take priority over values in master, appName, named lists of sparkConfig.

**Usage**

```r
sparkR.session(
  master = "",
  appName = "SparkR",
  sparkHome = Sys.getenv("SPARK_HOME"),
  sparkConfig = list(),
  sparkJars = "",
  sparkPackages = "",
  enableHiveSupport = TRUE,
  ...
)
```
Arguments

master the Spark master URL.
appName application name to register with cluster manager.
sparkHome Spark Home directory.
sparkConfig named list of Spark configuration to set on worker nodes.
sparkJars character vector of jar files to pass to the worker nodes.
sparkPackages character vector of package coordinates
enableHiveSupport enable support for Hive, fallback if not built with Hive support; once set, this cannot be turned off on an existing session
... named Spark properties passed to the method.

Details

When called in an interactive session, this method checks for the Spark installation, and, if not found, it will be downloaded and cached automatically. Alternatively, install.spark can be called manually.

A default warehouse is created automatically in the current directory when a managed table is created via sql statement CREATE TABLE, for example. To change the location of the warehouse, set the named parameter spark.sql.warehouse.dir to the SparkSession. Along with the warehouse, an accompanied metastore may also be automatically created in the current directory when a new SparkSession is initialized with enableHiveSupport set to TRUE, which is the default. For more details, refer to Hive configuration at http://spark.apache.org/docs/latest/sql-programming-guide.html#hive-tables.

For details on how to initialize and use SparkR, refer to SparkR programming guide at http://spark.apache.org/docs/latest/sparkr.html#starting-up-sparksession.

Note

sparkR.session since 2.0.0

Examples

```r
## Not run:
sparkR.session()
df <- read.json(path)

sparkR.session("local[2]", "SparkR", "/home/spark")
sparkR.session("yarn-client", "SparkR", "/home/spark",
  list(spark.executor.memory="4g"),
  c("one.jar", "two.jar", "three.jar"),
  c("com.databricks:spark-avro_2.11:2.0.1"))

sparkR.session(spark.master = "yarn-client", spark.executor.memory = "4g")

## End(Not run)
```
sparkR.session.stop

Stop the Spark Session and Spark Context

Description
Stop the Spark Session and Spark Context.

Usage
sparkR.session.stop()
sparkR.stop()

Details
Also terminates the backend this R session is connected to.

Note
sparkR.session.stop since 2.0.0
sparkR.stop since 1.4.0

sparkR.uiWebUrl

Get the URL of the SparkUI instance for the current active SparkSession

Description
Get the URL of the SparkUI instance for the current active SparkSession.

Usage
sparkR.uiWebUrl()

Value
the SparkUI URL, or NA if it is disabled, or not started.

Note
sparkR.uiWebUrl since 2.1.1
Examples

```r
## Not run:
sparkR.session()
url <- sparkR.uiWebUrl()

## End(Not run)
```

---

**sparkR.version**  
*Get version of Spark on which this application is running*

Description

Get version of Spark on which this application is running.

Usage

```r
sparkR.version()
```

Value

a character string of the Spark version

Note

sparkR.version since 2.0.1

Examples

```r
## Not run:
sparkR.session()
version <- sparkR.version()

## End(Not run)
```

---

**sparkRHive.init**  
*(Deprecated) Initialize a new HiveContext*

Description

This function creates a HiveContext from an existing JavaSparkContext.

Usage

```r
sparkRHive.init(jsc = NULL)
```
sparkR.init

Arguments
jsc

The existing JavaSparkContext created with SparkR.init()

Details

Starting SparkR 2.0, a SparkSession is initialized and returned instead. This API is deprecated and kept for backward compatibility only.

Note

sparkRHive.init since 1.4.0

See Also

sparkR.session

Examples

```r
## Not run:
s <- sparkR.init()
sqlContext <- sparkRHive.init(sc)
## End(Not run)
```

Description

This function creates a SparkContext from an existing JavaSparkContext and then uses it to initialize a new SQLContext

Usage

sparkR.init(jsc = NULL)

Arguments

jsc

The existing JavaSparkContext created with SparkR.init()

Details

Starting SparkR 2.0, a SparkSession is initialized and returned instead. This API is deprecated and kept for backward compatibility only.

Note

sparkR.init since 1.4.0
See Also

sparkR.session

Examples

```r
## Not run:
sc <- sparkR.init()
sqlContext <- sparkRSQL.init(sc)

## End(Not run)
```

---

### sql

**SQL Query**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executes a SQL query using Spark, returning the result as a SparkDataFrame.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
</tr>
</thead>
</table>
| ```r
## Default S3 method:
sql(sqlQuery)
``` |

<table>
<thead>
<tr>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlQuery</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkDataFrame</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql since 1.4.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
</table>
| ```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
createOrReplaceTempView(df, "table")
new_df <- sql("SELECT * FROM table")

## End(Not run)
``` |
**startsWith**

**Description**

Determines if entries of `x` start with string (entries of) `prefix` respectively, where strings are recycled to common lengths.

**Usage**

```r
startsWith(x, prefix)
```

## S4 method for signature 'Column'

```r
startsWith(x, prefix)
```

**Arguments**

- `x` vector of character string whose "starts" are considered
- `prefix` character vector (often of length one)

**Note**

startsWith since 1.4.0

**See Also**

Other `colum_func`: `alias()`, `between()`, `cast()`, `endsWith()`, `otherwise()`, `over()`, `substr()`

**status**

**Description**

Prints the current status of the query in JSON format.

**Usage**

```r
status(x)
```

## S4 method for signature 'StreamingQuery'

```r
status(x)
```

**Arguments**

- `x` a StreamingQuery.
stopQuery

Note

status(StreamingQuery) since 2.2.0
experimental

See Also

Other StreamingQuery methods: `awaitTermination()`, `explain()`, `isActive()`, `lastProgress()`, `queryName()`, `stopQuery()`

Examples

```r
## Not run: status(sq)
```

---

Description

Stops the execution of this query if it is running. This method blocks until the execution is stopped.

Usage

```r
stopQuery(x)
```

## S4 method for signature 'StreamingQuery'

```r
stopQuery(x)
```

Arguments

- `x`: a StreamingQuery.

Note

stopQuery(StreamingQuery) since 2.2.0
experimental

See Also

Other StreamingQuery methods: `awaitTermination()`, `explain()`, `isActive()`, `lastProgress()`, `queryName()`, `status()`

Examples

```r
## Not run: stopQuery(sq)
```
storageLevel

Description

Get storageLevel of this SparkDataFrame.

Usage

```r
## S4 method for signature 'SparkDataFrame'
storageLevel(x)
```

Arguments

- `x` the SparkDataFrame to get the storageLevel.

Note

storageLevel since 2.1.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
persist(df, "MEMORY_AND_DISK")
storageLevel(df)
```

## End(Not run)
str

Compactly display the structure of a dataset

Description

Display the structure of a SparkDataFrame, including column names, column types, as well as a small sample of rows.

Usage

```
## S4 method for signature 'SparkDataFrame'
str(object)
```

Arguments

- `object` a SparkDataFrame

Note

str since 1.6.1

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method,broadcast(),cache(),checkpoint(),coalesce(),collect(),colnames(),coltypes(),createOrReplaceTempView(),crossJoin(),cube(),dapplyCollect(),dapply(),describe(),dim(),distinct(),dropDuplicates(),dropna(),drop(),dtypes(),exceptAll(),except(),explain(),filter(),first(),gapplyCollect(),gapply(),getNumPartitions(),group_by(),head(),hint(),histogram(),insertInto(),intersectAll(),intersect(),isLocal(),isStreaming(),join(),limit(),localCheckpoint(),merge(),mutate(),ncol(),nrow(),persist(),printSchema(),randomSplit(),rbind(),rename(),repartitionByRange(),repartition(),rollup(),sample(),saveAsTable(),schema(),selectExpr(),select(),showDF(),show(),storageLevel(),subset(),summary(),take(),toJSON(),unionByName(),union(),unpersist(),withColumn(),withWatermark(),with(),write.df(),write.jdbc(),write.json(),write.orc(),write.parquet(),write.stream(),write.text()

Examples

```
## Not run:
# Create a SparkDataFrame from the Iris dataset
irisDF <- createDataFrame(iris)

# Show the structure of the SparkDataFrame
str(irisDF)

## End(Not run)
```
StreamingQuery-class

S4 class that represents a StreamingQuery

Description

StreamingQuery can be created by using read.stream() and write.stream()

Arguments

ssq A Java object reference to the backing Scala StreamingQuery

Note

StreamingQuery since 2.2.0

experimental

See Also

read.stream

structField

structField

Description

Create a structField object that contains the metadata for a single field in a schema.

Usage

structField(x, ...)

## S3 method for class 'obj'
structField(x, ...)

## S3 method for class 'character'
structField(x, type, nullable = TRUE, ...)

Arguments

x the name of the field.
...
additional argument(s) passed to the method.
type The data type of the field
nullable A logical vector indicating whether or not the field is nullable
**Value**

A structField object.

**Note**

structField since 1.4.0

**Examples**

```r
## Not run:
field1 <- structField("a", "integer")
field2 <- structField("c", "string")
field3 <- structField("avg", "double")
schema <- structType(field1, field2, field3)
df1 <- gapply(df, list("a", "c"),
               function(key, x) { y <- data.frame(key, mean(x$b), stringsAsFactors = FALSE) },
               schema)
## End(Not run)
```

---

**structType**

---

**Description**

Create a structType object that contains the metadata for a SparkDataFrame. Intended for use with createDataFrame and toDF.

**Usage**

```r
structType(x, ...)
```

```
## S3 method for class 'jobj'
structType(x, ...)
```

```
## S3 method for class 'structField'
structType(x, ...)
```

```
## S3 method for class 'character'
structType(x, ...)
```

**Arguments**

- `x` a structField object (created with the structField method). Since Spark 2.3, this can be a DDL-formatted string, which is a comma separated list of field definitions, e.g., "a INT, b STRING".
- `...` additional structField objects
Value

a structType object

Note

structType since 1.4.0

Examples

```r
## Not run:
schema <- structType(structField("a", "integer"), structField("c", "string"),
                      structField("avg", "double"))
df1 <- gapply(df, list("a", "c"),
              function(key, x) { y <- data.frame(key, mean(x$b), stringsAsFactors = FALSE) },
              schema)
schema <- structType("a INT, c STRING, avg DOUBLE")
df1 <- gapply(df, list("a", "c"),
              function(key, x) { y <- data.frame(key, mean(x$b), stringsAsFactors = FALSE) },
              schema)
## End(Not run)
```

subset

Subset

Description

Return subsets of SparkDataFrame according to given conditions

Usage

```
subset(x, ...)

## S4 method for signature 'SparkDataFrame,numericOrcharacter'
x[[i]]

## S4 replacement method for signature 'SparkDataFrame,numericOrcharacter'
x[[i]] <- value

## S4 method for signature 'SparkDataFrame'
x[i, j, ..., drop = F]

## S4 method for signature 'SparkDataFrame'
subset(x, subset, select, drop = F, ...)
```
Arguments

**x**  
a SparkDataFrame.

...  
currently not used.

**i, subset**  
(Optional) a logical expression to filter on rows. For extract operator [[] and  
replacement operator [[]<-, the indexing parameter for a single Column.

**value**  
a Column or an atomic vector in the length of 1 as literal value, or NULL. If NULL,  
the specified Column is dropped.

**j, select**  
expression for the single Column or a list of columns to select from the Spark-  
DataFrame.

**drop**  
if TRUE, a Column will be returned if the resulting dataset has only one column.  
Otherwise, a SparkDataFrame will always be returned.

Value

A new SparkDataFrame containing only the rows that meet the condition with selected columns.

Note

[[ since 1.4.0  
[<- since 2.1.1  
[ since 1.4.0  
since 1.5.0

See Also

withColumn

Other SparkDataFrame functions: SparkDataFrame-class, .agg(), .alias(), .arrange(), .as.data.frame(),  
attach, SparkDataFrame-method, .broadcast(), .cache(), .checkpoint(), .coalesce(), .collect(),  
colnames(), .coltypes(), .createOrReplaceTempView(), .crossJoin(), .cube(), .dapplyCollect(),  
dapply(), .describe(), .dim(), .distinct(), .dropDuplicates(), .dropna(), .drop(), .dtypes(),  
exceptAll(), .except(), .explain(), .filter(), .first(), .gapplyCollect(), .gapply(), .getNumPartitions(),  
group_by(), .head(), .hint(), .histogram(), .insertInto(), .intersectAll(), .intersect(),  
isLocal(), .isStreaming(), .join(), .limit(), .localCheckpoint(), .merge(), .mutate(), .ncol(),  
.nrow(), .persist(), .printSchema(), .randomSplit(), .rbind(), .rename(), .repartitionByRange(),  
.repartition(), .rollup(), .sample(), .saveAsTable(), .schema(), .selectExpr(), .select(),  
.showDF(), .show(), .storageLevel(), .str(), .summary(), .take(), .toJSON(), .unionByName(),  
.union(), .unpersist(), .withColumn(), .withWatermark(), .with(), .write.df(), .write.jdbc(),  
.write.json(), .write.orc(), .write.parquet(), .write.stream(), .write.text()

Other subsetting functions: filter(), select()

Examples

```r
## Not run:
# Columns can be selected using [[ and [

df[[2]] == df["age"]

df[,2] == df,"age"]
```
df[, c("name", "age")]
# Or to filter rows
df[df$age > 20,]
# SparkDataFrame can be subset on both rows and Columns
df[df$name == "Smith", c(1,2)]
df[df$age %in% c(19, 30), 1:2]
subset(df, df$age %in% c(19, 30), 1:2)
subset(df, df$age %in% c(19), select = c(1,2))
subset(df, select = c(1,2))
# Columns can be selected and set
df[["age"]]<- 23
df[[1]] <- df$age
df[[2]] <- NULL # drop column

## End(Not run)

---

### substr

#### Description
An expression that returns a substring.

#### Usage
```r
## S4 method for signature 'Column'
substr(x, start, stop)
```

#### Arguments
- **x**: a Column.
- **start**: starting position. It should be 1-base.
- **stop**: ending position.

#### Note
substr since 1.4.0

#### See Also
Other column functions: `alias()`., `between()`, `cast()`, `endsWith()`, `otherwise()`, `over()`, `startsWith()`

#### Examples
```r
## Not run:
df <- createDataFrame(list(list(a="abcdef")))
collect(select(df, substr(df$a, 1, 4))) # the result is 'abcd'.
collect(select(df, substr(df$a, 2, 4))) # the result is 'bcd'.
```

## End(Not run)
Description

Computes specified statistics for numeric and string columns. Available statistics are:

- count
- mean
- stddev
- min
- max
- arbitrary approximate percentiles specified as a percentage (eg, “75%”)

If no statistics are given, this function computes count, mean, stddev, min, approximate quartiles (percentiles at 25%, 50%, and 75%), and max. This function is meant for exploratory data analysis, as we make no guarantee about the backward compatibility of the schema of the resulting Dataset. If you want to programmatically compute summary statistics, use the `agg` function instead.

Usage

```r
summary(object, ...)
```

## S4 method for signature 'SparkDataFrame'

```r
summary(object, ...)
```

Arguments

- `object` a SparkDataFrame to be summarized.
- `...` (optional) statistics to be computed for all columns.

Value

A SparkDataFrame.

Note

- `summary(SparkDataFrame)` since 1.5.0
- The statistics provided by `summary` were change in 2.3.0 use `describe` for previous defaults.
See Also

describe

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach(), SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset().take(), toJSON(), unionByColumn(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
summary(df)
summary(df, "min", "25%", "75%", "max")
summary(select(df, "age", "height"))
## End(Not run)
```

---

### tableNames

#### Table Names

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the names of tables in the given database as an array.</td>
</tr>
</tbody>
</table>

### Usage

```r
## Default S3 method:
tableNames(databaseName = NULL)
```

### Arguments

- `databaseName` (optional) name of the database

### Value

- a list of table names
Note
tableNames since 1.4.0

Examples
## Not run:
sparkR.session()
tableNames("hive")

## End(Not run)

---

<table>
<thead>
<tr>
<th>tables</th>
<th>Tables</th>
</tr>
</thead>
</table>

Description

Returns a SparkDataFrame containing names of tables in the given database.

Usage

## Default S3 method:
tables(databaseName = NULL)

Arguments
databaseName (optional) name of the database

Value

a SparkDataFrame

Note
tables since 1.4.0

See Also

listTables

Examples

## Not run:
sparkR.session()
tables("hive")

## End(Not run)
Create a SparkDataFrame from a SparkSQL table or view

Description

Returns the specified table or view as a SparkDataFrame. The table or view must already exist or have already been registered in the SparkSession.

Usage

```
tableToDF(tableName)
```

Arguments

- `tableName`: the qualified or unqualified name that designates a table or view. If a database is specified, it identifies the table/view from the database. Otherwise, it first attempts to find a temporary view with the given name and then match the table/view from the current database.

Value

SparkDataFrame

Note

`tableToDF` since 2.0.0

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
createOrReplaceTempView(df, "table")
new_df <- tableToDF("table")
## End(Not run)
```
take  

*Take the first NUM rows of a SparkDataFrame and return the results as a R data.frame*

Description

Take the first NUM rows of a SparkDataFrame and return the results as a R data.frame

Usage

```
take(x, num)
## S4 method for signature 'SparkDataFrame,numERIC'
take(x, num)
```

Arguments

- `x` a SparkDataFrame.
- `num` number of rows to take.

Note

take since 1.4.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()
Description

Converts a SparkDataFrame into a SparkDataFrame of JSON string.

Usage

```r
## S4 method for signature 'SparkDataFrame'
toJson(x)
```

Arguments

- `x`: a SparkDataFrame

Details

Each row is turned into a JSON document with columns as different fields. The returned SparkDataFrame has a single character column with the name `value`.

Value

- a SparkDataFrame

Note

toJSON since 2.2.0

See Also

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach()`, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `groupBy()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isLocal()`, `isStreaming()`, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `ncol()`, `nrow()`, `persist()`, `printSchema()`, `randomSplit()`, `rename()`, `repartitionByRange()`, `repartition()`, `rollup()`, `sample()`, `saveAsTable()`, `schema()`, `selectExpr()`, `select()`, `showDF()`, `show()`, `storageLevel()`, `str()`, `subset()`, `summary()`, `take()`, `unionByName()`, `union()`, `unpersist()`, `withColumn()`, `withWatermark()`, `with()`, `write.df()`, `write.jdbc()`, `write.json()`, `write.orc()`, `write.parquet()`, `write.stream()`, `write.text()`
Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.parquet"
df <- read.parquet(path)
df_json <- toJSON(df)

## End(Not run)
```

<table>
<thead>
<tr>
<th>uncacheTable</th>
<th>Uncache Table</th>
</tr>
</thead>
</table>

Description

Removes the specified table from the in-memory cache.

Usage

```r
## Default S3 method:
uncacheTable(tableName)
```

Arguments

- `tableName` the qualified or unqualified name that designates a table. If no database identifier is provided, it refers to a table in the current database.

Value

SparkDataFrame

Note

uncacheTable since 1.4.0

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
createOrReplaceTempView(df, "table")
uncacheTable("table")

## End(Not run)
```
union

Return a new SparkDataFrame containing the union of rows in this SparkDataFrame and another SparkDataFrame. This is equivalent to UNION ALL in SQL. Input SparkDataFrames can have different schemas (names and data types).

Usage

```r
union(x, y)
unionAll(x, y)
```

Arguments

- `x`: A SparkDataFrame
- `y`: A SparkDataFrame

Details

Note: This does not remove duplicate rows across the two SparkDataFrames. Also as standard in SQL, this function resolves columns by position (not by name).

Value

A SparkDataFrame containing the result of the union.

Note

- `union` since 2.0.0
- `unionAll` since 1.4.0

See Also

`rbind unionByName`

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach()`, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`. 
Examples

## Not run:
sparkR.session()
df1 <- read.json(path)
df2 <- read.json(path2)
unioned <- union(df, df2)
unions <- rbind(df, df2, df3, df4)

## End(Not run)

unionByName

Return a new SparkDataFrame containing the union of rows, matched by column names

Description

Return a new SparkDataFrame containing the union of rows in this SparkDataFrame and another SparkDataFrame. This is different from union function, and both UNION ALL and UNION DISTINCT in SQL as column positions are not taken into account. Input SparkDataFrames can have different data types in the schema.

Usage

unionByName(x, y)

## S4 method for signature 'SparkDataFrame,SparkDataFrame'
unionByName(x, y)

Arguments

x A SparkDataFrame
y A SparkDataFrame

Details

Note: This does not remove duplicate rows across the two SparkDataFrames. This function resolves columns by name (not by position).
Value

A SparkDataFrame containing the result of the union.

Note

unionByName since 2.3.0

See Also

rbind union

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()
Arguments

- `x` the SparkDataFrame to unpersist.
- `...` further arguments to be passed to or from other methods.
- `blocking` whether to block until all blocks are deleted.

Note

unpersist since 1.4.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
p path <- "path/to/file.json"
df <- read.json(path)
persist(df, "MEMORY_AND_DISK")
unpersist(df)

## End(Not run)
```

Description

Creates a WindowSpec with the ordering defined.
windowPartitionBy

Usage

windowOrderBy(col, ...)

## S4 method for signature 'character'
windowOrderBy(col, ...)

## S4 method for signature 'Column'
windowOrderBy(col, ...)

Arguments

- `col` A column name or Column by which rows are ordered within windows.
- `...` Optional column names or Columns in addition to `col`, by which rows are ordered within windows.

Note

windowOrderBy(character) since 2.0.0

windowOrderBy(Column) since 2.0.0

Examples

## Not run:
ws <- windowOrderBy("key1", "key2")
df1 <- select(df, over(lead("value", 1), ws))

ws <- windowOrderBy(df$key1, df$key2)
df1 <- select(df, over(lead("value", 1), ws))

## End(Not run)

Description

Creates a WindowSpec with the partitioning defined.

Usage

windowPartitionBy(col, ...)

## S4 method for signature 'character'
windowPartitionBy(col, ...)

## S4 method for signature 'Column'
windowPartitionBy(col, ...)
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Note

- windowPartitionBy(character) since 2.0.0
- windowPartitionBy(Column) since 2.0.0

Examples

```r
## Not run:
ws <- orderBy(windowPartitionBy("key1", "key2"), "key3")
df1 <- select(df, over(lead("value", 1), ws))

ws <- orderBy(windowPartitionBy(df$key1, df$key2), df$key3)
df1 <- select(df, over(lead("value", 1), ws))

## End(Not run)
```

WindowSpec-class

S4 class that represents a WindowSpec

Description

WindowSpec can be created by using windowPartitionBy() or windowOrderBy().

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>sws</td>
</tr>
</tbody>
</table>

Note

- WindowSpec since 2.0.0

See Also

- windowPartitionBy, windowOrderBy
with

Evaluate a R expression in an environment constructed from a SparkDataFrame

Description

Evaluate a R expression in an environment constructed from a SparkDataFrame with() allows access
to columns of a SparkDataFrame by simply referring to their name. It appends every column of a
SparkDataFrame into a new environment. Then, the given expression is evaluated in this new
environment.

Usage

with(data, expr, ...)

## S4 method for signature 'SparkDataFrame'

with(data, expr, ...)

Arguments

data (SparkDataFrame) SparkDataFrame to use for constructing an environment.
expr (expression) Expression to evaluate.
... arguments to be passed to future methods.

Note

with since 1.6.0

See Also

attach

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(),
attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(),
colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(),
dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(),
exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(),
group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(),
isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(),
nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(),
repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(),
showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(),
union(), unpersist(), withColumn(), withWatermark(), write.df(), write.jdbc(), write.json(),
write.orc(), write.parquet(), write.stream(), write.text()
Examples

```r
## Not run:
with(irisDf, nrow(Sepal_Width))
## End(Not run)
```

### Description

Return a new SparkDataFrame by adding a column or replacing the existing column that has the same name.

### Usage

```r
withColumn(x, colName, col)
```

#### Arguments

- `x`: a SparkDataFrame.
- `colName`: a column name.
- `col`: a Column expression (which must refer only to this SparkDataFrame), or an atomic vector in the length of 1 as literal value.

### Value

A SparkDataFrame with the new column added or the existing column replaced.

### Note

`withColumn` since 1.4.0

### See Also

`rename`, `mutate`, `subset`

Other SparkDataFrame functions: `SparkDataFrame-class`, `agg()`, `alias()`, `arrange()`, `as.data.frame()`, `attach()`, `SparkDataFrame-method`, `broadcast()`, `cache()`, `checkpoint()`, `coalesce()`, `collect()`, `colnames()`, `coltypes()`, `createOrReplaceTempView()`, `crossJoin()`, `cube()`, `dapplyCollect()`, `dapply()`, `describe()`, `dim()`, `distinct()`, `dropDuplicates()`, `dropna()`, `drop()`, `dtypes()`, `exceptAll()`, `except()`, `explain()`, `filter()`, `first()`, `gapplyCollect()`, `gapply()`, `getNumPartitions()`, `group_by()`, `head()`, `hint()`, `histogram()`, `insertInto()`, `intersectAll()`, `intersect()`, `isLocal()``, `isStreaming()``, `join()`, `limit()`, `localCheckpoint()`, `merge()`, `mutate()`, `ncol()`, `nrow()`, `persist()`, `printSchema()`, `randomSplit()`, `rbind()``, `rename()``, `repartitionByRange()`,
withWatermark

repartition(). rollup(). sample(). saveAsTable(). schema(). selectExpr(). select().
showDF(). show(). storageLevel(). str(). subset(). summary(). take(). toJSON(). unionByName().
union(). unpersist(). withWatermark(). with(). write.df(). write.jdbc(). write.json().
write.orc(). write.parquet(). write.stream(). write.text()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
newDF <- withColumn(df, "newCol", df$col1 * 5)
# Replace an existing column
newDF2 <- withColumn(newDF, "newCol", newDF$col1)
newDF3 <- withColumn(newDF, "newCol", 42)
# Use extract operator to set an existing or new column
df[["age"]]<-23
df[[2]] <- df$col1
df[[2]] <- NULL # drop column
## End(Not run)
```

withWatermark

Description

Defines an event time watermark for this streaming SparkDataFrame. A watermark tracks a point in time before which we assume no more late data is going to arrive.

Usage

```r
withWatermark(x, eventTime, delayThreshold)
```

## S4 method for signature 'SparkDataFrame,character,character'
withWatermark(x, eventTime, delayThreshold)

Arguments

- `x` : a streaming SparkDataFrame
- `eventTime` : a string specifying the name of the Column that contains the event time of the row.
- `delayThreshold` : a string specifying the minimum delay to wait to data to arrive late, relative to the latest record that has been processed in the form of an interval (e.g. "1 minute" or "5 hours"). NOTE: This should not be negative.
Details

Spark will use this watermark for several purposes:

- To know when a given time window aggregation can be finalized and thus can be emitted when using output modes that do not allow updates.
- To minimize the amount of state that we need to keep for on-going aggregations.

The current watermark is computed by looking at the $\text{MAX(eventTime)}$ seen across all of the partitions in the query minus a user specified $\text{delayThreshold}$. Due to the cost of coordinating this value across partitions, the actual watermark used is only guaranteed to be at least $\text{delayThreshold}$ behind the actual event time. In some cases we may still process records that arrive more than $\text{delayThreshold}$ late.

Value

a SparkDataFrame.

Note

withWatermark since 2.3.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
schema <- structType(structField("time", "timestamp"), structField("value", "double"))
df <- read.stream("json", path = jsonDir, schema = schema, maxFilesPerTrigger = 1)
df <- withWatermark(df, "time", "10 minutes")

## End(Not run)
```
**write.df**

Save the contents of SparkDataFrame to a data source.

**Description**

The data source is specified by the `source` and a set of options (...). If `source` is not specified, the default data source configured by `spark.sql.sources.default` will be used.

**Usage**

```r
write.df(df, path = NULL, ...)
saveDF(df, path, source = NULL, mode = "error", ...)
write.df(df, path = NULL, ...)
## S4 method for signature 'SparkDataFrame'
write.df(df, path = NULL, source = NULL, mode = "error", ...)
## S4 method for signature 'SparkDataFrame,character'
saveDF(df, path, source = NULL, mode = "error", ...)
```

**Arguments**

- `df` a SparkDataFrame.
- `path` a name for the table.
- `...` additional argument(s) passed to the method.
- `source` a name for external data source.
- `mode` one of 'append', 'overwrite', 'error', 'errorifexists', 'ignore' save mode (it is 'error' by default)

**Details**

Additionally, mode is used to specify the behavior of the save operation when data already exists in the data source. There are four modes:

- 'append': Contents of this SparkDataFrame are expected to be appended to existing data.
- 'overwrite': Existing data is expected to be overwritten by the contents of this SparkDataFrame.
- 'error' or 'errorifexists': An exception is expected to be thrown.
- 'ignore': The save operation is expected to not save the contents of the SparkDataFrame and to not change the existing data.

**Note**

- `write.df` since 1.4.0
- `saveDF` since 1.4.0
write.jdbc

Save the content of SparkDataFrame to an external database table via JDBC.

Description

Save the content of the SparkDataFrame to an external database table via JDBC. Additional JDBC database connection properties can be set (...)

Usage

write.jdbc(x, url, tableName, mode = "error", ...)

## S4 method for signature 'SparkDataFrame,character,character'
write.jdbc(x, url, tableName, mode = "error", ...)

Arguments

x a SparkDataFrame.

url JDBC database url of the form jdbc:subprotocol:subname.

tableName yhe name of the table in the external database.
mode 
one of 'append', 'overwrite', 'error', 'errorifexists', 'ignore' save mode (it is 'error' by default)

... 
additional JDBC database connection properties.

Details

Also, mode is used to specify the behavior of the save operation when data already exists in the data source. There are four modes:

- 'append': Contents of this SparkDataFrame are expected to be appended to existing data.
- 'overwrite': Existing data is expected to be overwritten by the contents of this SparkDataFrame.
- 'error' or 'errorifexists': An exception is expected to be thrown.
- 'ignore': The save operation is expected to not save the contents of the SparkDataFrame and to not change the existing data.

Note

write.jdbc since 2.0.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.json(), write.orc(), write.parquet(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
jdbcUrl <- "jdbc:mysql://localhost:3306/databasename"
write.jdbc(df, jdbcUrl, "table", user = "username", password = "password")

## End(Not run)
```
Save the contents of SparkDataFrame as a JSON file

Description

Save the contents of a SparkDataFrame as a JSON file (JSON Lines text format or newline-delimited JSON). Files written out with this method can be read back in as a SparkDataFrame using read.json().

Usage

write.json(x, path, ...)

## S4 method for signature 'SparkDataFrame,character'
write.json(x, path, mode = "error", ...)

Arguments

x  
A SparkDataFrame

path  
The directory where the file is saved

...  
additional argument(s) passed to the method.

mode  
one of 'append', 'overwrite', 'error', 'errorifexists', 'ignore' save mode (it is 'error' by default)

Note

write.json since 1.6.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with().write.df(), write.jdbc(), write.orc(), write.parquet(), write.stream(), write.text()
write.ml

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
write.json(df, "/tmp/sparkr-tmp/"

## End(Not run)
```

---

**write.ml**

*Saves the MLlib model to the input path*

---

**Description**

Saves the MLlib model to the input path. For more information, see the specific MLlib model below.

**Usage**

```r
write.ml(object, path, ...)
```

**Arguments**

- `object`: a fitted ML model object.
- `path`: the directory where the model is saved.
- `...`: additional argument(s) passed to the method.

**See Also**

- `spark.als`, `spark.bisectingKmeans`, `spark.decisionTree`,
- `spark.gaussianMixture`, `spark.gbt`,
- `spark.glm`, `glm`, `spark.isoreg`,
- `spark.kmeans`,
- `spark.lda`, `spark.logit`,
- `spark.mlp`, `spark.naiveBayes`,
- `spark.randomForest`, `spark.survreg`, `spark.svmLinear`,
- `read.ml`
Save the contents of SparkDataFrame as an ORC file, preserving the schema.

**Description**

Save the contents of a SparkDataFrame as an ORC file, preserving the schema. Files written out with this method can be read back in as a SparkDataFrame using read.orc().

**Usage**

write.orc(x, path, ...)

```r
## S4 method for signature 'SparkDataFrame,character'
write.orc(x, path, mode = "error", ...)
```

**Arguments**

- **x**  
  A SparkDataFrame

- **path**  
  The directory where the file is saved

- **...**  
  additional argument(s) passed to the method.

- **mode**  
  one of 'append', 'overwrite', 'error', 'errorifexists', 'ignore' save mode (it is 'error' by default)

**Note**

write.orc since 2.0.0

**See Also**

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with().write.df(), write.jdbc(), write.json(), write.parquet(), write.stream(), write.text()
Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
write.orc(df, "/tmp/sparkr-tmp1/"

## End(Not run)
```

write.parquet  
Save the contents of SparkDataFrame as a Parquet file, preserving the schema.

Description

Save the contents of a SparkDataFrame as a Parquet file, preserving the schema. Files written out with this method can be read back in as a SparkDataFrame using read.parquet().

Usage

```r
write.parquet(x, path, ...)
saveAsParquetFile(x, path)
```

Arguments

- `x`  
  A SparkDataFrame
- `path`  
  The directory where the file is saved
- `...`  
  additional argument(s) passed to the method.
- `mode`  
  one of `"append"`, `"overwrite"`, `"error"`, `"errorexists"`, `"ignore"` save mode (it is `"error"` by default)

Note

write.parquet since 1.6.0
saveAsParquetFile since 1.4.0
See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach, SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.stream(), write.text()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.json"
df <- read.json(path)
write.parquet(df, "/tmp/sparkr-tmp1/"
saveAsParquetFile(df, "/tmp/sparkr-tmp2/"
## End(Not run)
```

---

`write.stream`  
*Write the streaming SparkDataFrame to a data source.*

Description

The data source is specified by the source and a set of options (...). If source is not specified, the default data source configured by spark.sql.sources.default will be used.

Usage

```r
write.stream(df, source = NULL, outputMode = NULL, ...)  
```

## S4 method for signature 'SparkDataFrame'

`write.stream`  
```r
  df,
  source = NULL,
  outputMode = NULL,
  partitionBy = NULL,
  trigger.processingTime = NULL,
  trigger.once = NULL,
  ...  
)```
write.stream

Arguments

- `df`: a streaming SparkDataFrame.
- `source`: a name for external data source.
- `outputMode`: one of 'append', 'complete', 'update'.
- `partitionBy`: a name or a list of names of columns to partition the output by on the file system. If specified, the output is laid out on the file system similar to Hive's partitioning scheme.
- `trigger.processingTime`: a processing time interval as a string, e.g. '5 seconds', '1 minute'. This is a trigger that runs a query periodically based on the processing time. If value is '0 seconds', the query will run as fast as possible, this is the default. Only one trigger can be set.
- `trigger.once`: a logical, must be set to TRUE. This is a trigger that processes only one batch of data in a streaming query then terminates the query. Only one trigger can be set.

Details

Additionally, `outputMode` specifies how data of a streaming SparkDataFrame is written to an output data source. There are three modes:

- **append**: Only the new rows in the streaming SparkDataFrame will be written out. This output mode can be only be used in queries that do not contain any aggregation.
- **complete**: All the rows in the streaming SparkDataFrame will be written out every time there are some updates. This output mode can only be used in queries that contain aggregations.
- **update**: Only the rows that were updated in the streaming SparkDataFrame will be written out every time there are some updates. If the query doesn’t contain aggregations, it will be equivalent to append mode.

Note

write.stream since 2.2.0
experimental

See Also

read.stream

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach,SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(),...
write.text

Save the content of SparkDataFrame in a text file at the specified path.

Description

Save the content of the SparkDataFrame in a text file at the specified path. The SparkDataFrame must have only one column of string type with the name "value". Each row becomes a new line in the output file.

Usage

write.text(x, path, ...)

## S4 method for signature 'SparkDataFrame,character'
write.text(x, path, mode = "error", ...)

Arguments

- x: A SparkDataFrame
- path: The directory where the file is saved
- mode: one of 'append', 'overwrite', 'error', 'errorifexists', 'ignore' save mode (it is 'error' by default)
Note

write.text since 2.0.0

See Also

Other SparkDataFrame functions: SparkDataFrame-class, agg(), alias(), arrange(), as.data.frame(), attach(), SparkDataFrame-method, broadcast(), cache(), checkpoint(), coalesce(), collect(), colnames(), coltypes(), createOrReplaceTempView(), crossJoin(), cube(), dapplyCollect(), dapply(), describe(), dim(), distinct(), dropDuplicates(), dropna(), drop(), dtypes(), exceptAll(), except(), explain(), filter(), first(), gapplyCollect(), gapply(), getNumPartitions(), group_by(), head(), hint(), histogram(), insertInto(), intersectAll(), intersect(), isLocal(), isStreaming(), join(), limit(), localCheckpoint(), merge(), mutate(), ncol(), nrow(), persist(), printSchema(), randomSplit(), rbind(), rename(), repartitionByRange(), repartition(), rollup(), sample(), saveAsTable(), schema(), selectExpr(), select(), showDF(), show(), storageLevel(), str(), subset(), summary(), take(), toJSON(), unionByName(), union(), unpersist(), withColumn(), withWatermark(), with(), write.df(), write.jdbc(), write.json(), write.orc(), write.parquet(), write.stream()

Examples

```r
## Not run:
sparkR.session()
path <- "path/to/file.txt"
df <- read.text(path)
write.text(df, "/tmp/sparkr-tmp/"

## End(Not run)
```

Description

Equality test that is safe for null values.

Usage

```r
x %<=>% value
```

## S4 method for signature 'Column'
```r
x %<=>% value
```

Arguments

- `x` a Column
- `value` a value to compare
Details

Can be used, unlike standard equality operator, to perform null-safe joins. Equivalent to Scala `Column.<=>` and `Column.eqNullSafe`.

Note

`%<=>%` since 2.3.0

Examples

```r
## Not run:
df1 <- createDataFrame(data.frame(
  x = c(1, NA, 3, NA), y = c(2, 6, 3, NA)
))

head(select(df1, df1$x == df1$y, df1$x %<=>% df1$y))

df2 <- createDataFrame(data.frame(y = c(3, NA)))
count(join(df1, df2, df1$y == df2$y))
count(join(df1, df2, df1$y %<=>% df2$y))
## End(Not run)
```

%in%

*Match a column with given values.*

Description

Match a column with given values.

Usage

```r
## S4 method for signature 'Column'
x %in% table
```

Arguments

- `x` a Column.
- `table` a collection of values (coercible to list) to compare with.

Value

A matched values as a result of comparing with given values.

Note

`%in%` since 1.5.0
Examples

```r
## Not run:
filter(df, "age in (10, 30)")
where(df, df$age %in% c(10, 30))

## End(Not run)
```
Index

! (subset), 261
(Subset), 261
[[, SparkDataFrame-method (subset), 261
[[, SparkDataFrame, numericOrcharacter-method (subset), 261
[[-, (subset), 261
[[-, SparkDataFrame, numericOrcharacter-method (subset), 261
$(select), 188
$, SparkDataFrame-method (select), 188
$-, (select), 188
$-, SparkDataFrame-method (select), 188
%==%, Column-method (%==%), 291
%==%, Column-method (%==%), 291
%in%, Column-method (%in%), 292
%in%, Column-method (%in%), 292
abs (column_math_functions), 54
abs, Column-method (column_math_functions), 54
acos (column_math_functions), 54
acos, Column-method (column_math_functions), 54
add_months (column_datetime_diff_functions), 45
add_months, Column, numeric-method (column_datetime_diff_functions), 45
AFTSurvivalRegressionModel-class, 7
agg, GroupedData-method (agg), 7
agg, SparkDataFrame-method (agg), 7
alias, Column-method (alias), 9
alias, SparkDataFrame-method (alias), 9
ALSModel-class, 10
approxCountDistinct (column_aggregate_functions), 32
approxCountDistinct, Column-method (column_aggregate_functions), 32
approxQuantile, 10, 79, 81, 87, 112, 185
approxQuantile, SparkDataFrame, character, numeric, numeric-method (approxQuantile), 10
arrange, SparkDataFrame, character-method (arrange), 11
arrange, SparkDataFrame, Column-method

294
INDEX

(arrange), 11
array_contains
   (column_collection_functions), 37
array_contains, Column-method
   (column_collection_functions), 37
array_distinct
   (column_collection_functions), 37
array_distinct, Column-method
   (column_collection_functions), 37
array_except
   (column_collection_functions), 37
array_except, Column, Column-method
   (column_collection_functions), 37
array_except, Column-method
   (column_collection_functions), 37
array_intersect
   (column_collection_functions), 37
array_intersect, Column, Column-method
   (column_collection_functions), 37
array_intersect, Column-method
   (column_collection_functions), 37
array_join
   (column_collection_functions), 37
array_join, Column, character-method
   (column_collection_functions), 37
array_join, Column-method
   (column_collection_functions), 37
array_max
   (column_collection_functions), 37
array_max, Column-method
   (column_collection_functions), 37
array_min
   (column_collection_functions), 37
array_min, Column-method
   (column_collection_functions), 37
array_position
   (column_collection_functions), 37
array_position, Column-method
   (column_collection_functions), 37
array_remove
   (column_collection_functions), 37
array_remove, Column-method
   (column_collection_functions), 37
array_repeat
   (column_collection_functions), 37
array_repeat, Column, numericOrColumn-method
   (column_collection_functions), 37
array_sort
   (column_collection_functions), 37
array_sort, Column-method
   (column_collection_functions), 37
array_union
   (column_collection_functions), 37
array_union, Column, Column-method
   (column_collection_functions), 37
array_union, Column-method
   (column_collection_functions), 37
arrays_overlap
   (column_collection_functions), 37
arrays_overlap, Column, Column-method
   (column_collection_functions), 37
arrays_overlap, Column-method
   (column_collection_functions), 37
arrays_zip
   (column_collection_functions),
INDEX

arrays_zip, Column-method
  (column_collection_functions), 37
as.data.frame, 8, 9, 12, 13, 16, 19, 20, 24,
  26, 27, 29, 30, 84, 86, 88, 90, 92,
  94–96, 98–100, 103, 105–108, 110,
  114, 116, 121, 124, 126–129,
  132–134, 136, 137, 141, 145, 147,
  149, 151, 153, 157, 161, 164, 166,
  178, 180–182, 184, 187–190, 196,
  197, 244, 257, 258, 262, 265, 268,
  269, 271, 273, 274, 277, 278, 280,
  282–284, 286, 288, 289, 291
as.data.frame, SparkDataFrame-method
  (as.data.frame), 13
as.DataFrame (createDataFrame), 82
asc, 14
asci (column_string_functions), 68
asci, Column-method
  (column_string_functions), 68
asin (column_math_functions), 54
asin, Column-method
  (column_math_functions), 54
associationRules, FPGrowthModel-method
  (spark. fpGrowth), 206
atan (column_math_functions), 54
atan, Column-method
  (column_math_functions), 54
atan2 (column_math_functions), 54
atan2, Column-method
  (column_math_functions), 54
attach, 277
attach (attach, SparkDataFrame-method), 15
attach, SparkDataFrame-method, 15
avg, 16, 36, 79–81, 110, 139
avg, Column-method (avg), 16
awaitTermination, 17, 107, 134, 140, 162,
  256
awaitTermination, StreamingQuery-method
  (awaitTermination), 17
base64 (column_string_functions), 68
base64, Column-method
  (column_string_functions), 68
between, 9, 18, 23, 104, 155, 255, 263
between, Column-method (between), 18
bin (column_math_functions), 54
bin, Column-method
  (column_math_functions), 54
BitsectingKMeansModel-class, 18
bitwiseNOT
  (column_nonaggregate_functions), 62
bitwiseNOT, Column-method
  (column_nonaggregate_functions), 62
broadcast, 8, 9, 12, 14, 16, 19, 20, 24, 26, 27,
  29, 30, 84, 86, 88, 90, 92, 94–96,
  98–100, 103, 105–108, 110, 114,
  116, 121, 124, 126–129, 132–134,
  136, 137, 141, 145, 147, 149, 151,
  153, 157, 161, 164, 166, 178,
  180–182, 184, 187–190, 196, 197,
  244, 257, 258, 262, 265, 268, 269,
  271, 273, 274, 277, 278, 280,
  282–284, 286, 288, 289, 291
broadcast, SparkDataFrame-method
  (broadcast), 19
bround (column_math_functions), 54
bround, Column-method
  (column_math_functions), 54
cache, 8, 9, 12, 14, 16, 19, 20, 24, 26, 27, 29,
  30, 84, 86, 88, 90, 92, 94–96,
  98–100, 103, 105–108, 110, 114,
  116, 121, 124, 126–129, 132–134,
  136, 137, 141, 145, 147, 149, 151,
  153, 157, 161, 164, 166, 178,
  180–182, 184, 187–190, 196, 197,
  244, 257, 258, 262, 265, 268, 269,
  271, 273, 274, 277, 278, 280,
  282–284, 286, 288, 289, 291
cache, SparkDataFrame-method (cache), 20
cacheTable, 21
cancelJobGroup, 22
cast, 9, 18, 22, 104, 155, 255, 263
cast, Column-method (cast), 22
cbtrt (column_math_functions), 54
cbrtrt, Column-method
  (column_math_functions), 54
ceil (column_math_functions), 54
ceil, Column-method
  (column_math_functions), 54
ceiling (column_math_functions), 54
ceiling, Column-method
  (column_math_functions), 54
INDEX


collect_list, Column-method (collect_list), 26

collect_list, Column-method (collect_list), 32

collect_set, Column-method (collect), 25

collect_set, Column-method (collect), 29

collect_set, Column-method (collect), 32


colnames, SparkDataFrame-method (colnames), 28

colnames, SparkDataFrame-method (colnames), 28
colnames<-(colnames), 28
colnames<-, SparkDataFrame-method (colnames), 28
coltypes, SparkDataFrame-method (coltypes), 29
coltypes<-(coltypes), 29
coltypes<-, SparkDataFrame-method (coltypes), 29
column, 31, 67, 152
column, character-method (column), 31
column, obj-method (column), 31
Column-class (column), 31
column_aggregate_functions, 17, 32, 79–81, 110, 139
column_collection_functions, 37
column_datetime_diff_functions, 45, 53
column_datetime_functions, 47, 48
column_math_functions, 54
column_misc_functions, 61
column_nonaggregate_functions, 31, 62, 152
column_string_functions, 68
column_window_functions, 75
columnfunctions (asc), 14
columns (colnames), 28
columns, SparkDataFrame-method (colnames), 28
concat (column_collection_functions), 37
concat, Column-method
  (column_collection_functions), 37
concat_ws (column_string_functions), 68
concat_ws, character, Column-method
contains (asc), 14
conv (column_math_functions), 54
conv, Column, numeric, numeric-method
  (column_math_functions), 54
corr, 11, 17, 36, 78, 80, 81, 87, 110, 112, 139, 185
corr, Column-method (corr), 78
corr, SparkDataFrame-method (corr), 78
cos (column_math_functions), 54
cos, Column-method
  (column_math_functions), 54
cosh (column_math_functions), 54
cosh, Column-method
  (column_math_functions), 54
count, 17, 36, 79, 79, 81, 110, 139
count, Column-method (count), 79
count, GroupedData-method (count), 79
count, SparkDataFrame-method (nrow), 152
countDistinct
  (column_aggregate_functions), 32
countDistinct, Column-method
  (column_aggregate_functions), 32
cov, 11, 17, 36, 79, 80, 80, 87, 110, 112, 139, 185
cov, characterOrColumn-method (cov), 80
cov, SparkDataFrame-method (cov), 80
covar_pop (cov), 80
covar_pop, characterOrColumn-method (cov), 80
covar_samp (cov), 80
covar_samp, characterOrColumn-method (cov), 80
crc32 (column_misc_functions), 61
crc32, Column-method
  (column_misc_functions), 61
create_array
  (column_nonaggregate_functions), 62
create_array, Column-method
  (column_nonaggregate_functions), 62
create_map
  (column_nonaggregate_functions), 62
create_map, Column-method
  (column_nonaggregate_functions), 62
createDataFrame, 82, 243, 244
crateExternalTable, 83
createOrReplaceTempView, SparkDataFrame, character-method
  (createOrReplaceTempView), 84
createTable, 83, 85
crossJoin, SparkDataFrame, SparkDataFrame-method (crossJoin), 86
crosstab, SparkDataFrame, character, character-method (crosstab), 87
dapply, SparkDataFrame, function, characterOrstructType-method (dapply), 90
dapplyCollect, SparkDataFrame, function-method (dapplyCollect), 91
data.frame, 113, 116
date_add (column_datetime_diff_functions), 45
date_add, Column, numeric-method (column_datetime_diff_functions), 45
date_format (column_datetime_diff_functions), 45
date_format, Column, character-method (column_datetime_diff_functions), 45
date_sub (column_datetime_diff_functions), 45
date_sub, Column, numeric-method (column_datetime_diff_functions), 45
date_trunc (column_datetime_diff_functions), 48
date_trunc, character, Column-method (column_datetime_diff_functions), 48
datediff (column_datetime_diff_functions), 45
datediff, Column-method (column_datetime_diff_functions), 45
dayofmonth (column_datetime_diff_functions), 48
dayofmonth, Column-method (column_datetime_diff_functions), 48
dayofweek (column_datetime_diff_functions), 48
dayofweek, Column-method (column_datetime_diff_functions), 48
DecisionTreeClassificationModel-class, 93
DecisionTreeRegressionModel-class, 93
decode (column_string_functions), 68
decode, Column, character-method (column_string_functions), 68
dense_rank (column_window_functions), 75
dense_rank, missing-method (column_window_functions), 75
desc (asc), 14

dayofmonth, Column-method (column_datetime_diff_functions), 48
dayofyear (column_datetime_diff_functions), 48
dayofyear, Column-method (column_datetime_diff_functions), 48

date_add, Column, numeric-method (column_datetime_diff_functions), 45
date_format (column_datetime_diff_functions), 45
date_format, Column, character-method (column_datetime_diff_functions), 45
date_sub (column_datetime_diff_functions), 45
date_trunc (column_datetime_diff_functions), 48
date_trunc, character, Column-method (column_datetime_diff_functions), 48
datediff (column_datetime_diff_functions), 45
datediff, Column-method (column_datetime_diff_functions), 45
dayofmonth (column_datetime_diff_functions), 48
dayofmonth, Column-method (column_datetime_diff_functions), 48
dayofweek (column_datetime_diff_functions), 48
dayofweek, Column-method (column_datetime_diff_functions), 48
DecisionTreeClassificationModel-class, 93
DecisionTreeRegressionModel-class, 93
decode (column_string_functions), 68
decode, Column, character-method (column_string_functions), 68
dense_rank (column_window_functions), 75
dense_rank, missing-method (column_window_functions), 75
desc (asc), 14
describe, 8, 9, 12, 14, 16, 19, 20, 24, 26, 27,
INDEX

distinct, SparkDataFrame-method (distinct), 96
drop, SparkDataFrame-method (drop), 97

drop, ANY-method (drop), 97
drop, SparkDataFrame-method (drop), 97
dropDuplicates, SparkDataFrame-method (dropDuplicates), 98
dropna, SparkDataFrame-method (dropna), 99
dropTempTable, 101
dropTempView, 101, 102
dtypes, SparkDataFrame-method (dtyps), 103
element_at, (column_collection_functions), 37
element_at, Column-method (column_collection_functions), 37
encode (column_string_functions), 68
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page References</th>
</tr>
</thead>
<tbody>
<tr>
<td>except, Column, character-method</td>
<td>68</td>
</tr>
<tr>
<td>except, SparkDataFrame, SparkDataFrame-method (except)</td>
<td>104</td>
</tr>
<tr>
<td>exceptAll, SparkDataFrame, SparkDataFrame-method (exceptAll)</td>
<td>104</td>
</tr>
<tr>
<td>exp (column_math_functions), 54</td>
<td></td>
</tr>
<tr>
<td>exp, Column-method (column_math_functions), 54</td>
<td></td>
</tr>
<tr>
<td>explain, SparkDataFrame-method (explain)</td>
<td>106</td>
</tr>
<tr>
<td>explain, StreamingQuery-method (explain)</td>
<td>106</td>
</tr>
<tr>
<td>explode (column_collection_functions)</td>
<td>37</td>
</tr>
<tr>
<td>explode, Column-method (column_collection_functions), 37</td>
<td></td>
</tr>
<tr>
<td>explode, outer (column_collection_functions), 37</td>
<td></td>
</tr>
<tr>
<td>explode, outer, Column-method (column_collection_functions), 37</td>
<td></td>
</tr>
<tr>
<td>expm1 (column_math_functions), 54</td>
<td></td>
</tr>
<tr>
<td>expm1, Column-method (column_math_functions), 54</td>
<td></td>
</tr>
<tr>
<td>expr (column_nonaggregates_functions), 62</td>
<td></td>
</tr>
<tr>
<td>expr, character-method (column_nonaggregate_functions), 62</td>
<td></td>
</tr>
<tr>
<td>factorial (column_math_functions), 54</td>
<td></td>
</tr>
<tr>
<td>factorial, Column-method (column_math_functions), 54</td>
<td></td>
</tr>
<tr>
<td>fillna (dropna), 99</td>
<td></td>
</tr>
<tr>
<td>fillna, SparkDataFrame-method (dropna)</td>
<td>99</td>
</tr>
<tr>
<td>filter, SparkDataFrame, characterOrColumn-method (filter)</td>
<td>108</td>
</tr>
<tr>
<td>first, characterOrColumn-method (first), 109</td>
<td></td>
</tr>
<tr>
<td>first, SparkDataFrame-method (first)</td>
<td>109</td>
</tr>
<tr>
<td>fitted, 110</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Pages</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>install.spark</code></td>
<td>130</td>
</tr>
<tr>
<td><code>instr</code></td>
<td>(column_string_functions), 68</td>
</tr>
</tbody>
</table>

**Note:** The above list contains a mix of method names and page numbers, indicating the locations within the document where these methods and terms are discussed. This list is not exhaustive and may not cover all occurrences within the document.
INDEX

KMeansModel-class, 138
KSTest-class, 138
kurtosis (column_aggregate_functions), 32
kurtosis, Column-method (column_aggregate_functions), 32
lag (column_window_functions), 75
lag, characterOrColumn-method (column_window_functions), 75
last, 17, 36, 79–81, 110, 138
last, characterOrColumn-method(last), 138
last_day (column_datetime_functions), 48
last_day, Column-method (column_datetime_functions), 48
lastProgress, 17, 107, 134, 139, 162, 256
lastProgress, StreamingQuery-method (lastProgress), 139
LDAModel-class, 140
lead (column_window_functions), 75
lead, characterOrColumn, numeric-method (column_window_functions), 75
least (column_nonaggregate_functions), 62
least, Column-method (column_nonaggregate_functions), 62
length (column_string_functions), 68
length, Column-method (column_string_functions), 68
levenshtein (column_string_functions), 68
levenshtein, Column-method (column_string_functions), 68
like (asc), 14
limit, SparkDataFrame, numeric-method (limit), 140
LinearSVCModel-class, 141
listColumns, 142
listDatabases, 142
listFunctions, 143
listTables, 144, 266
lit (column_nonaggregate_functions), 62
lit, ANY-method (column_nonaggregate_functions), 62
loadDF (read.df), 167
localCheckpoint, SparkDataFrame-method (localCheckpoint), 144
locate (column_string_functions), 68
locate, character, Column-method (column_string_functions), 68
log (column_math_functions), 54
log, Column-method (column_math_functions), 54
log10 (column_math_functions), 54
log10, Column-method (column_math_functions), 54
log1p (column_math_functions), 54
log1p, Column-method (column_math_functions), 54
log2 (column_math_functions), 54
log2, Column-method (column_math_functions), 54
LogisticRegressionModel-class, 146
lower (column_string_functions), 68
lower, Column-method (column_string_functions), 68
lpad (column_string_functions), 68
lpad, Column, numeric, character-method (column_string_functions), 68
ltrim (column_string_functions), 68
ltrim, Column, character-method (column_string_functions), 68
ltrim, Column, missing-method
map_values
(column_collection_functions), 37
map_values,Column-method
(column_collection_functions), 37
map_keys,Column-method
(column_collection_functions), 37
map_keys (column_collection_functions), 37
map_keys,Column-method
(column_collection_functions), 37
map_from_arrays(Column-method
(column_collection_functions), 37
map_from_arrays,Column,Column-method
(column_collection_functions), 37
min (column_aggregate_functions), 32
min,Column-method
(column_aggregate_functions), 32
merge,SparkDataFrame-method (merge), 146
mean (column_aggregate_functions), 32
mean,Column-method
(column_aggregate_functions), 32
minutes_between,Column-method
(column_aggregate_functions), 48
n (count), 79
n,Column-method (count), 79
n_distinct
(column_aggregate_functions), 32
n_distinct,Column-method
(column_aggregate_functions), 32
na.omit (dropna), 99
na.omit,SparkDataFrame-method (dropna), 99
NaiveBayesModel-class, 150
names (colnames), 28
names,SparkDataFrame-method (colnames), 28
names<- (colnames), 28
MultilayerPerceptronClassificationModel-class, 148
names<-(names,SparkDataFrame-method
(colnames), 61
months between
(column_datetime_diff_functions), 45
months_between,Column-method
(column_datetime_diff_functions), 45
merge,SparkDataFrame,SparkDataFrame-method
(merge), 146
min (column_aggregate_functions), 32
min,Column-method
(column_aggregate_functions), 32
predict,AFTSurvivalRegressionModel-method (spark.survreg), 239
predict,ALSModel-method (spark.als), 198
predict,BisectingKMeansModel-method (spark.bisectingKmeans), 201
predict,DecisionTreeClassificationModel-method (spark.decisionTree), 203
predict,DecisionTreeRegressionModel-method (spark.decisionTree), 203
predict,FPGrowthModel-method (spark.fpGrowth), 206
predict,GaussianMixtureModel,SparkDataFrame-method (spark.gaussianMixture), 208
predict,GaussianMixtureModel-method (spark.gaussianMixture), 208
predict,GBTClassificationModel-method (spark.gbt), 210
predict,GBTRegressionModel-method (spark.gbt), 210
predict,GeneralizedLinearRegressionModel-method (spark.glm), 215
predict,IsotonicRegressionModel,SparkDataFrame-method (spark.isoreg), 218
predict,IsotonicRegressionModel-method (spark.isoreg), 218
predict,KMeansModel-method (spark.kmeans), 220
predict,LinearSVCModel,SparkDataFrame-method (spark.svmLinear), 241
predict,LinearSVCModel-method (spark.svmLinear), 241
predict,LogisticRegressionModel,SparkDataFrame-method (spark.logit), 228
predict,LogisticRegressionModel-method (spark.logit), 228
predict,MultiLayerPerceptronClassificationModel-method (spark.mlp), 231
predict,NaiveBayesModel-method (spark.naiveBayes), 234
predict,RandomForestClassificationModel-method (spark.randomForest), 236
predict,RandomForestRegressionModel-method (spark.randomForest), 236
print.jobj, 159
print.structField, 160
print.structType, 160
print.summary.DecisionTreeClassificationModel (spark.decisionTree), 203
print.summary.GBTClassificationModel (spark.gbt), 210
print.summary.GBTRegressionModel (spark.gbt), 210
print.summary.GeneralizedLinearRegressionModel (spark.glm), 215
print.summary.KSTest (spark.kstest), 222
print.summary.RandomForestClassificationModel (spark.randomForest), 236
print.summary.RandomForestRegressionModel (spark.randomForest), 236
printSchema, SparkDataFrame-method (printSchema), 161
quarter (column_datetime_functions), 48
queryName, 17, 107, 134, 140, 162, 256
queryName, StreamingQuery-method
rand (column_nonaggregate_functions), 62
rand, missing-method
rand, numeric-method
degreg (column_nonaggregate_functions), 62
random, missing-method
random, numeric-method
print.summary.DecisionTreeRegressionModel (spark.decisionTree), 203
print.summary.RandomForestRegressionModel (spark.randomForest), 236
printSchema, SparkDataFrame-method (printSchema), 161
### INDEX

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>RandomForestClassificationModel-class</td>
<td>163</td>
</tr>
<tr>
<td>RandomForestRegressionModel-class</td>
<td>163</td>
</tr>
<tr>
<td>rank</td>
<td>75</td>
</tr>
<tr>
<td>rank, ANY-method</td>
<td>75</td>
</tr>
<tr>
<td>rank, missing-method</td>
<td>75</td>
</tr>
<tr>
<td>rbind, SparkDataFrame-method (rbind)</td>
<td>166</td>
</tr>
<tr>
<td>read df</td>
<td>167</td>
</tr>
<tr>
<td>read.jdbc</td>
<td>168</td>
</tr>
<tr>
<td>read.json</td>
<td>169, 243, 244</td>
</tr>
<tr>
<td>read.ml</td>
<td>170, 200, 202, 207, 209, 217, 221, 227, 233, 285</td>
</tr>
<tr>
<td>read.orc</td>
<td>171</td>
</tr>
<tr>
<td>read.parquet</td>
<td>172</td>
</tr>
<tr>
<td>read.stream</td>
<td>136, 172, 259, 289</td>
</tr>
<tr>
<td>read.text</td>
<td>174</td>
</tr>
<tr>
<td>recoverPartitions</td>
<td>175</td>
</tr>
<tr>
<td>refreshByPath</td>
<td>175</td>
</tr>
<tr>
<td>refreshTable</td>
<td>176</td>
</tr>
<tr>
<td>regexp_extract (column_string_functions)</td>
<td>68</td>
</tr>
<tr>
<td>regexp_replace</td>
<td>68</td>
</tr>
<tr>
<td>regexp_replace (column_string_functions)</td>
<td>68</td>
</tr>
<tr>
<td>registerTempTable</td>
<td>177</td>
</tr>
<tr>
<td>registerTempTable, SparkDataFrame-method (rename)</td>
<td>178</td>
</tr>
<tr>
<td>repartition (repartition)</td>
<td>179</td>
</tr>
<tr>
<td>repeat_string</td>
<td>68</td>
</tr>
<tr>
<td>repeat_string, Column, numeric-method</td>
<td>68</td>
</tr>
</tbody>
</table>
INDEX

310

(reverse (column_collection_functions), 37)

repeat_string, Column-method (column_string_functions), 68

reverse (column_collection_functions), 37

reverse, Column-method (column_string_functions), 37

rint (column_math_functions), 54

rint, Column-method (column_math_functions), 54

rlike (asc), 14


rollup, SparkDataFrame-method (rollup), 181

round (column_math_functions), 54

round, Column-method (column_math_functions), 54

row_number (column_window_functions), 75

row_number, missing-method (column_window_functions), 75

rowsBetween, 154, 156, 165, 183

rowsBetween, WindowSpec, numeric, numeric-method (rowsBetween), 183

rpad (column_string_functions), 68

rpad, Column, numeric, character-method (column_string_functions), 68

rtrim (column_string_functions), 68

rtrim, Column, character-method (column_string_functions), 68

rtrim, Column, missing-method (column_string_functions), 68


saveAsTable, SparkDataFrame, character-method (saveAsTable), 186

saveAsTable, SparkDataFrame, character-method (write.parquet), 287

saveAsTable, SparkDataFrame, character-method (write.parquet), 287

saveAsTable, SparkDataFrame, character-method (saveAsTable), 186

saveDF (write.df), 281

saveDF, SparkDataFrame, character-method (write.df), 281


schema, SparkDataFrame-method (schema), 187

sd (column_aggregate_functions), 32

sd, Column-method (column_aggregate_functions), 32

second (column_datetime_functions), 48

second, Column-method (column_datetime_functions), 48
INDEX


shiftRightUnsigned, Column, numeric-method (column_math_functions), 54


shuffle (column_collection_functions), 37

showDataFrame, character-method (select), 188

select, SparkDataFrame, character-method (select), 188

select, SparkDataFrame, Column-method (select), 188

select, SparkDataFrame, list-method (select), 188


selectExpr, SparkDataFrame, character-method (selectExpr), 190

setCheckpointDir, 24, 191

setCurrentDatabase, 192

setJobDescription, 192

setJobGroup, 193

setLocalProperty, 194

setLoglevel, 194

sha1 (column_misc_functions), 61

sha1, Column-method (column_misc_functions), 61

sha2 (column_misc_functions), 61

sha2, Column, numeric-method (column_misc_functions), 61

shiftLeft (column_math_functions), 54

shiftLeft, Column, numeric-method (column_math_functions), 54

shiftRight (column_math_functions), 54

shiftRight, Column, numeric-method (column_math_functions), 54

shiftRightUnsigned (column_math_functions), 54

sign (column_math_functions), 54

sign, Column-method (column_math_functions), 54

signum (column_math_functions), 54

signum, Column-method (column_math_functions), 54

sin (column_math_functions), 54

sin, Column-method (column_math_functions), 54

sinh (column_math_functions), 54

sinh, Column-method (column_math_functions), 54

size (column_collection_functions), 37

size, Column-method
toDegrees (column_math_functions), 54
unhex (column_math_functions), 54
union, SparkDataFrame, SparkDataFrame-method (union), 271
unionAll (union), 271
unionAll, SparkDataFrame, SparkDataFrame-method (union), 271
unionByName, SparkDataFrame, SparkDataFrame-method (unionByName), 272
unique (distinct), 96
unique, SparkDataFrame-method (distinct), 96
unix_timestamp (column_datetime_functions), 48
unix_timestamp, Column, character-method (column_datetime_functions), 48
unix_timestamp, Column, missing-method (column_datetime_functions), 48
unix_timestamp, Column, character-method (column_datetime_functions), 48
unix_timestamp, Column, missing-method (column_datetime_functions), 48
unhex, Column-method (column_math_functions), 54
update, SparkDataFrame-method (union), 271
updateAll (union), 271
updateAll, SparkDataFrame, SparkDataFrame-method (union), 271
updateByName, SparkDataFrame, SparkDataFrame-method (updateByName), 272
toDegrees, Column-method (column_math_functions), 54
toDegrees, SparkDataFrame-method (toJSON), 269
toRadians (column_math_functions), 54
toRadians, Column-method (column_math_functions), 54
transform (mutate), 148
transform, SparkDataFrame-method (mutate), 148
translate (column_string_functions), 68
translate, Column, character, character-method (column_string_functions), 68
trim (column_string_functions), 68
trim, Column, character-method (column_string_functions), 68
trim, Column, missing-method (column_string_functions), 68
trunc (column_datetime_functions), 48
trunc, Column-method (column_datetime_functions), 48
unbase64 (column_string_functions), 68
unbase64, Column-method (column_string_functions), 68
uncacheTable, 270
unpersist, SparkDataFrame-method (unpersist), 273
upper, Column-method (column_string_functions), 68

var, Column-method (column_aggregate_functions), 32

var_pop, Column-method (column_aggregate_functions), 32

var_pop, Column-method (column_aggregate_functions), 32

var_samp, Column-method (column_aggregate_functions), 32

var_samp, Column-method (column_aggregate_functions), 32

variance, Column-method (column_aggregate_functions), 32

variance, Column-method (column_aggregate_functions), 32

weekofyear, column_datetime_functions, 48

weekofyear, Column-method (column_datetime_functions), 48

when, column_nonaggregate_functions, 62

when, Column-method (column_nonaggregate_functions), 62

where, filter, 108

where, SparkDataFrame, characterOrColumn-method (filter), 108

window, column_datetime Functions, 48

window, Column-method (column_datetime_functions), 48

windowOrderBy, 274, 276

windowOrderBy, character-method (windowOrderBy), 274

windowOrderBy, Column-method (windowOrderBy), 274

windowPartitionBy, 275, 276

windowPartitionBy, character-method (windowPartitionBy), 275


WindowSpec-class, 276


write.df, 8, 9, 12, 14, 16, 19, 20, 24, 26, 27, 29, 30, 34, 87, 89, 90, 92, 94–96, 98, 99, 101, 103, 105–107, 109, 110, 114, 117, 121, 124, 126–129, 132–134, 136, 137, 141, 145, 147, 149, 151, 153, 157, 161, 164, 166, 179–182, 184, 187–190, 196, 197,
INDEX

244, 257, 258, 262, 265, 268, 269, 272–274, 277, 279, 280, 281, 283, 284, 286, 288, 290, 291
write.df, SparkDataFrame, method
(write.df), 281
write.jdbc, SparkDataFrame, method
(write.jdbc), 282
write.json, SparkDataFrame, method
(write.json), 284
write.ml, AFTSurvivalRegressionModel, method
(spark.survreg), 239
write.ml, ALSModel, method
(spark.als), 198
write.ml, BisectingKMeansModel, method
(spark.bisectingKmeans), 201
write.ml, DecisionTreeClassificationModel, method
(spark.decisionTree), 203
write.ml, DecisionTreeRegressionModel, method
(spark.decisionTree), 203
write.ml, FPGrowthModel, method
(spark.fpgrowth), 206
write.ml, GaussianMixtureModel, method
write.parquet, SparkDataFrame, method
(write.parquet), 287
write.ml, GBTClassificationModel, method
(spark.gbt), 210
write.ml, GBTRegressionModel, method
98, 99, 101, 103, 105–107, 109, 110,
write.ml, GeneralizedLinearRegressionModel, method
(spark.glm), 215
write.ml, IsotonicRegressionModel, method
(write.isoreg), 218
write.ml, KMeansModel, method
(spark.kmeans), 220
write.ml, LDAModel, method
(spark.lda), 225
write.ml, LinearSVCModel, method
(write.svmLinear), 241
write.ml, LogisticRegressionModel, method
(write.logit), 228
write.ml, MultilayerPerceptronClassificationModel, method
(spark.mlp), 231
write.ml, NaiveBayesModel, method
(spark.naiveBayes), 234
write.ml, RandomForestClassificationModel, method
(spark.randomForest), 236
write.ml, RandomForestRegressionModel, method
(spark.randomForest), 236
write.parquet, SparkDataFrame, method
(write.parquet), 287
write.parquet, SparkDataFrame, method
(write.parquet), 287
write.parquet, SparkDataFrame, method
(write.parquet), 287
write.parquet, SparkDataFrame, method
(write.parquet), 287
write.stream, SparkDataFrame-method
  (write.stream), 288
write.text, 8, 9, 12, 14, 16, 19, 20, 24, 26,
  27, 29, 30, 84, 87, 89, 90, 92, 94–96,
  98, 99, 101, 103, 105–107, 109, 110,
  114, 117, 121, 124, 126–129,
  132–134, 136, 137, 141, 145, 147,
  149, 151, 153, 157, 161, 162, 164,
  166, 173, 179–182, 184, 187–190,
  196, 197, 244, 257, 258, 262, 265,
  268, 269, 272–274, 277, 279, 280,
  282–284, 286, 288, 288, 291
write.text, SparkDataFrame, character-method
  (write.text), 290

year (column_datetime_functions), 48
year, Column-method
  (column_datetime_functions), 48