Search the docs ...

Spark SQL

Core Classes

SparkSession(sparkContext[, jsparkSession])	The entry point to programming Spark with the Dataset and DataFrame API.
DataFrame(jdf, sql_ctx)	A distributed collection of data grouped into named columns.
Column(jc)	A column in a DataFrame.
Row	A row in **DataFrame**.
GroupedData(jgd, df)	A set of methods for aggregations on a **DataFrame**, created by **DataFrame.groupBy()**.
PandasCogroupedOps(gd1, gd2)	A logical grouping of two **GroupedData**, created by **GroupedData.cogroup()**.
DataFrameNaFunctions(df)	Functionality for working with missing data in **DataFrame**.
DataFrameStatFunctions(df)	Functionality for statistic functions with **DataFrame**.
Window	Utility functions for defining window in DataFrames.

Spark Session APIs

The entry point to programming Spark with the Dataset and DataFrame API. To create a Spark session, you should use `SparkSession.builder` attribute. See also **SparkSession**.

SparkSession.builder.appName(name)	Sets a name for the application, which will be shown in the Spark web UI.
SparkSession.builder.config([key, value, conf])	Sets a config option.
SparkSession.builder.enableHiveSupport()	Enables Hive support, including connectivity to a persistent Hive metastore, support for Hive SerDes, and Hive user-defined functions.
SparkSession.builder.getOrCreate()	Gets an existing **SparkSession** or, if there is no existing one, creates a new one based on the options set in this builder.
SparkSession.builder.master(master)	Sets the Spark master URL to connect to, such as “local” to run locally, “local[4]” to run locally with 4 cores, or “spark://master:7077” to run on a Spark standalone cluster.

SparkSession.catalog	Interface through which the user may create, drop, alter or query underlying databases, tables, functions, etc.
SparkSession.conf	Runtime configuration interface for Spark.
SparkSession.createDataFrame(data[, schema, …])	Creates a DataFrame from an RDD, a list or a pandas.DataFrame.
SparkSession.getActiveSession()	Returns the active SparkSession for the current thread, returned by the builder
SparkSession.newSession()	Returns a new SparkSession as new session, that has separate SQLConf, registered temporary views and UDFs, but shared SparkContext and table cache.
SparkSession.range(start[, end, step, …])	Create a DataFrame with single pyspark.sql.types.LongType column named id, containing elements in a range from start to end (exclusive) with step value step.
SparkSession.read	Returns a DataFrameReader that can be used to read data in as a DataFrame.
SparkSession.readStream	Returns a DataStreamReader that can be used to read data streams as a streaming DataFrame.
SparkSession.sparkContext	Returns the underlying SparkContext.
SparkSession.sql(sqlQuery)	Returns a DataFrame representing the result of the given query.
SparkSession.stop()	Stop the underlying SparkContext.
SparkSession.streams	Returns a StreamingQueryManager that allows managing all the StreamingQuery instances active on *this* context.
SparkSession.table(tableName)	Returns the specified table as a DataFrame.
SparkSession.udf	Returns a UDFRegistration for UDF registration.
SparkSession.version	The version of Spark on which this application is running.

Configuration

RuntimeConfig(jconf)	User-facing configuration API, accessible through *SparkSession.conf*.

Input and Output

DataFrameReader.csv(path[, schema, sep, …])	Loads a CSV file and returns the result as a DataFrame.
DataFrameReader.format(source)	Specifies the input data source format.
DataFrameReader.jdbc(url, table[, column, …])	Construct a DataFrame representing the database table named table accessible via JDBC URL url and connection properties.
DataFrameReader.json(path[, schema, …])	Loads JSON files and returns the results as a DataFrame.
DataFrameReader.load([path, format, schema])	Loads data from a data source and returns it as a DataFrame.
DataFrameReader.option(key, value)	Adds an input option for the underlying data source.
DataFrameReader.options(**options)	Adds input options for the underlying data source.
DataFrameReader.orc(path[, mergeSchema, …])	Loads ORC files, returning the result as a DataFrame.

DataFrameReader.parquet(*paths, **options)	Loads Parquet files, returning the result as a DataFrame.
DataFrameReader.schema(schema)	Specifies the input schema.
DataFrameReader.table(tableName)	Returns the specified table as a DataFrame.
DataFrameWriter.bucketBy(numBuckets, col, *cols)	Buckets the output by the given columns.If specified, the output is laid out on the file system similar to Hive's bucketing scheme.
DataFrameWriter.csv(path[, mode, …])	Saves the content of the DataFrame in CSV format at the specified path.
DataFrameWriter.format(source)	Specifies the underlying output data source.
DataFrameWriter.insertInto(tableName[, …])	Inserts the content of the DataFrame to the specified table.
DataFrameWriter.jdbc(url, table[, mode, …])	Saves the content of the DataFrame to an external database table via JDBC.
DataFrameWriter.json(path[, mode, …])	Saves the content of the DataFrame in JSON format (JSON Lines text format or newline-delimited JSON) at the specified path.
DataFrameWriter.mode(saveMode)	Specifies the behavior when data or table already exists.
DataFrameWriter.option(key, value)	Adds an output option for the underlying data source.
DataFrameWriter.options(**options)	Adds output options for the underlying data source.
DataFrameWriter.orc(path[, mode, …])	Saves the content of the DataFrame in ORC format at the specified path.
DataFrameWriter.parquet(path[, mode, …])	Saves the content of the DataFrame in Parquet format at the specified path.
DataFrameWriter.partitionBy(*cols)	Partitions the output by the given columns on the file system.
DataFrameWriter.save([path, format, mode, …])	Saves the contents of the DataFrame to a data source.
DataFrameWriter.saveAsTable(name[, format, …])	Saves the content of the DataFrame as the specified table.
DataFrameWriter.sortBy(col, *cols)	Sorts the output in each bucket by the given columns on the file system.
DataFrameWriter.text(path[, compression, …])	Saves the content of the DataFrame in a text file at the specified path.

DataFrame APIs

DataFrame.agg(*exprs)	Aggregate on the entire DataFrame without groups (shorthand for `df.groupBy().agg()`).
DataFrame.alias(alias)	Returns a new DataFrame with an alias set.
DataFrame.approxQuantile(col, probabilities, …)	Calculates the approximate quantiles of numerical columns of a DataFrame.
DataFrame.cache()	Persists the DataFrame with the default storage level (*MEMORY_AND_DISK*).
DataFrame.checkpoint([eager])	Returns a checkpointed version of this DataFrame.
DataFrame.coalesce(numPartitions)	Returns a new DataFrame that has exactly *numPartitions* partitions.
DataFrame.colRegex(colName)	Selects column based on the column name specified as a regex and returns it as Column.
DataFrame.collect()	Returns all the records as a list of Row.
DataFrame.columns	Returns all column names as a list.

`DataFrame.corr`(col1, col2[, method])	Calculates the correlation of two columns of a `DataFrame` as a double value.
`DataFrame.count`()	Returns the number of rows in this `DataFrame`.
`DataFrame.cov`(col1, col2)	Calculate the sample covariance for the given columns, specified by their names, as a double value.
`DataFrame.createGlobalTempView`(name)	Creates a global temporary view with this `DataFrame`.
`DataFrame.createOrReplaceGlobalTempView`(name)	Creates or replaces a global temporary view using the given name.
`DataFrame.createOrReplaceTempView`(name)	Creates or replaces a local temporary view with this `DataFrame`.
`DataFrame.createTempView`(name)	Creates a local temporary view with this `DataFrame`.
`DataFrame.crossJoin`(other)	Returns the cartesian product with another `DataFrame`.
`DataFrame.crosstab`(col1, col2)	Computes a pair-wise frequency table of the given columns.
`DataFrame.cube`(*cols)	Create a multi-dimensional cube for the current `DataFrame` using the specified columns, so we can run aggregations on them.
`DataFrame.describe`(*cols)	Computes basic statistics for numeric and string columns.
`DataFrame.distinct`()	Returns a new `DataFrame` containing the distinct rows in this `DataFrame`.
`DataFrame.drop`(*cols)	Returns a new `DataFrame` that drops the specified column.
`DataFrame.dropDuplicates`([subset])	Return a new `DataFrame` with duplicate rows removed, optionally only considering certain columns.
`DataFrame.drop_duplicates`([subset])	`drop_duplicates()` is an alias for `dropDuplicates()`.
`DataFrame.dropna`([how, thresh, subset])	Returns a new `DataFrame` omitting rows with null values.
`DataFrame.dtypes`	Returns all column names and their data types as a list.
`DataFrame.exceptAll`(other)	Return a new `DataFrame` containing rows in this `DataFrame` but not in another `DataFrame` while preserving duplicates.
`DataFrame.explain`([extended, mode])	Prints the (logical and physical) plans to the console for debugging purpose.
`DataFrame.fillna`(value[, subset])	Replace null values, alias for `na.fill()`.
`DataFrame.filter`(condition)	Filters rows using the given condition.
`DataFrame.first`()	Returns the first row as a `Row`.
`DataFrame.foreach`(f)	Applies the `f` function to all `Row` of this `DataFrame`.
`DataFrame.foreachPartition`(f)	Applies the `f` function to each partition of this `DataFrame`.
`DataFrame.freqItems`(cols[, support])	Finding frequent items for columns, possibly with false positives.
`DataFrame.groupBy`(*cols)	Groups the `DataFrame` using the specified columns, so we can run aggregation on them.
`DataFrame.head`([n])	Returns the first `n` rows.
`DataFrame.hint`(name, *parameters)	Specifies some hint on the current `DataFrame`.
`DataFrame.inputFiles`()	Returns a best-effort snapshot of the files that compose this `DataFrame`.

DataFrame.intersect(other)	Return a new DataFrame containing rows only in both this DataFrame and another DataFrame.
DataFrame.intersectAll(other)	Return a new DataFrame containing rows in both this DataFrame and another DataFrame while preserving duplicates.
DataFrame.isLocal()	Returns True if the collect() and take() methods can be run locally (without any Spark executors).
DataFrame.isStreaming	Returns True if this DataFrame contains one or more sources that continuously return data as it arrives.
DataFrame.join(other[, on, how])	Joins with another DataFrame, using the given join expression.
DataFrame.limit(num)	Limits the result count to the number specified.
DataFrame.localCheckpoint([eager])	Returns a locally checkpointed version of this DataFrame.
DataFrame.mapInPandas(func, schema)	Maps an iterator of batches in the current DataFrame using a Python native function that takes and outputs a pandas DataFrame, and returns the result as a DataFrame.
DataFrame.na	Returns a DataFrameNaFunctions for handling missing values.
DataFrame.orderBy(*cols, **kwargs)	Returns a new DataFrame sorted by the specified column(s).
DataFrame.persist([storageLevel])	Sets the storage level to persist the contents of the DataFrame across operations after the first time it is computed.
DataFrame.printSchema()	Prints out the schema in the tree format.
DataFrame.randomSplit(weights[, seed])	Randomly splits this DataFrame with the provided weights.
DataFrame.rdd	Returns the content as an pyspark.RDD of Row.
DataFrame.registerTempTable(name)	Registers this DataFrame as a temporary table using the given name.
DataFrame.repartition(numPartitions, *cols)	Returns a new DataFrame partitioned by the given partitioning expressions.
DataFrame.repartitionByRange(numPartitions, …)	Returns a new DataFrame partitioned by the given partitioning expressions.
DataFrame.replace(to_replace[, value, subset])	Returns a new DataFrame replacing a value with another value.
DataFrame.rollup(*cols)	Create a multi-dimensional rollup for the current DataFrame using the specified columns, so we can run aggregation on them.
DataFrame.sameSemantics(other)	Returns *True* when the logical query plans inside both DataFrames are equal and therefore return same results.
DataFrame.sample([withReplacement, …])	Returns a sampled subset of this DataFrame.
DataFrame.sampleBy(col, fractions[, seed])	Returns a stratified sample without replacement based on the fraction given on each stratum.
DataFrame.schema	Returns the schema of this DataFrame as a pyspark.sql.types.StructType.
DataFrame.select(*cols)	Projects a set of expressions and returns a new DataFrame.
DataFrame.selectExpr(*expr)	Projects a set of SQL expressions and returns a new DataFrame.
DataFrame.semanticHash()	Returns a hash code of the logical query plan against this DataFrame.

DataFrame.show([n, truncate, vertical])	Prints the first n rows to the console.
DataFrame.sort(*cols, **kwargs)	Returns a new DataFrame sorted by the specified column(s).
DataFrame.sortWithinPartitions(*cols, **kwargs)	Returns a new DataFrame with each partition sorted by the specified column(s).
DataFrame.stat	Returns a DataFrameStatFunctions for statistic functions.
DataFrame.storageLevel	Get the DataFrame's current storage level.
DataFrame.subtract(other)	Return a new DataFrame containing rows in this DataFrame but not in another DataFrame.
DataFrame.summary(*statistics)	Computes specified statistics for numeric and string columns.
DataFrame.tail(num)	Returns the last num rows as a list of Row.
DataFrame.take(num)	Returns the first num rows as a list of Row.
DataFrame.toDF(*cols)	Returns a new DataFrame that with new specified column names
DataFrame.toJSON([use_unicode])	Converts a DataFrame into a RDD of string.
DataFrame.toLocalIterator([prefetchPartitions])	Returns an iterator that contains all of the rows in this DataFrame.
DataFrame.toPandas()	Returns the contents of this DataFrame as Pandas pandas.DataFrame.
DataFrame.transform(func)	Returns a new DataFrame.
DataFrame.union(other)	Return a new DataFrame containing union of rows in this and another DataFrame.
DataFrame.unionAll(other)	Return a new DataFrame containing union of rows in this and another DataFrame.
DataFrame.unionByName(other[, …])	Returns a new DataFrame containing union of rows in this and another DataFrame.
DataFrame.unpersist([blocking])	Marks the DataFrame as non-persistent, and remove all blocks for it from memory and disk.
DataFrame.where(condition)	where() is an alias for filter().
DataFrame.withColumn(colName, col)	Returns a new DataFrame by adding a column or replacing the existing column that has the same name.
DataFrame.withColumnRenamed(existing, new)	Returns a new DataFrame by renaming an existing column.
DataFrame.withWatermark(eventTime, …)	Defines an event time watermark for this DataFrame.
DataFrame.write	Interface for saving the content of the non-streaming DataFrame out into external storage.
DataFrame.writeStream	Interface for saving the content of the streaming DataFrame out into external storage.
DataFrame.writeTo(table)	Create a write configuration builder for v2 sources.
DataFrameNaFunctions.drop([how, thresh, subset])	Returns a new DataFrame omitting rows with null values.
DataFrameNaFunctions.fill(value[, subset])	Replace null values, alias for na.fill().
DataFrameNaFunctions.replace(to_replace[, …])	Returns a new DataFrame replacing a value with another value.
DataFrameStatFunctions.approxQuantile(col, …)	Calculates the approximate quantiles of numerical columns of a DataFrame.
DataFrameStatFunctions.corr(col1, col2[, method])	Calculates the correlation of two columns of a DataFrame as a double value.

`DataFrameStatFunctions.cov`(col1, col2)	Calculate the sample covariance for the given columns, specified by their names, as a double value.
`DataFrameStatFunctions.crosstab`(col1, col2)	Computes a pair-wise frequency table of the given columns.
`DataFrameStatFunctions.freqItems`(cols[, support])	Finding frequent items for columns, possibly with false positives.
`DataFrameStatFunctions.sampleBy`(col, fractions)	Returns a stratified sample without replacement based on the fraction given on each stratum.

Column APIs

`Column.alias`(*alias, **kwargs)	Returns this column aliased with a new name or names (in the case of expressions that return more than one column, such as explode).
`Column.asc`()	Returns a sort expression based on ascending order of the column.
`Column.asc_nulls_first`()	Returns a sort expression based on ascending order of the column, and null values return before non-null values.
`Column.asc_nulls_last`()	Returns a sort expression based on ascending order of the column, and null values appear after non-null values.
`Column.astype`(dataType)	`astype()` is an alias for `cast()`.
`Column.between`(lowerBound, upperBound)	True if the current column is between the lower bound and upper bound, inclusive.
`Column.bitwiseAND`(other)	Compute bitwise AND of this expression with another expression.
`Column.bitwiseOR`(other)	Compute bitwise OR of this expression with another expression.
`Column.bitwiseXOR`(other)	Compute bitwise XOR of this expression with another expression.
`Column.cast`(dataType)	Casts the column into type `dataType`.
`Column.contains`(other)	Contains the other element.
`Column.desc`()	Returns a sort expression based on the descending order of the column.
`Column.desc_nulls_first`()	Returns a sort expression based on the descending order of the column, and null values appear before non-null values.
`Column.desc_nulls_last`()	Returns a sort expression based on the descending order of the column, and null values appear after non-null values.
`Column.dropFields`(*fieldNames)	An expression that drops fields in `StructType` by name.
`Column.endswith`(other)	String ends with.
`Column.eqNullSafe`(other)	Equality test that is safe for null values.
`Column.getField`(name)	An expression that gets a field by name in a `StructType`.
`Column.getItem`(key)	An expression that gets an item at position `ordinal` out of a list, or gets an item by key out of a dict.
`Column.isNotNull`()	True if the current expression is NOT null.
`Column.isNull`()	True if the current expression is null.

`Column.isin`(*cols)	A boolean expression that is evaluated to true if the value of this expression is contained by the evaluated values of the arguments.
`Column.like`(other)	SQL like expression.
`Column.name`(*alias, **kwargs)	`name()` is an alias for `alias()`.
`Column.otherwise`(value)	Evaluates a list of conditions and returns one of multiple possible result expressions.
`Column.over`(window)	Define a windowing column.
`Column.rlike`(other)	SQL RLIKE expression (LIKE with Regex).
`Column.startswith`(other)	String starts with.
`Column.substr`(startPos, length)	Return a `Column` which is a substring of the column.
`Column.when`(condition, value)	Evaluates a list of conditions and returns one of multiple possible result expressions.
`Column.withField`(fieldName, col)	An expression that adds/replaces a field in `StructType` by name.

Data Types

`ArrayType`(elementType[, containsNull])	Array data type.
`BinaryType`	Binary (byte array) data type.
`BooleanType`	Boolean data type.
`ByteType`	Byte data type, i.e.
`DataType`	Base class for data types.
`DateType`	Date (datetime.date) data type.
`DecimalType`([precision, scale])	Decimal (decimal.Decimal) data type.
`DoubleType`	Double data type, representing double precision floats.
`FloatType`	Float data type, representing single precision floats.
`IntegerType`	Int data type, i.e.
`LongType`	Long data type, i.e.
`MapType`(keyType, valueType[, valueContainsNull])	Map data type.
`NullType`	Null type.
`ShortType`	Short data type, i.e.
`StringType`	String data type.
`StructField`(name, dataType[, nullable, metadata])	A field in `StructType`.
`StructType`([fields])	Struct type, consisting of a list of `StructField`.
`TimestampType`	Timestamp (datetime.datetime) data type.

Row

`Row.asDict`([recursive])	Return as a dict

Functions

`abs`(col)	Computes the absolute value.
`acos`(col)	*New in version 1.4.0.*
`acosh`(col)	Computes inverse hyperbolic cosine of the input column.
`add_months`(start, months)	Returns the date that is *months* months after *start*
`aggregate`(col, initialValue, merge[, finish])	Applies a binary operator to an initial state and all elements in the array, and reduces this to a single state.
`approxCountDistinct`(col[, rsd])	*Deprecated since version 2.1.0.*
`approx_count_distinct`(col[, rsd])	Aggregate function: returns a new `Column` for approximate distinct count of column *col*.
`array`(*cols)	Creates a new array column.
`array_contains`(col, value)	Collection function: returns null if the array is null, true if the array contains the given value, and false otherwise.
`array_distinct`(col)	Collection function: removes duplicate values from the array.
`array_except`(col1, col2)	Collection function: returns an array of the elements in col1 but not in col2, without duplicates.
`array_intersect`(col1, col2)	Collection function: returns an array of the elements in the intersection of col1 and col2, without duplicates.
`array_join`(col, delimiter[, null_replacement])	Concatenates the elements of *column* using the *delimiter*.
`array_max`(col)	Collection function: returns the maximum value of the array.
`array_min`(col)	Collection function: returns the minimum value of the array.
`array_position`(col, value)	Collection function: Locates the position of the first occurrence of the given value in the given array.
`array_remove`(col, element)	Collection function: Remove all elements that equal to element from the given array.
`array_repeat`(col, count)	Collection function: creates an array containing a column repeated count times.
`array_sort`(col)	Collection function: sorts the input array in ascending order.
`array_union`(col1, col2)	Collection function: returns an array of the elements in the union of col1 and col2, without duplicates.
`arrays_overlap`(a1, a2)	Collection function: returns true if the arrays contain any common non-null element; if not, returns null if both the arrays are non-empty and any of them contains a null element; returns false otherwise.

arrays_zip(*cols)	Collection function: Returns a merged array of structs in which the N-th struct contains all N-th values of input arrays.
asc(col)	Returns a sort expression based on the ascending order of the given column name.
asc_nulls_first(col)	Returns a sort expression based on the ascending order of the given column name, and null values return before non-null values.
asc_nulls_last(col)	Returns a sort expression based on the ascending order of the given column name, and null values appear after non-null values.
ascii(col)	Computes the numeric value of the first character of the string column.
asin(col)	*New in version 1.3.0.*
asinh(col)	Computes inverse hyperbolic sine of the input column.
assert_true(col[, errMsg])	Returns null if the input column is true; throws an exception with the provided error message otherwise.
atan(col)	*New in version 1.4.0.*
atanh(col)	Computes inverse hyperbolic tangent of the input column.
atan2(col1, col2)	*New in version 1.4.0.*
avg(col)	Aggregate function: returns the average of the values in a group.
base64(col)	Computes the BASE64 encoding of a binary column and returns it as a string column.
bin(col)	Returns the string representation of the binary value of the given column.
bitwiseNOT(col)	Computes bitwise not.
broadcast(df)	Marks a DataFrame as small enough for use in broadcast joins.
bround(col[, scale])	Round the given value to *scale* decimal places using HALF_EVEN rounding mode if *scale* >= 0 or at integral part when *scale* < 0.
bucket(numBuckets, col)	Partition transform function: A transform for any type that partitions by a hash of the input column.
cbrt(col)	Computes the cube-root of the given value.
ceil(col)	Computes the ceiling of the given value.
coalesce(*cols)	Returns the first column that is not null.
col(col)	Returns a **Column** based on the given column name.' Examples — —– >>> col('x') Column<'x'> >>> column('x') Column<'x'>
collect_list(col)	Aggregate function: returns a list of objects with duplicates.
collect_set(col)	Aggregate function: returns a set of objects with duplicate elements eliminated.
column(col)	Returns a **Column** based on the given column name.' Examples — —– >>> col('x') Column<'x'> >>> column('x') Column<'x'>

concat(*cols)	Concatenates multiple input columns together into a single column.
concat_ws(sep, *cols)	Concatenates multiple input string columns together into a single string column, using the given separator.
conv(col, fromBase, toBase)	Convert a number in a string column from one base to another.
corr(col1, col2)	Returns a new Column for the Pearson Correlation Coefficient for col1 and col2.
cos(col)	*New in version 1.4.0.*
cosh(col)	*New in version 1.4.0.*
count(col)	Aggregate function: returns the number of items in a group.
countDistinct(col, *cols)	Returns a new Column for distinct count of col or cols.
covar_pop(col1, col2)	Returns a new Column for the population covariance of col1 and col2.
covar_samp(col1, col2)	Returns a new Column for the sample covariance of col1 and col2.
crc32(col)	Calculates the cyclic redundancy check value (CRC32) of a binary column and returns the value as a bigint.
create_map(*cols)	Creates a new map column.
cume_dist()	Window function: returns the cumulative distribution of values within a window partition, i.e.
current_date()	Returns the current date at the start of query evaluation as a DateType column.
current_timestamp()	Returns the current timestamp at the start of query evaluation as a TimestampType column.
date_add(start, days)	Returns the date that is *days* days after *start*
date_format(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.
date_sub(start, days)	Returns the date that is *days* days before *start*
date_trunc(format, timestamp)	Returns timestamp truncated to the unit specified by the format.
datediff(end, start)	Returns the number of days from *start* to *end*.
dayofmonth(col)	Extract the day of the month of a given date as integer.
dayofweek(col)	Extract the day of the week of a given date as integer.
dayofyear(col)	Extract the day of the year of a given date as integer.
days(col)	Partition transform function: A transform for timestamps and dates to partition data into days.
decode(col, charset)	Computes the first argument into a string from a binary using the provided character set (one of ‘US-ASCII’, ‘ISO-8859-1’, ‘UTF-8’, ‘UTF-16BE’, ‘UTF-16LE’, ‘UTF-16’).
degrees(col)	Converts an angle measured in radians to an approximately equivalent angle measured in degrees.
dense_rank()	Window function: returns the rank of rows within a window partition, without any gaps.

`desc`(col)	Returns a sort expression based on the descending order of the given column name.
`desc_nulls_first`(col)	Returns a sort expression based on the descending order of the given column name, and null values appear before non-null values.
`desc_nulls_last`(col)	Returns a sort expression based on the descending order of the given column name, and null values appear after non-null values.
`element_at`(col, extraction)	Collection function: Returns element of array at given index in extraction if col is array.
`encode`(col, charset)	Computes the first argument into a binary from a string using the provided character set (one of ‘US-ASCII’, ‘ISO-8859-1’, ‘UTF-8’, ‘UTF-16BE’, ‘UTF-16LE’, ‘UTF-16’).
`exists`(col, f)	Returns whether a predicate holds for one or more elements in the array.
`exp`(col)	Computes the exponential of the given value.
`explode`(col)	Returns a new row for each element in the given array or map.
`explode_outer`(col)	Returns a new row for each element in the given array or map.
`expm1`(col)	Computes the exponential of the given value minus one.
`expr`(str)	Parses the expression string into the column that it represents
`factorial`(col)	Computes the factorial of the given value.
`filter`(col, f)	Returns an array of elements for which a predicate holds in a given array.
`first`(col[, ignorenulls])	Aggregate function: returns the first value in a group.
`flatten`(col)	Collection function: creates a single array from an array of arrays.
`floor`(col)	Computes the floor of the given value.
`forall`(col, f)	Returns whether a predicate holds for every element in the array.
`format_number`(col, d)	Formats the number X to a format like ‘#,–#,–#.–’, rounded to d decimal places with HALF_EVEN round mode, and returns the result as a string.
`format_string`(format, *cols)	Formats the arguments in printf-style and returns the result as a string column.
`from_csv`(col, schema[, options])	Parses a column containing a CSV string to a row with the specified schema.
`from_json`(col, schema[, options])	Parses a column containing a JSON string into a `MapType` with `StringType` as keys type, `StructType` or `ArrayType` with the specified schema.
`from_unixtime`(timestamp[, format])	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 given format.
`from_utc_timestamp`(timestamp, tz)	This is a common function for databases supporting TIMESTAMP WITHOUT TIMEZONE.
`get_json_object`(col, path)	Extracts json object from a json string based on json path specified, and returns json string of the extracted json object.
`greatest`(*cols)	Returns the greatest value of the list of column names, skipping null values.
`grouping`(col)	Aggregate function: 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.
`grouping_id`(*cols)	Aggregate function: returns the level of grouping, equals to

hash(*cols)	Calculates the hash code of given columns, and returns the result as an int column.
hex(col)	Computes hex value of the given column, which could be **pyspark.sql.types.StringType**, **pyspark.sql.types.BinaryType**, **pyspark.sql.types.IntegerType** or **pyspark.sql.types.LongType**.
hour(col)	Extract the hours of a given date as integer.
hours(col)	Partition transform function: A transform for timestamps to partition data into hours.
hypot(col1, col2)	Computes `sqrt(a^2 + b^2)` without intermediate overflow or underflow.
initcap(col)	Translate the first letter of each word to upper case in the sentence.
input_file_name()	Creates a string column for the file name of the current Spark task.
instr(str, substr)	Locate the position of the first occurrence of substr column in the given string.
isnan(col)	An expression that returns true iff the column is NaN.
isnull(col)	An expression that returns true iff the column is null.
json_tuple(col, *fields)	Creates a new row for a json column according to the given field names.
kurtosis(col)	Aggregate function: returns the kurtosis of the values in a group.
lag(col[, offset, default])	Window function: returns the value that is *offset* rows before the current row, and *default* if there is less than *offset* rows before the current row.
last(col[, ignorenulls])	Aggregate function: returns the last value in a group.
last_day(date)	Returns the last day of the month which the given date belongs to.
lead(col[, offset, default])	Window function: returns the value that is *offset* rows after the current row, and *default* if there is less than *offset* rows after the current row.
least(*cols)	Returns the least value of the list of column names, skipping null values.
length(col)	Computes the character length of string data or number of bytes of binary data.
levenshtein(left, right)	Computes the Levenshtein distance of the two given strings.
lit(col)	Creates a **Column** of literal value.
locate(substr, str[, pos])	Locate the position of the first occurrence of substr in a string column, after position pos.
log(arg1[, arg2])	Returns the first argument-based logarithm of the second argument.
log10(col)	Computes the logarithm of the given value in Base 10.
log1p(col)	Computes the natural logarithm of the given value plus one.
log2(col)	Returns the base-2 logarithm of the argument.
lower(col)	Converts a string expression to lower case.
lpad(col, len, pad)	Left-pad the string column to width *len* with *pad*.
ltrim(col)	Trim the spaces from left end for the specified string value.
map_concat(*cols)	Returns the union of all the given maps.

map_entries(col)	Collection function: Returns an unordered array of all entries in the given map.
map_filter(col, f)	Returns a map whose key-value pairs satisfy a predicate.
map_from_arrays(col1, col2)	Creates a new map from two arrays.
map_from_entries(col)	Collection function: Returns a map created from the given array of entries.
map_keys(col)	Collection function: Returns an unordered array containing the keys of the map.
map_values(col)	Collection function: Returns an unordered array containing the values of the map.
map_zip_with(col1, col2, f)	Merge two given maps, key-wise into a single map using a function.
max(col)	Aggregate function: returns the maximum value of the expression in a group.
md5(col)	Calculates the MD5 digest and returns the value as a 32 character hex string.
mean(col)	Aggregate function: returns the average of the values in a group.
min(col)	Aggregate function: returns the minimum value of the expression in a group.
minute(col)	Extract the minutes of a given date as integer.
monotonically_increasing_id()	A column that generates monotonically increasing 64-bit integers.
month(col)	Extract the month of a given date as integer.
months(col)	Partition transform function: A transform for timestamps and dates to partition data into months.
months_between(date1, date2[, roundOff])	Returns number of months between dates date1 and date2.
nanvl(col1, col2)	Returns col1 if it is not NaN, or col2 if col1 is NaN.
next_day(date, dayOfWeek)	Returns the first date which is later than the value of the date column.
nth_value(col, offset[, ignoreNulls])	Window function: returns the value that is the *offset*th row of the window frame (counting from 1), and *null* if the size of window frame is less than *offset* rows.
ntile(n)	Window function: returns the ntile group id (from 1 to *n* inclusive) in an ordered window partition.
overlay(src, replace, pos[, len])	Overlay the specified portion of *src* with *replace*, starting from byte position *pos* of *src* and proceeding for *len* bytes.
pandas_udf([f, returnType, functionType])	Creates a pandas user defined function (a.k.a.
percent_rank()	Window function: returns the relative rank (i.e.
percentile_approx(col, percentage[, accuracy])	Returns the approximate *percentile* of the numeric column *col* which is the smallest value in the ordered *col* values (sorted from least to greatest) such that no more than *percentage* of *col* values is less than the value or equal to that value.
posexplode(col)	Returns a new row for each element with position in the given array or map.
posexplode_outer(col)	Returns a new row for each element with position in the given array or map.
pow(col1, col2)	Returns the value of the first argument raised to the power of the second argument.
quarter(col)	Extract the quarter of a given date as integer.

radians(col)	Converts an angle measured in degrees to an approximately equivalent angle measured in radians.
raise_error(errMsg)	Throws an exception with the provided error message.
rand([seed])	Generates a random column with independent and identically distributed (i.i.d.) samples uniformly distributed in [0.0, 1.0).
randn([seed])	Generates a column with independent and identically distributed (i.i.d.) samples from the standard normal distribution.
rank()	Window function: returns the rank of rows within a window partition.
regexp_extract(str, pattern, idx)	Extract a specific group matched by a Java regex, from the specified string column.
regexp_replace(str, pattern, replacement)	Replace all substrings of the specified string value that match regexp with rep.
repeat(col, n)	Repeats a string column n times, and returns it as a new string column.
reverse(col)	Collection function: returns a reversed string or an array with reverse order of elements.
rint(col)	Returns the double value that is closest in value to the argument and is equal to a mathematical integer.
round(col[, scale])	Round the given value to *scale* decimal places using HALF_UP rounding mode if *scale* >= 0 or at integral part when *scale* < 0.
row_number()	Window function: returns a sequential number starting at 1 within a window partition.
rpad(col, len, pad)	Right-pad the string column to width *len* with *pad*.
rtrim(col)	Trim the spaces from right end for the specified string value.
schema_of_csv(csv[, options])	Parses a CSV string and infers its schema in DDL format.
schema_of_json(json[, options])	Parses a JSON string and infers its schema in DDL format.
second(col)	Extract the seconds of a given date as integer.
sequence(start, stop[, step])	Generate a sequence of integers from *start* to *stop*, incrementing by *step*.
sha1(col)	Returns the hex string result of SHA-1.
sha2(col, numBits)	Returns the hex string result of SHA-2 family of hash functions (SHA-224, SHA-256, SHA-384, and SHA-512).
shiftLeft(col, numBits)	Shift the given value numBits left.
shiftRight(col, numBits)	(Signed) shift the given value numBits right.
shiftRightUnsigned(col, numBits)	Unsigned shift the given value numBits right.
shuffle(col)	Collection function: Generates a random permutation of the given array.
signum(col)	Computes the signum of the given value.
sin(col)	*New in version 1.4.0.*

`sinh`(col)	*New in version 1.4.0.*
`size`(col)	Collection function: returns the length of the array or map stored in the column.
`skewness`(col)	Aggregate function: returns the skewness of the values in a group.
`slice`(x, start, length)	Collection function: returns an array containing all the elements in *x* from index *start* (array indices start at 1, or from the end if *start* is negative) with the specified *length*.
`sort_array`(col[, asc])	Collection function: sorts the input array in ascending or descending order according to the natural ordering of the array elements.
`soundex`(col)	Returns the SoundEx encoding for a string
`spark_partition_id`()	A column for partition ID.
`split`(str, pattern[, limit])	Splits str around matches of the given pattern.
`sqrt`(col)	Computes the square root of the specified float value.
`stddev`(col)	Aggregate function: alias for stddev_samp.
`stddev_pop`(col)	Aggregate function: returns population standard deviation of the expression in a group.
`stddev_samp`(col)	Aggregate function: returns the unbiased sample standard deviation of the expression in a group.
`struct`(*cols)	Creates a new struct column.
`substring`(str, pos, len)	Substring starts at *pos* and is of length *len* when str is String type or returns the slice of byte array that starts at *pos* in byte and is of length *len* when str is Binary type.
`substring_index`(str, delim, count)	Returns the substring from string str before count occurrences of the delimiter delim.
`sum`(col)	Aggregate function: returns the sum of all values in the expression.
`sumDistinct`(col)	Aggregate function: returns the sum of distinct values in the expression.
`tan`(col)	*New in version 1.4.0.*
`tanh`(col)	*New in version 1.4.0.*
`timestamp_seconds`(col)	*New in version 3.1.0.*
`toDegrees`(col)	*Deprecated since version 2.1.0.*
`toRadians`(col)	*Deprecated since version 2.1.0.*
`to_csv`(col[, options])	Converts a column containing a `StructType` into a CSV string.
`to_date`(col[, format])	Converts a `Column` into `pyspark.sql.types.DateType` using the optionally specified format.
`to_json`(col[, options])	Converts a column containing a `StructType`, `ArrayType` or a `MapType` into a JSON string.

`to_timestamp`(col[, format])	Converts a `Column` into `pyspark.sql.types.TimestampType` using the optionally specified format.
`to_utc_timestamp`(timestamp, tz)	This is a common function for databases supporting TIMESTAMP WITHOUT TIMEZONE.
`transform`(col, f)	Returns an array of elements after applying a transformation to each element in the input array.
`transform_keys`(col, f)	Applies a function to every key-value pair in a map and returns a map with the results of those applications as the new keys for the pairs.
`transform_values`(col, f)	Applies a function to every key-value pair in a map and returns a map with the results of those applications as the new values for the pairs.
`translate`(srcCol, matching, replace)	A function translate any character in the *srcCol* by a character in *matching*.
`trim`(col)	Trim the spaces from both ends for the specified string column.
`trunc`(date, format)	Returns date truncated to the unit specified by the format.
`udf`([f, returnType])	Creates a user defined function (UDF).
`unbase64`(col)	Decodes a BASE64 encoded string column and returns it as a binary column.
`unhex`(col)	Inverse of hex.
`unix_timestamp`([timestamp, format])	Convert time string with given pattern (‘yyyy-MM-dd HH:mm:ss’, by default) to Unix time stamp (in seconds), using the default timezone and the default locale, return null if fail.
`upper`(col)	Converts a string expression to upper case.
`var_pop`(col)	Aggregate function: returns the population variance of the values in a group.
`var_samp`(col)	Aggregate function: returns the unbiased sample variance of the values in a group.
`variance`(col)	Aggregate function: alias for var_samp
`weekofyear`(col)	Extract the week number of a given date as integer.
`when`(condition, value)	Evaluates a list of conditions and returns one of multiple possible result expressions.
`window`(timeColumn, windowDuration[, …])	Bucketize rows into one or more time windows given a timestamp specifying column.
`xxhash64`(*cols)	Calculates the hash code of given columns using the 64-bit variant of the xxHash algorithm, and returns the result as a long column.
`year`(col)	Extract the year of a given date as integer.
`years`(col)	Partition transform function: A transform for timestamps and dates to partition data into years.
`zip_with`(left, right, f)	Merge two given arrays, element-wise, into a single array using a function.

`from_avro`(data, jsonFormatSchema[, options])	Converts a binary column of Avro format into its corresponding catalyst value.
`to_avro`(data[, jsonFormatSchema])	Converts a column into binary of avro format.

Window

`Window.currentRow`	
`Window.orderBy`(*cols)	Creates a `WindowSpec` with the ordering defined.

`Window.partitionBy`(*cols)	Creates a `WindowSpec` with the partitioning defined.
`Window.rangeBetween`(start, end)	Creates a `WindowSpec` with the frame boundaries defined, from *start* (inclusive) to *end* (inclusive).
`Window.rowsBetween`(start, end)	Creates a `WindowSpec` with the frame boundaries defined, from *start* (inclusive) to *end* (inclusive).
`Window.unboundedFollowing`	
`Window.unboundedPreceding`	
`WindowSpec.orderBy`(*cols)	Defines the ordering columns in a `WindowSpec`.
`WindowSpec.partitionBy`(*cols)	Defines the partitioning columns in a `WindowSpec`.
`WindowSpec.rangeBetween`(start, end)	Defines the frame boundaries, from *start* (inclusive) to *end* (inclusive).
`WindowSpec.rowsBetween`(start, end)	Defines the frame boundaries, from *start* (inclusive) to *end* (inclusive).

Grouping

`GroupedData.agg`(*exprs)	Compute aggregates and returns the result as a `DataFrame`.
`GroupedData.apply`(udf)	It is an alias of `pyspark.sql.GroupedData.applyInPandas()`; however, it takes a `pyspark.sql.functions.pandas_udf()` whereas `pyspark.sql.GroupedData.applyInPandas()` takes a Python native function.
`GroupedData.applyInPandas`(func, schema)	Maps each group of the current `DataFrame` using a pandas udf and returns the result as a *DataFrame*.
`GroupedData.avg`(*cols)	Computes average values for each numeric columns for each group.
`GroupedData.cogroup`(other)	Cogroups this group with another group so that we can run cogrouped operations.
`GroupedData.count`()	Counts the number of records for each group.
`GroupedData.max`(*cols)	Computes the max value for each numeric columns for each group.
`GroupedData.mean`(*cols)	Computes average values for each numeric columns for each group.
`GroupedData.min`(*cols)	Computes the min value for each numeric column for each group.
`GroupedData.pivot`(pivot_col[, values])	Pivots a column of the current `DataFrame` and perform the specified aggregation.
`GroupedData.sum`(*cols)	Computes the sum for each numeric columns for each group.
`PandasCogroupedOps.applyInPandas`(func, schema)	Applies a function to each cogroup using pandas and returns the result as a *DataFrame*.

<< API Reference

pyspark.sql.SparkSession >>