**pyspark.sql.DataFrame**

A distributed collection of data grouped into named columns.

A DataFrame is equivalent to a relational table in Spark SQL, and can be created using various functions in SparkSession:

Once created, it can be manipulated using the various domain-specific-language (DSL) functions defined in: DataFrame, Column.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Methods**   |  |  | | --- | --- | | [**agg**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.agg.html#pyspark.sql.DataFrame.agg)(\*exprs) | Aggregate on the entire [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) without groups (shorthand for df.groupBy().agg()). | | [**alias**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.alias.html#pyspark.sql.DataFrame.alias)(alias) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) with an alias set. | | [**approxQuantile**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.approxQuantile.html#pyspark.sql.DataFrame.approxQuantile)(col, probabilities, relativeError) | Calculates the approximate quantiles of numerical columns of a [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**cache**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.cache.html#pyspark.sql.DataFrame.cache)() | Persists the [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) with the default storage level (*MEMORY\_AND\_DISK*). | | [**checkpoint**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.checkpoint.html#pyspark.sql.DataFrame.checkpoint)([eager]) | Returns a checkpointed version of this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**coalesce**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.coalesce.html#pyspark.sql.DataFrame.coalesce)(numPartitions) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) that has exactly *numPartitions* partitions. | | [**colRegex**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.colRegex.html#pyspark.sql.DataFrame.colRegex)(colName) | Selects column based on the column name specified as a regex and returns it as [**Column**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.html#pyspark.sql.Column). | | [**collect**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.collect.html#pyspark.sql.DataFrame.collect)() | Returns all the records as a list of [**Row**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Row.html#pyspark.sql.Row). | | [**corr**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.corr.html#pyspark.sql.DataFrame.corr)(col1, col2[, method]) | Calculates the correlation of two columns of a [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) as a double value. | | [**count**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.count.html#pyspark.sql.DataFrame.count)() | Returns the number of rows in this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**cov**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.cov.html#pyspark.sql.DataFrame.cov)(col1, col2) | Calculate the sample covariance for the given columns, specified by their names, as a double value. | | [**createGlobalTempView**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.createGlobalTempView.html#pyspark.sql.DataFrame.createGlobalTempView)(name) | Creates a global temporary view with this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**createOrReplaceGlobalTempView**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.createOrReplaceGlobalTempView.html#pyspark.sql.DataFrame.createOrReplaceGlobalTempView)(name) | Creates or replaces a global temporary view using the given name. | | [**createOrReplaceTempView**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.createOrReplaceTempView.html#pyspark.sql.DataFrame.createOrReplaceTempView)(name) | Creates or replaces a local temporary view with this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**createTempView**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.createTempView.html#pyspark.sql.DataFrame.createTempView)(name) | Creates a local temporary view with this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**crossJoin**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.crossJoin.html#pyspark.sql.DataFrame.crossJoin)(other) | Returns the cartesian product with another [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**crosstab**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.crosstab.html#pyspark.sql.DataFrame.crosstab)(col1, col2) | Computes a pair-wise frequency table of the given columns. | | [**cube**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.cube.html#pyspark.sql.DataFrame.cube)(\*cols) | Create a multi-dimensional cube for the current [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) using the specified columns, so we can run aggregations on them. | | [**describe**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.describe.html#pyspark.sql.DataFrame.describe)(\*cols) | Computes basic statistics for numeric and string columns. | | [**distinct**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.distinct.html#pyspark.sql.DataFrame.distinct)() | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) containing the distinct rows in this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**drop**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.drop.html#pyspark.sql.DataFrame.drop)(\*cols) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) that drops the specified column. | | [**dropDuplicates**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.dropDuplicates.html#pyspark.sql.DataFrame.dropDuplicates)([subset]) | Return a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) with duplicate rows removed, optionally only considering certain columns. | | [**drop\_duplicates**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.drop_duplicates.html#pyspark.sql.DataFrame.drop_duplicates)([subset]) | [**drop\_duplicates()**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.drop_duplicates.html#pyspark.sql.DataFrame.drop_duplicates) is an alias for [**dropDuplicates()**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.dropDuplicates.html#pyspark.sql.DataFrame.dropDuplicates). | | [**dropna**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.dropna.html#pyspark.sql.DataFrame.dropna)([how, thresh, subset]) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) omitting rows with null values. | | [**exceptAll**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.exceptAll.html#pyspark.sql.DataFrame.exceptAll)(other) | Return a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) containing rows in this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) but not in another [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) while preserving duplicates. | | [**explain**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.explain.html#pyspark.sql.DataFrame.explain)([extended, mode]) | Prints the (logical and physical) plans to the console for debugging purpose. | | [**fillna**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.fillna.html#pyspark.sql.DataFrame.fillna)(value[, subset]) | Replace null values, alias for na.fill(). | | [**filter**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.filter.html#pyspark.sql.DataFrame.filter)(condition) | Filters rows using the given condition. | | [**first**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.first.html#pyspark.sql.DataFrame.first)() | Returns the first row as a [**Row**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Row.html#pyspark.sql.Row). | | [**foreach**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.foreach.html#pyspark.sql.DataFrame.foreach)(f) | Applies the f function to all [**Row**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Row.html#pyspark.sql.Row) of this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**foreachPartition**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.foreachPartition.html#pyspark.sql.DataFrame.foreachPartition)(f) | Applies the f function to each partition of this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**freqItems**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.freqItems.html#pyspark.sql.DataFrame.freqItems)(cols[, support]) | Finding frequent items for columns, possibly with false positives. | | [**groupBy**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.groupBy.html#pyspark.sql.DataFrame.groupBy)(\*cols) | Groups the [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) using the specified columns, so we can run aggregation on them. | | **groupby**(\*cols) | **groupby()** is an alias for [**groupBy()**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.groupBy.html#pyspark.sql.DataFrame.groupBy). | | [**head**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.head.html#pyspark.sql.DataFrame.head)([n]) | Returns the first n rows. | | [**hint**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.hint.html#pyspark.sql.DataFrame.hint)(name, \*parameters) | Specifies some hint on the current [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**inputFiles**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.inputFiles.html#pyspark.sql.DataFrame.inputFiles)() | Returns a best-effort snapshot of the files that compose this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**intersect**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.intersect.html#pyspark.sql.DataFrame.intersect)(other) | Return a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) containing rows only in both this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) and another [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**intersectAll**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.intersectAll.html#pyspark.sql.DataFrame.intersectAll)(other) | Return a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) containing rows in both this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) and another [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) while preserving duplicates. | | [**isLocal**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.isLocal.html#pyspark.sql.DataFrame.isLocal)() | Returns True if the [**collect()**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.collect.html#pyspark.sql.DataFrame.collect) and [**take()**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.take.html#pyspark.sql.DataFrame.take) methods can be run locally (without any Spark executors). | | [**join**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.join.html#pyspark.sql.DataFrame.join)(other[, on, how]) | Joins with another [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame), using the given join expression. | | [**limit**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.limit.html#pyspark.sql.DataFrame.limit)(num) | Limits the result count to the number specified. | | [**localCheckpoint**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.localCheckpoint.html#pyspark.sql.DataFrame.localCheckpoint)([eager]) | Returns a locally checkpointed version of this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**mapInPandas**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.mapInPandas.html#pyspark.sql.DataFrame.mapInPandas)(func, schema) | Maps an iterator of batches in the current [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) using a Python native function that takes and outputs a pandas DataFrame, and returns the result as a [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**orderBy**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.orderBy.html#pyspark.sql.DataFrame.orderBy)(\*cols, \*\*kwargs) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) sorted by the specified column(s). | | [**persist**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.persist.html#pyspark.sql.DataFrame.persist)([storageLevel]) | Sets the storage level to persist the contents of the [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) across operations after the first time it is computed. | | [**printSchema**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.printSchema.html#pyspark.sql.DataFrame.printSchema)() | Prints out the schema in the tree format. | | [**randomSplit**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.randomSplit.html#pyspark.sql.DataFrame.randomSplit)(weights[, seed]) | Randomly splits this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) with the provided weights. | | [**registerTempTable**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.registerTempTable.html#pyspark.sql.DataFrame.registerTempTable)(name) | Registers this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) as a temporary table using the given name. | | [**repartition**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.repartition.html#pyspark.sql.DataFrame.repartition)(numPartitions, \*cols) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) partitioned by the given partitioning expressions. | | [**repartitionByRange**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.repartitionByRange.html#pyspark.sql.DataFrame.repartitionByRange)(numPartitions, \*cols) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) partitioned by the given partitioning expressions. | | [**replace**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.replace.html#pyspark.sql.DataFrame.replace)(to\_replace[, value, subset]) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) replacing a value with another value. | | [**rollup**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.rollup.html#pyspark.sql.DataFrame.rollup)(\*cols) | Create a multi-dimensional rollup for the current [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) using the specified columns, so we can run aggregation on them. | | [**sameSemantics**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.sameSemantics.html#pyspark.sql.DataFrame.sameSemantics)(other) | Returns *True* when the logical query plans inside both [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame)s are equal and therefore return same results. | | [**sample**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.sample.html#pyspark.sql.DataFrame.sample)([withReplacement, fraction, seed]) | Returns a sampled subset of this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**sampleBy**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.sampleBy.html#pyspark.sql.DataFrame.sampleBy)(col, fractions[, seed]) | Returns a stratified sample without replacement based on the fraction given on each stratum. | | [**select**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.select.html#pyspark.sql.DataFrame.select)(\*cols) | Projects a set of expressions and returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**selectExpr**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.selectExpr.html#pyspark.sql.DataFrame.selectExpr)(\*expr) | Projects a set of SQL expressions and returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**semanticHash**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.semanticHash.html#pyspark.sql.DataFrame.semanticHash)() | Returns a hash code of the logical query plan against this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**show**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.show.html#pyspark.sql.DataFrame.show)([n, truncate, vertical]) | Prints the first n rows to the console. | | [**sort**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.sort.html#pyspark.sql.DataFrame.sort)(\*cols, \*\*kwargs) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) sorted by the specified column(s). | | [**sortWithinPartitions**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.sortWithinPartitions.html#pyspark.sql.DataFrame.sortWithinPartitions)(\*cols, \*\*kwargs) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) with each partition sorted by the specified column(s). | | [**subtract**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.subtract.html#pyspark.sql.DataFrame.subtract)(other) | Return a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) containing rows in this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) but not in another [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**summary**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.summary.html#pyspark.sql.DataFrame.summary)(\*statistics) | Computes specified statistics for numeric and string columns. | | [**tail**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.tail.html#pyspark.sql.DataFrame.tail)(num) | Returns the last num rows as a **list** of [**Row**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Row.html#pyspark.sql.Row). | | [**take**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.take.html#pyspark.sql.DataFrame.take)(num) | Returns the first num rows as a **list** of [**Row**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Row.html#pyspark.sql.Row). | | [**toDF**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.toDF.html#pyspark.sql.DataFrame.toDF)(\*cols) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) that with new specified column names | | [**toJSON**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.toJSON.html#pyspark.sql.DataFrame.toJSON)([use\_unicode]) | Converts a [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) into a **RDD** of string. | | [**toLocalIterator**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.toLocalIterator.html#pyspark.sql.DataFrame.toLocalIterator)([prefetchPartitions]) | Returns an iterator that contains all of the rows in this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**toPandas**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.toPandas.html#pyspark.sql.DataFrame.toPandas)() | Returns the contents of this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) as Pandas pandas.DataFrame. | | **to\_koalas**([index\_col]) |  | | [**to\_pandas\_on\_spark**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.to_pandas_on_spark.html#pyspark.sql.DataFrame.to_pandas_on_spark)([index\_col]) | Converts the existing DataFrame into a pandas-on-Spark DataFrame. | | [**transform**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.transform.html#pyspark.sql.DataFrame.transform)(func) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**union**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.union.html#pyspark.sql.DataFrame.union)(other) | Return a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) containing union of rows in this and another [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**unionAll**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.unionAll.html#pyspark.sql.DataFrame.unionAll)(other) | Return a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) containing union of rows in this and another [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**unionByName**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.unionByName.html#pyspark.sql.DataFrame.unionByName)(other[, allowMissingColumns]) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) containing union of rows in this and another [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**unpersist**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.unpersist.html#pyspark.sql.DataFrame.unpersist)([blocking]) | Marks the [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) as non-persistent, and remove all blocks for it from memory and disk. | | [**where**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.where.html#pyspark.sql.DataFrame.where)(condition) | [**where()**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.where.html#pyspark.sql.DataFrame.where) is an alias for [**filter()**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.filter.html#pyspark.sql.DataFrame.filter). | | [**withColumn**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.withColumn.html#pyspark.sql.DataFrame.withColumn)(colName, col) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) by adding a column or replacing the existing column that has the same name. | | [**withColumnRenamed**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.withColumnRenamed.html#pyspark.sql.DataFrame.withColumnRenamed)(existing, new) | Returns a new [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) by renaming an existing column. | | [**withWatermark**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.withWatermark.html#pyspark.sql.DataFrame.withWatermark)(eventTime, delayThreshold) | Defines an event time watermark for this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). | | [**writeTo**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.writeTo.html#pyspark.sql.DataFrame.writeTo)(table) | Create a write configuration builder for v2 sources. | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Attributes**   |  |  | | --- | --- | | [**columns**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.columns.html#pyspark.sql.DataFrame.columns) | Returns all column names as a list. | | [**dtypes**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.dtypes.html#pyspark.sql.DataFrame.dtypes) | Returns all column names and their data types as a list. | | [**isStreaming**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.isStreaming.html#pyspark.sql.DataFrame.isStreaming) | Returns True if this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) contains one or more sources that continuously return data as it arrives. | | [**na**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.na.html#pyspark.sql.DataFrame.na) | Returns a [**DataFrameNaFunctions**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrameNaFunctions.html#pyspark.sql.DataFrameNaFunctions) for handling missing values. | | [**rdd**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.rdd.html#pyspark.sql.DataFrame.rdd) | Returns the content as an [**pyspark.RDD**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.html#pyspark.RDD) of [**Row**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Row.html#pyspark.sql.Row). | | [**schema**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.schema.html#pyspark.sql.DataFrame.schema) | Returns the schema of this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) as a [**pyspark.sql.types.StructType**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.types.StructType.html#pyspark.sql.types.StructType). | | [**stat**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.stat.html#pyspark.sql.DataFrame.stat) | Returns a [**DataFrameStatFunctions**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrameStatFunctions.html#pyspark.sql.DataFrameStatFunctions) for statistic functions. | | [**storageLevel**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.storageLevel.html#pyspark.sql.DataFrame.storageLevel) | Get the [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame)’s current storage level. | | [**write**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.write.html#pyspark.sql.DataFrame.write) | Interface for saving the content of the non-streaming [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) out into external storage. | | [**writeStream**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.writeStream.html#pyspark.sql.DataFrame.writeStream) | Interface for saving the content of the streaming [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) out into external storage. | |

|  |
| --- |
| * + [pyspark.pandas.DataFrame](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.html)   + [pyspark.pandas.DataFrame.index](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.index.html)   + [pyspark.pandas.DataFrame.columns](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.columns.html)   + [pyspark.pandas.DataFrame.empty](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.empty.html)   + [pyspark.pandas.DataFrame.dtypes](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.dtypes.html)   + [pyspark.pandas.DataFrame.shape](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.shape.html)   + [pyspark.pandas.DataFrame.axes](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.axes.html)   + [pyspark.pandas.DataFrame.ndim](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.ndim.html)   + [pyspark.pandas.DataFrame.size](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.size.html)   + [pyspark.pandas.DataFrame.select\_dtypes](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.select_dtypes.html)   + [pyspark.pandas.DataFrame.values](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.values.html)   + [pyspark.pandas.DataFrame.copy](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.copy.html)   + [pyspark.pandas.DataFrame.isna](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.isna.html)   + [pyspark.pandas.DataFrame.astype](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.astype.html)   + [pyspark.pandas.DataFrame.isnull](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.isnull.html)   + [pyspark.pandas.DataFrame.notna](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.notna.html)   + [pyspark.pandas.DataFrame.notnull](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.notnull.html)   + [pyspark.pandas.DataFrame.pad](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.pad.html)   + [pyspark.pandas.DataFrame.bool](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.bool.html)   + [pyspark.pandas.DataFrame.at](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.at.html)   + [pyspark.pandas.DataFrame.iat](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.iat.html)   + [pyspark.pandas.DataFrame.head](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.head.html)   + [pyspark.pandas.DataFrame.idxmax](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.idxmax.html)   + [pyspark.pandas.DataFrame.idxmin](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.idxmin.html)   + [pyspark.pandas.DataFrame.loc](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.loc.html)   + [pyspark.pandas.DataFrame.iloc](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.iloc.html)   + [pyspark.pandas.DataFrame.items](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.items.html)   + [pyspark.pandas.DataFrame.iteritems](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.iteritems.html)   + [pyspark.pandas.DataFrame.iterrows](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.iterrows.html)   + [pyspark.pandas.DataFrame.itertuples](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.itertuples.html)   + [pyspark.pandas.DataFrame.keys](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.keys.html)   + [pyspark.pandas.DataFrame.pop](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.pop.html)   + [pyspark.pandas.DataFrame.tail](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.tail.html)   + [pyspark.pandas.DataFrame.xs](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.xs.html)   + [pyspark.pandas.DataFrame.get](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.get.html)   + [pyspark.pandas.DataFrame.where](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.where.html)   + [pyspark.pandas.DataFrame.mask](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.mask.html)   + [pyspark.pandas.DataFrame.query](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.query.html)   + [pyspark.pandas.DataFrame.add](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.add.html)   + [pyspark.pandas.DataFrame.radd](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.radd.html)   + [pyspark.pandas.DataFrame.div](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.div.html)   + [pyspark.pandas.DataFrame.rdiv](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.rdiv.html)   + [pyspark.pandas.DataFrame.truediv](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.truediv.html)   + [pyspark.pandas.DataFrame.rtruediv](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.rtruediv.html)   + [pyspark.pandas.DataFrame.mul](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.mul.html)   + [pyspark.pandas.DataFrame.rmul](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.rmul.html)   + [pyspark.pandas.DataFrame.sub](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.sub.html)   + [pyspark.pandas.DataFrame.rsub](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.rsub.html)   + [pyspark.pandas.DataFrame.pow](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.pow.html)   + [pyspark.pandas.DataFrame.rpow](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.rpow.html)   + [pyspark.pandas.DataFrame.mod](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.mod.html)   + [pyspark.pandas.DataFrame.rmod](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.rmod.html)   + [pyspark.pandas.DataFrame.floordiv](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.floordiv.html)   + [pyspark.pandas.DataFrame.rfloordiv](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.rfloordiv.html)   + [pyspark.pandas.DataFrame.lt](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.lt.html)   + [pyspark.pandas.DataFrame.gt](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.gt.html)   + [pyspark.pandas.DataFrame.le](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.le.html)   + [pyspark.pandas.DataFrame.ge](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.ge.html)   + [pyspark.pandas.DataFrame.ne](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.ne.html)   + [pyspark.pandas.DataFrame.eq](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.eq.html)   + [pyspark.pandas.DataFrame.dot](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.dot.html)   + [pyspark.pandas.DataFrame.apply](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.apply.html)   + [pyspark.pandas.DataFrame.applymap](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.applymap.html)   + [pyspark.pandas.DataFrame.pipe](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.pipe.html)   + [pyspark.pandas.DataFrame.agg](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.agg.html)   + [pyspark.pandas.DataFrame.aggregate](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.aggregate.html)   + [pyspark.pandas.DataFrame.groupby](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.groupby.html)   + [pyspark.pandas.DataFrame.rolling](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.rolling.html)   + [pyspark.pandas.DataFrame.expanding](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.expanding.html)   + [pyspark.pandas.DataFrame.transform](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.transform.html)   + [pyspark.pandas.DataFrame.abs](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.abs.html)   + [pyspark.pandas.DataFrame.all](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.all.html)   + [pyspark.pandas.DataFrame.any](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.any.html)   + [pyspark.pandas.DataFrame.clip](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.clip.html)   + [pyspark.pandas.DataFrame.corr](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.corr.html)   + [pyspark.pandas.DataFrame.count](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.count.html)   + [pyspark.pandas.DataFrame.describe](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.describe.html)   + [pyspark.pandas.DataFrame.kurt](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.kurt.html)   + [pyspark.pandas.DataFrame.kurtosis](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.kurtosis.html)   + [pyspark.pandas.DataFrame.mad](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.mad.html)   + [pyspark.pandas.DataFrame.max](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.max.html)   + [pyspark.pandas.DataFrame.mean](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.mean.html)   + [pyspark.pandas.DataFrame.min](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.min.html)   + [pyspark.pandas.DataFrame.median](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.median.html)   + [pyspark.pandas.DataFrame.pct\_change](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.pct_change.html)   + [pyspark.pandas.DataFrame.prod](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.prod.html)   + [pyspark.pandas.DataFrame.product](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.product.html)   + [pyspark.pandas.DataFrame.quantile](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.quantile.html)   + [pyspark.pandas.DataFrame.nunique](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.nunique.html)   + [pyspark.pandas.DataFrame.sem](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.sem.html)   + [pyspark.pandas.DataFrame.skew](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.skew.html)   + [pyspark.pandas.DataFrame.sum](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.sum.html)   + [pyspark.pandas.DataFrame.std](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.std.html)   + [pyspark.pandas.DataFrame.var](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.var.html)   + [pyspark.pandas.DataFrame.cummin](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.cummin.html)   + [pyspark.pandas.DataFrame.cummax](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.cummax.html)   + [pyspark.pandas.DataFrame.cumsum](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.cumsum.html)   + [pyspark.pandas.DataFrame.cumprod](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.cumprod.html)   + [pyspark.pandas.DataFrame.round](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.round.html)   + [pyspark.pandas.DataFrame.diff](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.diff.html)   + [pyspark.pandas.DataFrame.eval](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.eval.html)   + [pyspark.pandas.DataFrame.add\_prefix](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.add_prefix.html)   + [pyspark.pandas.DataFrame.add\_suffix](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.add_suffix.html)   + [pyspark.pandas.DataFrame.align](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.align.html)   + [pyspark.pandas.DataFrame.at\_time](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.at_time.html)   + [pyspark.pandas.DataFrame.between\_time](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.between_time.html)   + [pyspark.pandas.DataFrame.drop](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.drop.html)   + [pyspark.pandas.DataFrame.droplevel](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.droplevel.html)   + [pyspark.pandas.DataFrame.drop\_duplicates](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.drop_duplicates.html)   + [pyspark.pandas.DataFrame.duplicated](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.duplicated.html)   + [pyspark.pandas.DataFrame.equals](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.equals.html)   + [pyspark.pandas.DataFrame.filter](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.filter.html)   + [pyspark.pandas.DataFrame.first](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.first.html)   + [pyspark.pandas.DataFrame.head](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.head.html)   + [pyspark.pandas.DataFrame.last](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.last.html)   + [pyspark.pandas.DataFrame.rename](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.rename.html)   + [pyspark.pandas.DataFrame.rename\_axis](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.rename_axis.html)   + [pyspark.pandas.DataFrame.reset\_index](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.reset_index.html)   + [pyspark.pandas.DataFrame.set\_index](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.set_index.html)   + [pyspark.pandas.DataFrame.swapaxes](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.swapaxes.html)   + [pyspark.pandas.DataFrame.swaplevel](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.swaplevel.html)   + [pyspark.pandas.DataFrame.take](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.take.html)   + [pyspark.pandas.DataFrame.isin](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.isin.html)   + [pyspark.pandas.DataFrame.sample](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.sample.html)   + [pyspark.pandas.DataFrame.truncate](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.truncate.html)   + [pyspark.pandas.DataFrame.backfill](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.backfill.html)   + [pyspark.pandas.DataFrame.dropna](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.dropna.html)   + [pyspark.pandas.DataFrame.fillna](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.fillna.html)   + [pyspark.pandas.DataFrame.replace](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.replace.html)   + [pyspark.pandas.DataFrame.bfill](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.bfill.html)   + [pyspark.pandas.DataFrame.ffill](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.ffill.html)   + [pyspark.pandas.DataFrame.pivot\_table](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.pivot_table.html)   + [pyspark.pandas.DataFrame.pivot](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.pivot.html)   + [pyspark.pandas.DataFrame.sort\_index](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.sort_index.html)   + [pyspark.pandas.DataFrame.sort\_values](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.sort_values.html)   + [pyspark.pandas.DataFrame.nlargest](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.nlargest.html)   + [pyspark.pandas.DataFrame.nsmallest](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.nsmallest.html)   + [pyspark.pandas.DataFrame.stack](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.stack.html)   + [pyspark.pandas.DataFrame.unstack](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.unstack.html)   + [pyspark.pandas.DataFrame.melt](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.melt.html)   + [pyspark.pandas.DataFrame.explode](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.explode.html)   + [pyspark.pandas.DataFrame.squeeze](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.squeeze.html)   + [**pyspark.pandas.DataFrame.T**](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.T.html)   + [pyspark.pandas.DataFrame.transpose](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.transpose.html)   + [pyspark.pandas.DataFrame.reindex](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.reindex.html)   + [pyspark.pandas.DataFrame.reindex\_like](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.reindex_like.html)   + [pyspark.pandas.DataFrame.rank](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.rank.html)   + [pyspark.pandas.DataFrame.append](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.append.html)   + [pyspark.pandas.DataFrame.assign](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.assign.html)   + [pyspark.pandas.DataFrame.merge](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.merge.html)   + [pyspark.pandas.DataFrame.join](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.join.html)   + [pyspark.pandas.DataFrame.update](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.update.html)   + [pyspark.pandas.DataFrame.insert](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.insert.html)   + [pyspark.pandas.DataFrame.shift](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.shift.html)   + [pyspark.pandas.DataFrame.first\_valid\_index](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.first_valid_index.html)   + [pyspark.pandas.DataFrame.last\_valid\_index](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.last_valid_index.html)   + [pyspark.pandas.DataFrame.from\_records](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.from_records.html)   + [pyspark.pandas.DataFrame.info](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.info.html)   + [pyspark.pandas.DataFrame.to\_table](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_table.html)   + [pyspark.pandas.DataFrame.to\_delta](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_delta.html)   + [pyspark.pandas.DataFrame.to\_parquet](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_parquet.html)   + [pyspark.pandas.DataFrame.to\_spark\_io](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_spark_io.html)   + [pyspark.pandas.DataFrame.to\_csv](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_csv.html)   + [pyspark.pandas.DataFrame.to\_pandas](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_pandas.html)   + [pyspark.pandas.DataFrame.to\_html](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_html.html)   + [pyspark.pandas.DataFrame.to\_numpy](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_numpy.html)   + [pyspark.pandas.DataFrame.to\_spark](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_spark.html)   + [pyspark.pandas.DataFrame.to\_string](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_string.html)   + [pyspark.pandas.DataFrame.to\_json](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_json.html)   + [pyspark.pandas.DataFrame.to\_dict](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_dict.html)   + [pyspark.pandas.DataFrame.to\_excel](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_excel.html)   + [pyspark.pandas.DataFrame.to\_clipboard](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_clipboard.html)   + [pyspark.pandas.DataFrame.to\_markdown](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_markdown.html)   + [pyspark.pandas.DataFrame.to\_records](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_records.html)   + [pyspark.pandas.DataFrame.to\_latex](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.to_latex.html)   + [pyspark.pandas.DataFrame.style](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.style.html)   + [pyspark.pandas.DataFrame.spark.frame](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.spark.frame.html)   + [pyspark.pandas.DataFrame.spark.cache](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.spark.cache.html)   + [pyspark.pandas.DataFrame.spark.persist](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.spark.persist.html)   + [pyspark.pandas.DataFrame.spark.hint](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.spark.hint.html)   + [pyspark.pandas.DataFrame.spark.to\_table](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.spark.to_table.html)   + [pyspark.pandas.DataFrame.spark.to\_spark\_io](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.spark.to_spark_io.html)   + [pyspark.pandas.DataFrame.spark.apply](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.spark.apply.html)   + [pyspark.pandas.DataFrame.spark.repartition](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.spark.repartition.html)   + [pyspark.pandas.DataFrame.spark.coalesce](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.spark.coalesce.html)   + [pyspark.pandas.DataFrame.plot](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.plot.html)   + [pyspark.pandas.DataFrame.plot.area](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.plot.area.html)   + [pyspark.pandas.DataFrame.plot.barh](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.plot.barh.html)   + [pyspark.pandas.DataFrame.plot.bar](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.plot.bar.html)   + [pyspark.pandas.DataFrame.plot.hist](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.plot.hist.html)   + [pyspark.pandas.DataFrame.plot.line](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.plot.line.html)   + [pyspark.pandas.DataFrame.plot.pie](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.plot.pie.html)   + [pyspark.pandas.DataFrame.plot.scatter](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.plot.scatter.html)   + [pyspark.pandas.DataFrame.plot.density](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.plot.density.html)   + [pyspark.pandas.DataFrame.hist](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.hist.html)   + [pyspark.pandas.DataFrame.kde](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.kde.html)   + [pyspark.pandas.DataFrame.pandas\_on\_spark.apply\_batch](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.pandas_on_spark.apply_batch.html)   + [pyspark.pandas.DataFrame.pandas\_on\_spark.transform\_batch](https://spark.apache.org/docs/latest/api/python/reference/pyspark.pandas/api/pyspark.pandas.DataFrame.pandas_on_spark.transform_batch.html) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CREATE DATAFRAME**   |  | | --- | |  | | **SPARKSESSION** | **RDD** | **DATAFRAME** | | createDataFrame(rdd) | toDF() | toDF(\*cols) | | createDataFrame(dataList) | toDF(\*cols) |  | | createDataFrame(rowData,columns) |  |  | | createDataFrame(dataList,schema) |  |  |   columns = ["language","users\_count"]  data = [("Java", "20000"), ("Python", "100000"), ("Scala", "3000")]   * 1. **Using toDF() function**   dfFromRDD1 = rdd.toDF()  dfFromRDD1.printSchema()  columns = ["language","users\_count"]  dfFromRDD1 = rdd.toDF(columns)  dfFromRDD1.printSchema()  By default, the datatype of these columns infers to the type of data. We can change this behavior by supplying schema, where we can specify a column name, data type, and nullable for each field/column.  **1.2 Using createDataFrame() from SparkSession**  dfFromRDD2 = spark.createDataFrame(rdd).toDF(\*columns)  **2.1 Using createDataFrame() from SparkSession**  dfFromData2 = spark.createDataFrame(data).toDF(\*columns)  **2.2 Using createDataFrame() with the Row type**  rowData = map(lambda x: Row(\*x), data)  dfFromData3 = spark.createDataFrame(rowData,columns)  **2.3 Create DataFrame with schema**  from pyspark.sql.types import StructType,StructField, StringType, IntegerType  data2 = [("James","","Smith","36636","M",3000),  ("Michael","Rose","","40288","M",4000),  ]  schema = StructType([ \  StructField("firstname",StringType(),True), \  StructField("salary", IntegerType(), True) \  ])  df = spark.createDataFrame(data=data2,schema=schema)  df.printSchema()  df.show(truncate=False)  **3. Create DataFrame from Data sources**  df2 = spark.read.csv("/src/resources/file.csv")  df3 = spark.read.text("/src/resources/file.txt")  df4 = spark.read.json("/src/resources/file.json")  <https://sparkbyexamples.com/pyspark/pyspark-read-and-write-parquet-file/>  **2.3 Create DataFrame with schema**  from pyspark.sql.types import StructType,StructField, StringType, IntegerType  data2 = [("James","","Smith","36636","M",3000),  ("Michael","Rose","","40288","M",4000),  ]  schema = StructType([ \  StructField("firstname",StringType(),True), \  StructField("salary", IntegerType(), True) \  ])  df = spark.createDataFrame(data=data2,schema=schema)  df.printSchema()  df.show(truncate=False) |

|  |
| --- |
| **CREATING EMPTY DATAFRAMES**  While working with files, sometimes we may not receive a file for processing, however, we still need to create a DataFrame manually with the same schema we expect. If we don’t create with the same schema, our operations/transformations (like union’s) on DataFrame fail as we refer to the columns that may not present.  To handle situations similar to these, we always need to create a DataFrame with the same schema, which means the same column names and datatypes regardless of the file exists or empty file processing.  1. Create Empty RDD in PySpark  from pyspark.sql import SparkSession  spark = SparkSession.builder.getOrCreate()  emptyRDD = spark.sparkContext.emptyRDD()  # or  rdd2= spark.sparkContext.parallelize([])  2. Create Empty DataFrame with Schema (StructType)  In order to create an empty PySpark DataFrame manually with schema ( column names & data types) first, Create a schema using StructType and StructField .  from pyspark.sql.types import StructType,StructField, StringType  schema = StructType([  StructField('firstname', StringType(), True),  StructField('middlename', StringType(), True),  ])  df = spark.createDataFrame(emptyRDD,schema)  3. Convert Empty RDD to DataFrame  #Convert empty RDD to Dataframe  df1 = emptyRDD.toDF(schema)  4. Create Empty DataFrame with Schema.  #Create empty DataFrame directly.  df2 = spark.createDataFrame([], schema)  5. Create Empty DataFrame without Schema (no columns)  #Create empty DatFrame with no schema (no columns)  df3 = spark.createDataFrame([], StructType([]))  df3.printSchema() |

|  |
| --- |
| **Convert PySpark RDD to DataFrame**  <https://sparkbyexamples.com/pyspark/convert-pyspark-rdd-to-dataframe/> |

|  |
| --- |
| **Convert PySpark DataFrame to Pandas**   * PySpark DataFrame can be converted to Python pandas DataFrame using a function toPandas(). * operations on Pyspark run faster than Pandas due to its distributed nature and parallel execution on multiple cores * pandas run operations on a single node whereas PySpark runs on multiple machines. * PySpark processes operations many times faster than pandas. * toPandas() results in the collection of all records in the PySpark DataFrame to the driver program and should be done only on a small subset of the data. running on larger dataset’s results in memory error and crashes the application. To deal with a larger dataset, you can also try increasing memory on the driver. * pandas add a sequence number to the result as a row Index. You can rename pandas columns by using rename() function. * can rename pandas columns by using rename() function.   pysparkDF = spark.createDataFrame(data, schema = columns)  pandasDF = pysparkDF.toPandas()  print(pandasDF)  # Nested structure elements  from pyspark.sql.types import StructType, StructField, StringType,IntegerType  dataStruct = [(("James","","Smith"),"36636","M","3000"), \  (("Michael","Rose",""),"40288","M","4000"), \  (("Robert","","Williams"),"42114","M","4000"), \  (("Maria","Anne","Jones"),"39192","F","4000"), \  (("Jen","Mary","Brown"),"","F","-1") \  ]  schemaStruct = StructType([  StructField('name', StructType([  StructField('firstname', StringType(), True),  StructField('middlename', StringType(), True),  StructField('lastname', StringType(), True)  ])),  StructField('dob', StringType(), True),  StructField('gender', StringType(), True),  StructField('salary', StringType(), True)  ])  df = spark.createDataFrame(data=dataStruct, schema = schemaStruct)  df.printSchema()  pandasDF2 = df.toPandas()  print(pandasDF2) |

|  |
| --- |
| **pyspark.sql.DataFrameReader.csv**(path, schema=None, sep=None, encoding=None, quote=None, escape=None, comment=None, header=None, inferSchema=None…..etc)  Loads a CSV file and returns the result as a DataFrame.  This function will go through the input once to determine the input schema if inferSchema is enabled. To avoid going through the entire data once, disable inferSchema option or specify the schema explicitly using schema.  >>>df = spark.read.csv('python/test\_support/sql/ages.csv')  >>>df.dtypes  [('\_c0', 'string'), ('\_c1', 'string')]  >>>rdd = sc.textFile('python/test\_support/sql/ages.csv')  >>>df2 = spark.read.csv(rdd)  >>>df2.dtypes  [('\_c0', 'string'), ('\_c1', 'string')] |

|  |
| --- |
| **pyspark.sql.DataFrameReader.format(source)**  df = spark.read.format('json').load('python/test\_support/sql/people.json') |

|  |
| --- |
| **pyspark.sql.DataFrameReader.jdbc**(url, table, column=None, lowerBound=None, upperBound=None, numPartitions=None, predicates=None, properties=None)[source] |

|  |
| --- |
| **pyspark.sql.DataFrameReader.json((path, schema=None,….etc)**  Loads JSON files and returns the results as a DataFrame.  JSON Lines (newline-delimited JSON) is supported by default. For JSON (one record per file), set the multiLine parameter to true.  If the schema parameter is not specified, this function goes through the input once to determine the input schema.  df1 = spark.read.json('python/test\_support/sql/people.json')  rdd = sc.textFile('python/test\_support/sql/people.json')  df2 = spark.read.json(rdd) |

|  |
| --- |
| **pyspark.sql.DataFrameReader.load(path=None, format=None, schema=None, \*\*options)**  df = spark.read.format("parquet").load('python/test\_support/sql/parquet\_partitioned',  opt1=True, opt2=1, opt3='str')  df = spark.read.format('json').load(['python/test\_support/sql/people.json',  'python/test\_support/sql/people1.json']) |

|  |
| --- |
| **pyspark.sql.DataFrameReader.option(key, value)**  **pyspark.sql.DataFrameReader.options(\*\*options)**  Adds an input option for the underlying data source. |

|  |
| --- |
| **pyspark.sql.DataFrameReader.orc (**path, mergeSchema=None, pathGlobFilter=None, recursiveFileLookup=None, modifiedBefore=None, modifiedAfter=None)[source]¶ |

|  |
| --- |
| **pyspark.sql.DataFrameReader.parquet(\*paths, \*\*options)**  df = spark.read.parquet('python/test\_support/sql/parquet\_partitioned') |

|  |
| --- |
| **pyspark.sql.DataFrameReader.schema(schema)**  Some data sources (e.g. JSON) can infer the input schema automatically from data. By specifying the schema here, the underlying data source can skip the schema inference step, and thus speed up data loading.  s = spark.read.schema(“col0 INT, col1 DOUBLE”) |

|  |
| --- |
| **pyspark.sql.DataFrameReader.table(tablename)**  Returns the specified table as a DataFrame.  >>>df = spark.read.parquet('python/test\_support/sql/parquet\_partitioned')  >>>df.createOrReplaceTempView('tmpTable')  >>>spark.read.table('tmpTable').dtypes  [('name', 'string'), ('year', 'int'), ('month', 'int'), ('day', 'int')] |

|  |
| --- |
| **pyspark.sql.DataFrameWriter.bucketBy(numBuckets, col, \*cols)** |

|  |
| --- |
| **pyspark.sql.DataFrameWriter.csv(path, mode=None, compression=None, sep=None, quote=None, escape=None, header=None, nullValue=None, escapeQuotes=None, quoteAll=None, dateFormat=None, timestampFormat=None, ignoreLeadingWhiteSpace=None, ignoreTrailingWhiteSpace=None, charToEscapeQuoteEscaping=None, encoding=None, emptyValue=None, lineSep=None)**  df.write.csv(os.path.join(tempfile.mkdtemp(), 'data')) |

|  |
| --- |
| **pyspark.sql.DataFrameWriter.format(source)**  Specifies the underlying output data source.  df.write.format('json').save(os.path.join(tempfile.mkdtemp(), 'data')) |

|  |
| --- |
| **pyspark.sql.DataFrameWriter.insertInto(tableName, overwrite=None)**  Inserts the content of the DataFrame to the specified table.  It requires that the schema of the DataFrame is the same as the schema of the table. |

|  |
| --- |
| pyspark.sql.DataFrameWriter.jdbc(url, table, mode=None, properties=None)  Saves the content of the DataFrame to an external database table via JDBC.  etc |

|  |
| --- |
| pyspark.sql.DataFrameWriter.json(path, mode=None, compression=None, dateFormat=None, timestampFormat=None, lineSep=None, encoding=None, ignoreNullFields=None)  **path*str***  the path in any Hadoop supported file system  **mode*str, optional***  specifies the behavior of the save operation when data already exists.   * append: Append contents of this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) to existing data. * overwrite: Overwrite existing data. * ignore: Silently ignore this operation if data already exists. * error or errorifexists (default case): Throw an exception if data already exists.   df.write.json(os.path.join(tempfile.mkdtemp(), 'data')) |

|  |
| --- |
| pyspark.sql.DataFrameWriter.mode(savemode)  Specifies the behavior when data or table already exists.  df.write.mode('append').parquet(os.path.join(tempfile.mkdtemp(), 'data')) |

|  |
| --- |
| pyspark.sql.DataFrameWriter.option(key, value)  pyspark.sql.DataFrameWriter.options(\*\*options)  Adds an output option for the underlying data source. |

|  |
| --- |
| pyspark.sql.DataFrameWriter.orc…etc |

|  |
| --- |
| pyspark.sql.DataFrameWriter.parquet (path, mode=None, partitionBy=None, compression=None)  **Parameters**  **path*str***  the path in any Hadoop supported file system  **mode*str, optional***  specifies the behavior of the save operation when data already exists.   * append: Append contents of this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) to existing data. * overwrite: Overwrite existing data. * ignore: Silently ignore this operation if data already exists. * error or errorifexists (default case): Throw an exception if data already exists.   **partitionBy*str or list, optional***  names of partitioning columns  **Other Parameters**  **Extra options**  For the extra options, refer to Data Source Option in the version you use. |

|  |
| --- |
| pyspark.sql.DataFrameWriter.partitionBy(\*cols)  Partitions the output by the given columns on the file system.  If specified, the output is laid out on the file system similar to Hive’s partitioning scheme.  df.write.partitionBy('year', 'month').parquet(os.path.join(tempfile.mkdtemp(), 'data')) |

|  |
| --- |
| pyspark.sql.DataFrameWriter.save(path=None, format=None, mode=None, partitionBy=None, \*\*options)  Saves the contents of the DataFrame to a data source.  The data source is specified by the format and a set of options. If format is not specified, the default data source configured by spark.sql.sources.default will be used. |

|  |
| --- |
| pyspark.sql.DataFrameWriter.saveAsTable(name, format=None, mode=None, partitionBy=None, \*\*options)  Saves the content of the [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) as the specified table.  In the case the table already exists, behavior of this function depends on the save mode, specified by the mode function (default to throwing an exception). When mode is Overwrite, the schema of the [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) does not need to be the same as that of the existing table.   * append: Append contents of this [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame) to existing data. * overwrite: Overwrite existing data. * error or errorifexists: Throw an exception if data already exists. * ignore: Silently ignore this operation if data already exists.   When mode is Append, if there is an existing table, we will use the format and options of the existing table. The column order in the schema of the DataFrame doesn’t need to be same as that of the existing table. Unlike DataFrameWriter.insertInto(), DataFrameWriter.saveAsTable() will use the column names to find the correct column positions. |

|  |
| --- |
| pyspark.sql.DataFrameWriter.sortBy(col, \*cols)  Sorts the output in each bucket by the given columns on the file system.  (df.write.format('parquet')  .bucketBy(100, 'year', 'month')  .sortBy('day')  .mode("overwrite")  .saveAsTable('sorted\_bucketed\_table')) |

|  |
| --- |
| pyspark.sql.DataFrameWriter.text(path, compression=None, lineSep=None)  Saves the content of the DataFrame in a text file at the specified path. The text files will be encoded as UTF-8.  The DataFrame must have only one column that is of string type.  Each row becomes a new line in the output file. |

|  |
| --- |
| pyspark.sql.DataFrame.agg(\*exprs)  Aggregate on the entire DataFrame without groups (shorthand for df.groupBy().agg()).  df.agg({"age": "max"}).collect() |

|  |
| --- |
| pyspark.sql.DataFrame.alias(alias)  Returns a new DataFrame with an alias set.  >>>from pyspark.sql.functions import \*  >>>df\_as1 = df.alias("df\_as1")  >>>df\_as2 = df.alias("df\_as2")  >>>joined\_df = df\_as1.join(df\_as2, col("df\_as1.name") == col("df\_as2.name"), 'inner')  >>>joined\_df.select("df\_as1.name", "df\_as2.name", "df\_as2.age") .sort(desc("df\_as1.name")).collect()  [Row(name='Bob', name='Bob', age=5), Row(name='Alice', name='Alice', age=2)] |

|  |
| --- |
| pyspark.sql.DataFrame.cache()  Persists the DataFrame with the default storage level (MEMORY\_AND\_DISK). |

|  |
| --- |
| pyspark.sql.DataFrame.checkpoint(eager=True)  Returns a checkpointed version of this DataFrame. Checkpointing can be used to truncate the logical plan of this DataFrame, 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 SparkContext.setCheckpointDir(). |

|  |
| --- |
| pyspark.sql.DataFrame.coalesce(numPartitions)  Returns a new DataFrame that has exactly numPartitions partitions.  Similar to coalesce defined on an RDD, 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.  However, if you’re doing a drastic coalesce, 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, you can 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).  df.coalesce(1).rdd.getNumPartitions() |

|  |
| --- |
| **pyspark.sql.DataFrame.colRegex(colName)**  Selects column based on the column name specified as a regex and returns it as Column.  df = spark.createDataFrame([("a", 1), ("b", 2), ("c", 3)], ["Col1", "Col2"])  df.select(df.colRegex("`(Col1)?+.+`")).show() |

|  |
| --- |
| **pyspark.sql.DataFrame.collect()**   * PySpark RDD/DataFrame collect() is an action operation that is used to retrieve all the elements of the dataset (from all nodes) to the driver node. * We should use the collect() on smaller dataset usually after filter(), group() e.t.c. Retrieving larger datasets results in OutOfMemory error. * Returns all the records as a list of Row. * collect() is an action hence it does not return a DataFrame instead, it returns data in an Array to the driver. * Once the data is in an array, you can use python for loop to process it further. * select() is a transformation that returns a new DataFrame and holds the columns that are selected whereas collect() is an action that returns the entire data set in an Array to the driver.   dataCollect = deptDF.collect()  print(dataCollect)  [Row(dept\_name='Finance', dept\_id=10),  Row(dept\_name='Marketing', dept\_id=20)]  for row in dataCollect:  print(row['dept\_name'] + "," +str(row['dept\_id']))  [Row(dept\_name='Finance', dept\_id=10),  Row(dept\_name='Marketing', dept\_id=20)  # Return the first element in an array (1st row).  deptDF.collect()[0]  Row(dept\_name='Finance', dept\_id=10)  #Returns value of First Row, First Column which is "Finance"  deptDF.collect()[0][0]  🡪‘Finance’  In case you want to just return certain elements of a DataFrame, you should call PySpark select() transformation first.  dataCollect = deptDF.select("dept\_name").collect() |

|  |
| --- |
| pyspark.sql.DataFrame.columns  >>>df.columns  ['age', 'name'] |

|  |
| --- |
| pyspark.sql.DataFrame.corr(col1, col2, method=None)[source]  The correlation method. Currently only supports “pearson” |

|  |
| --- |
| pyspark.sql.DataFrame.count()  Returns the number of rows in this DataFrame. |

|  |
| --- |
| pyspark.sql.DataFrame.cov(col1, col2)  Calculate the sample covariance for the given columns, specified by their names, as a double value. |

|  |
| --- |
| pyspark.sql.DataFrame.createGlobalTempView(name)  Creates a global temporary view with this DataFrame.  The lifetime of this temporary view is tied to this Spark application. throws TempTableAlreadyExistsException, if the view name already exists in the catalog.  >>>df.createGlobalTempView("people")  >>>df2 = spark.sql("select \* from global\_temp.people")  >>>sorted(df.collect()) == sorted(df2.collect())  True  >>>df.createGlobalTempView("people") |

|  |
| --- |
| pyspark.sql.DataFrame.createOrReplaceGlobalTempView(name)  Creates or replaces a global temporary view using the given name.  The lifetime of this temporary view is tied to this Spark application.  >>>df.createOrReplaceGlobalTempView("people")  >>>df2 = df.filter(df.age > 3)  >>>df2.createOrReplaceGlobalTempView("people")  >>>df3 = spark.sql("select \* from global\_temp.people")  >>>sorted(df3.collect()) == sorted(df2.collect())  True  >>>spark.catalog.dropGlobalTempView("people") |

|  |
| --- |
| pyspark.sql.DataFrame.createOrReplaceTempView(name)  Creates or replaces a local temporary view with this DataFrame.  The lifetime of this temporary table is tied to the SparkSession that was used to create this DataFrame.  df.createOrReplaceTempView("people")  df2 = df.filter(df.age > 3)  df2.createOrReplaceTempView("people")  df3 = spark.sql("select \* from people")  sorted(df3.collect()) == sorted(df2.collect())  spark.catalog.dropTempView("people") |

|  |
| --- |
| pyspark.sql.DataFrame.createTempView(name)  Creates a local temporary view with this DataFrame.  The lifetime of this temporary table is tied to the SparkSession that was used to create this DataFrame. throws TempTableAlreadyExistsException, if the view name already exists in the catalog.  df.createTempView("people")  df2 = spark.sql("select \* from people")  sorted(df.collect()) == sorted(df2.collect()) |

|  |
| --- |
| pyspark.sql.DataFrame.crossJoin(other)  Returns the cartesian product with another DataFrame.  df.select("age", "name").collect()  df2.select("name", "height").collect()  df.crossJoin(df2.select("height")).select("age", "name", "height").collect() |

|  |
| --- |
| pyspark.sql.DataFrame.cube(\*cols)  Create a multi-dimensional cube for the current DataFrame using the specified columns, so we can run aggregations on them.  df.cube("name", df.age).count().orderBy("name", "age").show() |

|  |
| --- |
| **pyspark.sql.DataFrame.describe(cols\*)**  Computes basic statistics for numeric and string columns. |

|  |
| --- |
| **pyspark.sql.DataFrame.distinct()**  Returns a new DataFrame containing the distinct rows in this DataFrame.  +-------------+----------+------+  |employee\_name|department|salary|  +-------------+----------+------+  |James |Sales |3000 |  |Michael |Sales |4600 |  Get Distinct Rows (By Comparing All Columns)  **distinctDF = df.distinct()**  print("Distinct count: "+str(distinctDF.count()))  distinctDF.show(truncate=False)  Alternatively, you can also run dropDuplicates() function which returns a new DataFrame after removing duplicate rows.  **df2 = df.dropDuplicates()**  print("Distinct count: "+str(df2.count()))  df2.show(truncate=False)  PySpark Distinct of Selected Multiple Columns:  PySpark doesn’t have a distinct method which takes columns that should run distinct on (drop duplicate rows on selected multiple columns) however, it provides another signature of dropDuplicates() function which takes multiple columns to eliminate duplicates.  dropDisDF = df.dropDuplicates(["department","salary"])  print("Distinct count of department & salary : "+str(dropDisDF.count()))  dropDisDF.show(truncate=False)  #Drop duplicates on selected columns  dropDisDF = df.dropDuplicates(["department","salary"])  print("Distinct count of department salary : "+str(dropDisDF.count()))  dropDisDF.show(truncate=False)  } |

|  |
| --- |
| **pyspark.sql.DataFrame.drop(cols\*)**  Returns a new DataFrame that drops the specified column. This is a no-op if schema doesn’t contain the given column name(s). |

|  |
| --- |
| **pyspark.sql.DataFrame.dropDuplicates(subset=None)**  Return a new DataFrame with duplicate rows removed, optionally only considering certain columns.  For a static batch DataFrame, it just drops duplicate rows. For a streaming DataFrame, it will keep all data across triggers as intermediate state to drop duplicates rows. You can use withWatermark() to limit how late the duplicate data can be and system will accordingly limit the state. In addition, too late data older than watermark will be dropped to avoid any possibility of duplicates.  drop\_duplicates() is an alias for dropDuplicates().  df.dropDuplicates(['name', 'height']).show() |

|  |
| --- |
| pyspark.sql.DataFrame.dropna(how='any', thresh=None, subset=None)  Returns a new DataFrame omitting rows with null values. DataFrame.dropna() and DataFrameNaFunctions.drop() are aliases of each other.  **how*str, optional***  ‘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.  **thresh: int, optional**  default None If specified, drop rows that have less than *thresh* non-null values. This overwrites the *how* parameter.  **subset*str, tuple or list, optional***  optional list of column names to consider.  df4.na.drop().show() |

|  |
| --- |
| pyspark.sql.DataFrame.dtypes  Returns all column names and their data types as a list. |

|  |
| --- |
| pyspark.sql.DataFrame.exceptAll(other)  Return a new DataFrame containing rows in this DataFrame but not in another DataFrame while preserving duplicates.  >>>df1 = spark.createDataFrame(  [("a", 1), ("a", 1), ("a", 1), ("a", 2), ("b", 3), ("c", 4)], ["C1", "C2"])  >>>df2 = spark.createDataFrame([("a", 1), ("b", 3)], ["C1", "C2"])  >>>df1.exceptAll(df2).show() |

|  |
| --- |
| pyspark.sql.DataFrame.explain(extended=None, mode=None)  Prints the (logical and physical) plans to the console for debugging purpose.  **extended*bool, optional***  default False. If False, prints only the physical plan. When this is a string without specifying the mode, it works as the mode is specified.  **mode*str, optional***  specifies the expected output format of plans.   * simple: Print only a physical plan. * extended: Print both logical and physical plans. * codegen: Print a physical plan and generated codes if they are available. * cost: Print a logical plan and statistics if they are available. * formatted: Split explain output into two sections: a physical plan outline and node details. |

|  |
| --- |
| pyspark.sql.DataFrame.fillna(value, subset=None)  Replace null values, alias for na.fill(). DataFrame.fillna() and DataFrameNaFunctions.fill() are aliases of each other.  df4.na.fill(50).show()  df5.na.fill(False).show()  df4.na.fill({'age': 50, 'name': 'unknown'}).show() |

|  |
| --- |
| pyspark.sql.DataFrame.filter(condition)  where() is an alias for filter().  df.filter(df.age > 3).collect()  df.where(df.age == 2).collect()  df.filter("age > 3").collect()  df.where("age = 2").collect() |

|  |
| --- |
| pyspark.sql.DataFrame.first()  Returns the first row as a Row. |

|  |
| --- |
| pyspark.sql.DataFrame.foreach(f)  Applies the f function to all Row of this DataFrame.  This is a shorthand for df.rdd.foreach().  def f(person):  print(person.name)  df.foreach(f) |

|  |
| --- |
| pyspark.sql.DataFrame.foreachPartition(f)  Applies the f function to each partition of this DataFrame.  This a shorthand for df.rdd.foreachPartition().  def f(people):  for person in people:  print(person.name)  df.foreachPartition(f) |

|  |
| --- |
| pyspark.sql.DataFrame.freqItems(cols, support=None)  … |

|  |
| --- |
| **pyspark.sql.DataFrame.groupBy(cols)**  PySpark groupBy() function is used to collect the identical data into groups on DataFrame and perform aggregate functions on the grouped data.  It returns GroupedData object which contains below aggregate functions:  Count(), Min(), Max(), Min()  Sum(), Avg(),Pivot()  Agg() : can calculate more than one aggregate at a time.  +-------------+----------+-----+------+---+-----+  |employee\_name|department|state|salary|age|bonus|  +-------------+----------+-----+------+---+-----+  | James| Sales| NY| 90000| 34|10000|  df.groupBy("department").sum("salary").show(truncate=False)  df.groupBy("department").count()  df.groupBy("department").min("salary")  df.groupBy("department").max("salary")  df.groupBy("department").avg( "salary")  df.groupBy("department").mean( "salary")  # group by on department,state and does sum() on salary and bonus columns.  df.groupBy("department","state") \  .sum("salary","bonus") \  .show(false)  Running more aggregates at a time  from pyspark.sql.functions import sum,avg,max,min,mean,count  df.groupBy("department") \  .agg(sum("salary").alias("sum\_salary"), \  avg("salary").alias("avg\_salary"), \  sum("bonus").alias("sum\_bonus"), \  max("bonus").alias("max\_bonus") \  ) \  .show(truncate=False)  Using filter on aggregate data  df.groupBy("department") \  .agg(sum("salary").alias("sum\_salary"), \  avg("salary").alias("avg\_salary"), \  sum("bonus").alias("sum\_bonus"), \  max("bonus").alias("max\_bonus")) \  .where(col("sum\_bonus") >= 50000) \  .show(truncate=False)  >>>df.groupBy().avg().collect()  [Row(avg(age)=3.5)]  >>>sorted(df.groupBy('name').agg({'age': 'mean'}).collect())  [Row(name='Alice', avg(age)=2.0), Row(name='Bob', avg(age)=5.0)]  >>>sorted(df.groupBy(df.name).avg().collect())  [Row(name='Alice', avg(age)=2.0), Row(name='Bob', avg(age)=5.0)]  >>>sorted(df.groupBy(['name', df.age]).count().collect())  [Row(name='Alice', age=2, count=1), Row(name='Bob', age=5, count=1)] |

|  |
| --- |
| pyspark.sql.DataFrame.head(n)  df.head(1) |

|  |
| --- |
| pyspark.sql.DataFrame.hint(name, \*parameters)  Specifies some hint on the current DataFrame.  … |

|  |
| --- |
| pyspark.sql.DataFrame.inputFiles()  Returns a best-effort snapshot of the files that compose this DataFrame. This method simply asks each constituent BaseRelation for its respective files and takes the union of all results. Depending on the source relations, this may not find all input files. Duplicates are removed.  >>df = spark.read.load("examples/src/main/resources/people.json", format="json")  >>len(df.inputFiles())  1 |

|  |
| --- |
| pyspark.sql.DataFrame.intersect(other)  Return a new DataFrame containing rows only in both this DataFrame and another DataFrame.  This is equivalent to INTERSECT in SQL. |

|  |
| --- |
| pyspark.sql.DataFrame.intersectAll(other)  Return a new DataFrame containing rows in both this DataFrame and another DataFrame while preserving duplicates.  This is equivalent to INTERSECT ALL in SQL. |

|  |
| --- |
| pyspark.sql.DataFrame.isLocal()  Returns True if the collect() and take() methods can be run locally (without any Spark executors) |

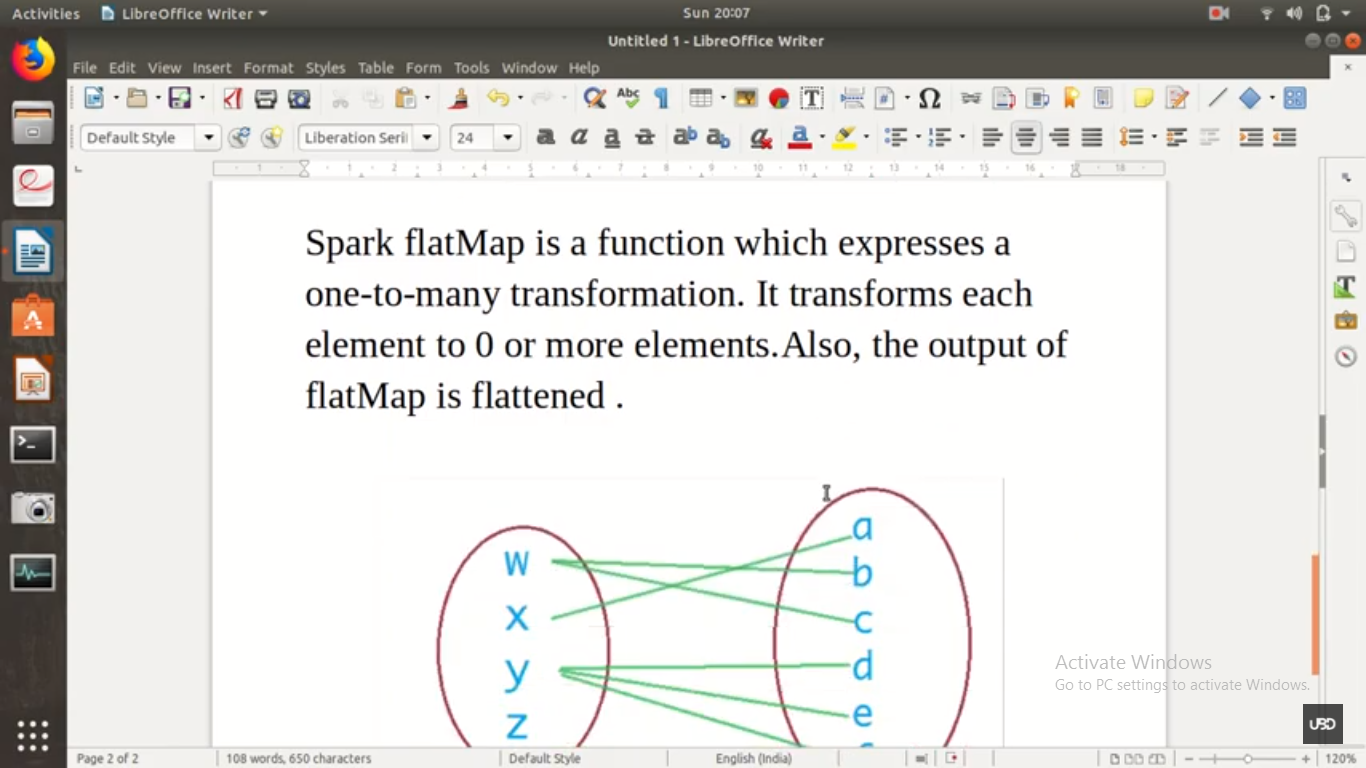
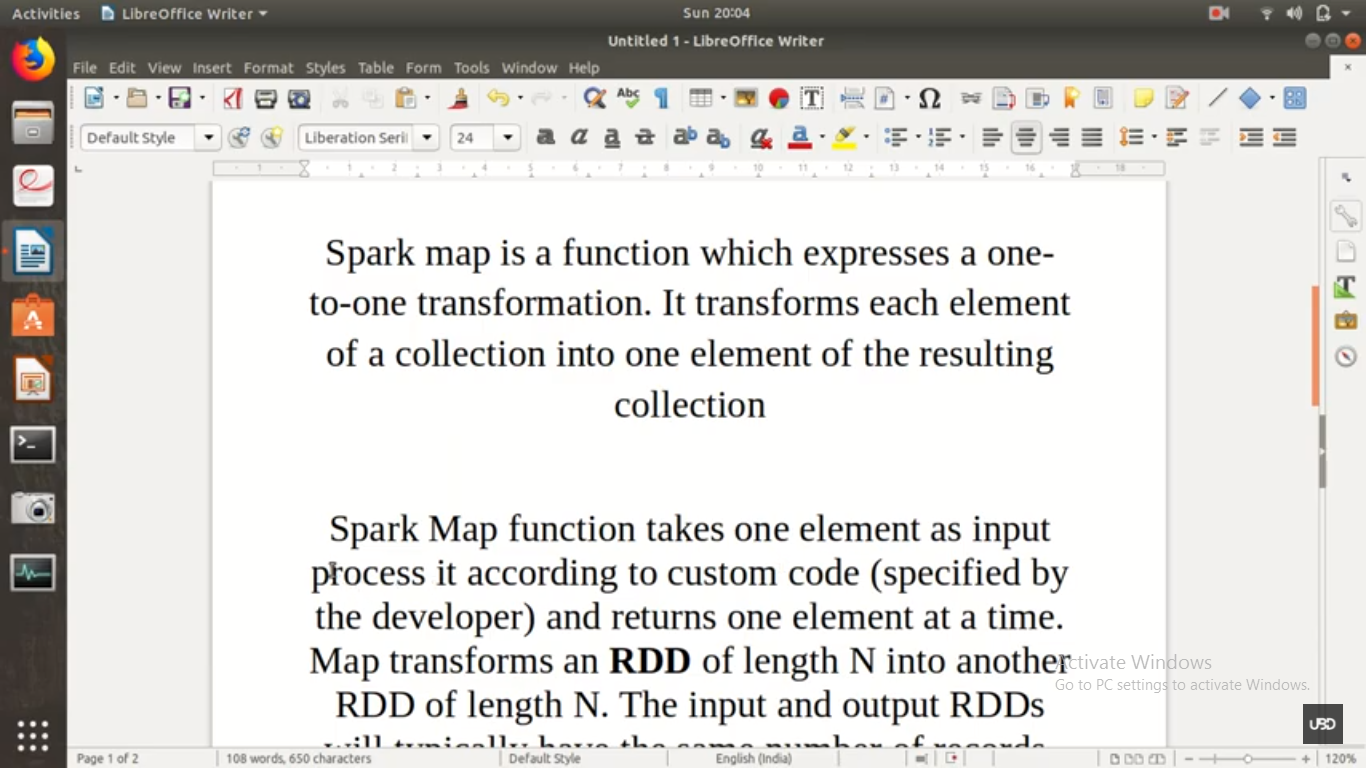
|  |
| --- |
| pyspark.sql.DataFrame.isStreaming()  Returns True if this DataFrame contains one or more sources that continuously return data as it arrives. |

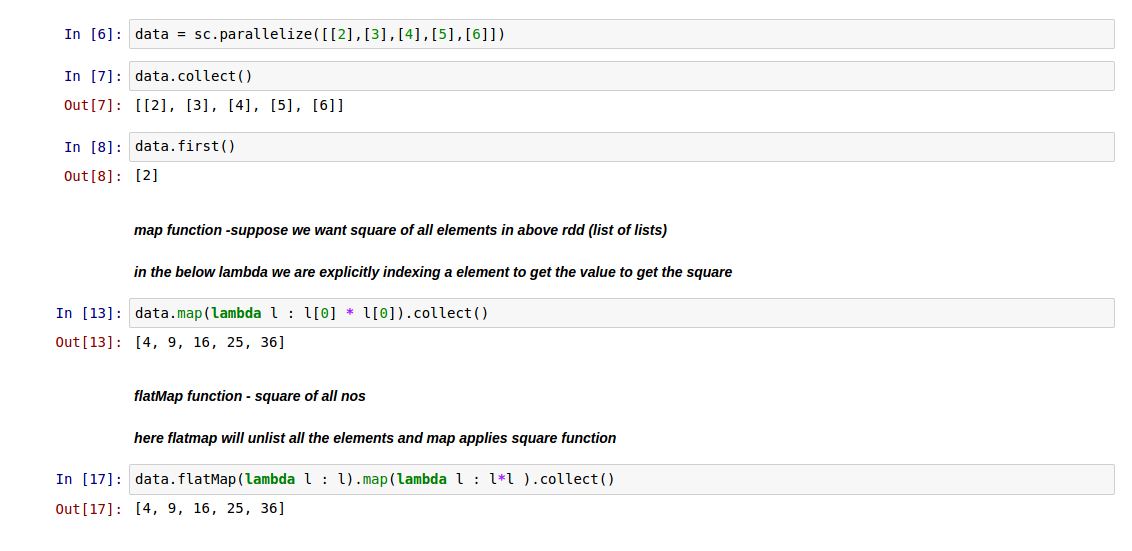
|  |
| --- |
| **pyspark.sql.DataFrame.join(other, on=None, how=None)**   * param other: Right side of the join * param on: a string for the join column name * param how: default inner. Must be one of inner, cross, outer,full, full\_outer, left, left\_outer, right, right\_outer,left\_semi, and left\_anti   Emp Dataset  +------+--------+---------------+-----------+-----------+------+------+  |emp\_id|name |superior\_emp\_id|year\_joined|emp\_dept\_id|gender|salary|  +------+--------+---------------+-----------+-----------+------+------+  |1 |Smith |-1 |2018 |10 |M |3000 |  Dept Dataset  +---------+-------+  |dept\_name|dept\_id|  +---------+-------+  |Finance |10 |  empDF.join(deptDF,empDF.emp\_dept\_id == deptDF.dept\_id,"inner")  +------+--------+---------------+-----------+-----------+------+------+---------+-------+  |emp\_id|name |superior\_emp\_id|year\_joined|emp\_dept\_id|gender|salary|dept\_name|dept\_id|  +------+--------+---------------+-----------+-----------+------+------+---------+-------+  |1 |Smith |-1 |2018 |10 |M |3000 |Finance |10 |  from pyspark.sql.functions import desc  df.join(df2, df.name == df2.name, 'outer').select(df.name, df2.height)  df.join(df2, 'name', 'outer').select('name', 'height').sort(desc("name")).collect()  cond = [df.name == df3.name, df.age == df3.age]  df.join(df3, cond, 'outer').select(df.name, df3.age).collect()  df.join(df2, 'name').select(df.name, df2.height).collect()  df.join(df4, ['name', 'age']).select(df.name, df.age).collect()  **Using SQL Expression**  Since PySpark SQL support native SQL syntax, we can also write join operations after creating temporary tables on DataFrames and use these tables on spark.sql().  empDF.createOrReplaceTempView("EMP")  deptDF.createOrReplaceTempView("DEPT")  joinDF = spark.sql("select \*  from EMP e, DEPT d  where e.emp\_dept\_id == d.dept\_id") \  .show(truncate=False)  **PySpark SQL Join on multiple DataFrames**  When you need to join more than two tables, you either use SQL expression after creating a temporary view on the DataFrame or use the result of join operation to join with another DataFrame like chaining them.  df1.join(df2,df1.id1 == df2.id2,"inner") \  .join(df3,df1.id1 == df3.id3,"inner") |

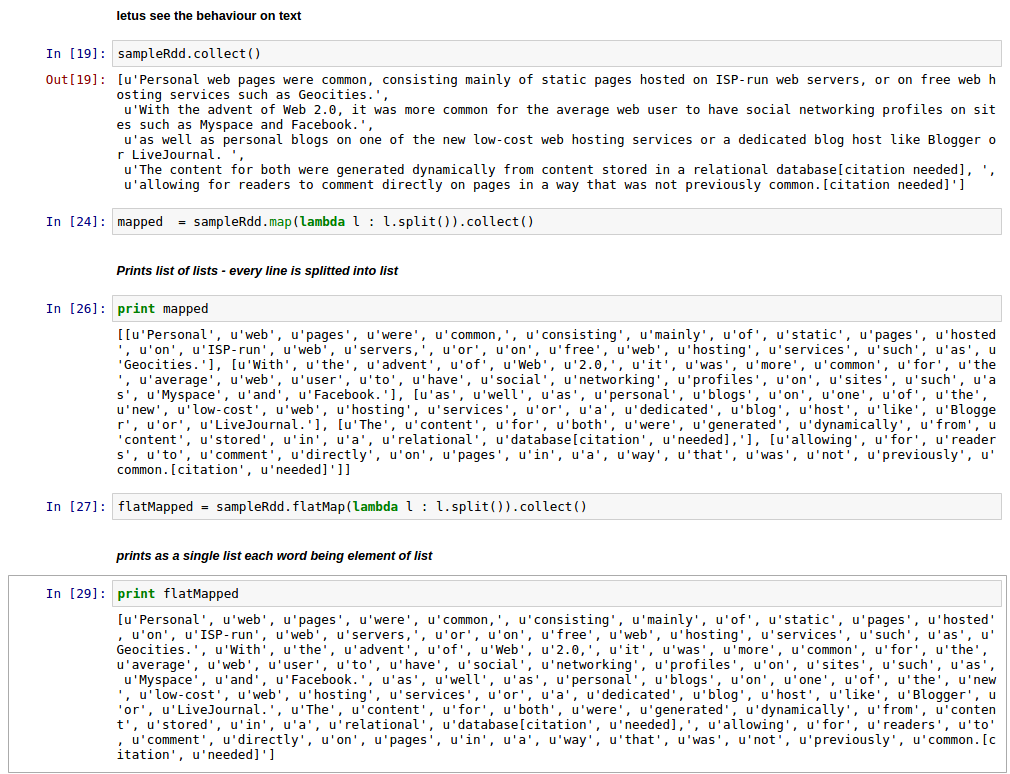
|  |
| --- |
| **pyspark.sql.DataFrame.limit(num)**  Limits the result count to the number specified.  >>>df.limit(1).collect()  [Row(age=2, name='Alice')]  >>>df.limit(0).collect()  [] |

|  |
| --- |
| pyspark.sql.DataFrame.localCheckpoint(eager=True)  Returns a locally checkpointed version of this DataFrame. Checkpointing can be used to truncate the logical plan of this DataFrame, 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. |

|  |
| --- |
| **pyspark.sql.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.  The function should take an iterator of pandas.DataFrames and return another iterator of pandas.DataFrames. All columns are passed together as an iterator of pandas.DataFrames to the function and the returned iterator of pandas.DataFrames are combined as a DataFrame. Each pandas.DataFrame size can be controlled by spark.sql.execution.arrow.maxRecordsPerBatch.  **func*function***  a Python native function that takes an iterator of *pandas.DataFrame*s, and outputs an iterator of *pandas.DataFrame*s.  **schema**[***pyspark.sql.types.DataType***](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.types.DataType.html#pyspark.sql.types.DataType)***or str***  the return type of the *func* in PySpark. The value can be either a [**pyspark.sql.types.DataType**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.types.DataType.html#pyspark.sql.types.DataType) object or a DDL-formatted type string.  from pyspark.sql.functions import pandas\_udf  df = spark.createDataFrame([(1, 21), (2, 30)], ("id", "age"))  def filter\_func(iterator):  for pdf in iterator:  yield pdf[pdf.id == 1]  df.mapInPandas(filter\_func, df.schema).show() |
| **rdd.map(f, preservesPartitioning=False)**  PySpark map (map()) is an RDD transformation that is used to apply the transformation function **(lambda) on every element of RDD/DataFrame and returns a new RDD.**  RDD map() transformation is used to apply any complex operations like adding a column, updating a column, transforming the data e.t.c, the output of map transformations would always have the same number of records as input.  DataFrame doesn’t have map() transformation to use with DataFrame hence you need to DataFrame to RDD first.  https://data-flair.training/blogs/wp-content/uploads/sites/2/2016/12/spark-flatmap-transformation-operation.gif  Map() function with RDD  data = ["Project","Gutenberg’s","Alice’s","Adventures",  "in","Wonderland","Project","Gutenberg’s","Adventures",  "in","Wonderland","Project","Gutenberg’s"]  rdd=spark.sparkContext.parallelize(data)  “““adding a new element with value 1 for each element, the result of the RDD is PairRDDFunctions which contains key-value pairs”””  rdd2=rdd.map(lambda x: (x,1))  for element in rdd2.collect():  print(element)  🡪(‘Projects’,1)  (‘Adventures’,1)  Map() function with dataframe  df = spark.createDataFrame(data=data, schema = columns)  df.show()  +---------+--------+------+------+  |firstname|lastname|gender|salary|  +---------+--------+------+------+  | James| Smith| M| 30|  +---------+--------+------+------+  # Refering columns by index.  rdd3=df.rdd.map(lambda x:  (x[0]+","+x[1],x[2],x[3]\*2)  )  df2=rdd3.toDF(["name","gender","new\_salary"])  +---------------+------+----------+  | name|gender|new\_salary|  +---------------+------+----------+  | James,Smith| M| 60|  +---------------+------+----------+  # Referring Column Names  rdd2=df.rdd.map(lambda x:  (x["firstname"]+","+x["lastname"],x["gender"],x["salary"]\*2)  )  # Referring Column Names  rdd2=df.rdd.map(lambda x:  (x.firstname+","+x.lastname,x.gender,x.salary\*2)  )  You can also create a custom function to perform an operation. Below func1() function executes for every DataFrame row from the lambda function.  # By Calling function  def func1(x):  firstName=x.firstname  lastName=x.lastname  name=firstName+","+lastName  gender=x.gender.lower()  salary=x.salary\*2  return (name,gender,salary)  rdd2=df.rdd.map(lambda x: func1(x)) |
| **rdd.flatMap(f, preservesPartitioning=False)**  PySpark flatMap() is a transformation operation that flattens the RDD/DataFrame (array/map DataFrame columns) after applying the function on every element and returns a new PySpark RDD/DataFrame.  data = ["Project Gutenberg’s",  "Alice’s Adventures in Wonderland",  "Project Gutenberg’s",  "Adventures in Wonderland",  "Project Gutenberg’s"]  rdd=spark.sparkContext.parallelize(data)  for element in rdd.collect():  print(element)  rdd2=rdd.flatMap(lambda x: x.split(" "))  for element in rdd2.collect():  print(element)  Using flatMap() transformation on DataFrame:  Unfortunately, PySpark DataFame doesn’t have flatMap() transformation however, DataFrame has explode() SQL function that is used to flatten the column.  arrayData = [  ('James',['Java','Scala'],{'hair':'black','eye':'brown'}),  ('Michael',['Spark','Java',None],{'hair':'brown','eye':None}),  ('Robert',['CSharp',''],{'hair':'red','eye':''})  ]  df = spark.createDataFrame(data=arrayData, schema = ['name','knownLanguages','properties'])  from pyspark.sql.functions import explode  df2 = df.select(df.name,explode(df.knownLanguages))  df2.show()  +---------+------+  | name| col|  +---------+------+  | James| Java|  | James| Scala|  | Michael| Spark| |







|  |
| --- |
| **pyspark.sql.DataFrame.na**  Returns a DataFrameNaFunctions for handling missing values. |

|  |
| --- |
| **pyspark.sql.DataFrame.orderBy(\*cols, ascending:bool)**  **pyspark.sql.DataFrame.sort(\*cols, \*\*kwargs)**  +-------------+----------+-----+------+---+-----+  |employee\_name|department|state|salary|age|bonus|  +-------------+----------+-----+------+---+-----+  | James| Sales| NY| 90000| 34|10000|  df.sort("department","state")  df.sort(df.age.desc()).collect()  df.sort("age", ascending=False).collect()  df.orderBy(df.age.desc()).collect()  df.orderBy("department","state")  df.orderBy(col("department"),col("state"))  df.orderBy(col("department").asc(),col("state").asc())  df.sort(df.department.asc(),df.state.asc())  df.sort(col("department").asc(),col("state").asc())  from pyspark.sql.functions import \*  df.sort(asc("age")).collect()  df.orderBy(desc("age"), "name").collect()  Besides asc() and desc() functions, PySpark also provides asc\_nulls\_first() and asc\_nulls\_last() and equivalent descending functions.  Below is an example of how to sort DataFrame using raw SQL syntax.  df.createOrReplaceTempView("EMP")  spark.sql("select employee\_name,department  from EMP  ORDER BY department asc").show(truncate=False) |

|  |
| --- |
| pyspark.sql.DataFrame.persist (storageLevel=StorageLevel(True, True, False, True, 1))  Sets the storage level to persist the contents of the DataFrame across operations after the first time it is computed. This can only be used to assign a new storage level if the DataFrame does not have a storage level set yet. If no storage level is specified defaults to (MEMORY\_AND\_DISK\_DESER) |

|  |
| --- |
| pyspark.sql.DataFrame.printSchema()  df.printSchema() |

|  |
| --- |
| pyspark.sql.DataFrame.randomSplit(weights, seed=None)  Randomly splits this DataFrame with the provided weights.  **weights*list***  list of doubles as weights with which to split the [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame). Weights will be normalized if they don’t sum up to 1.0.  **seed*int, optional***  The seed for sampling.  >>>splits = df4.randomSplit([1.0, 2.0], 24)  >>>splits[0].count()  2 |

|  |
| --- |
| pyspark.sql.DataFrame.rdd  Returns the content as an pyspark.RDD of Row. |

|  |
| --- |
| pyspark.sql.DataFrame.registerTempTable(name)  Registers this DataFrame as a temporary table using the given name.  The lifetime of this temporary table is tied to the SparkSession that was used to create this DataFrame.  >>>df.registerTempTable("people")  >>>df2 = spark.sql("select \* from people")  >>>sorted(df.collect()) == sorted(df2.collect())  True  >>>spark.catalog.dropTempView("people") |

|  |
| --- |
| pyspark.sql.DataFrame.repartition(numPartitions, \*cols)  Returns a new DataFrame partitioned by the given partitioning expressions. The resulting DataFrame is hash partitioned.  >>>df.repartition(10).rdd.getNumPartitions()  10  >>>data = df.union(df).repartition("age")  >>>data.show()  data = data.repartition(7, "age")  data.show()  data.rdd.getNumPartitions()  data = data.repartition(3, "name", "age") |

|  |
| --- |
| pyspark.sql.DataFrame.repartitionByRange(numPartitions, \*cols)  Returns a new DataFrame partitioned by the given partitioning expressions. The resulting DataFrame is range partitioned.  At least one partition-by expression must be specified. When no explicit sort order is specified, “ascending nulls first” is assumed.  >>>df.repartitionByRange(2, "age").rdd.getNumPartitions()  2  >>>df.show()  >>>df.repartitionByRange(1, "age").rdd.getNumPartitions()  1  >>>data = df.repartitionByRange("age")  >>>df.show() |

|  |
| --- |
| pyspark.sql.DataFrame.replace(to\_replace, value=<no value>, subset=None)  Returns a new DataFrame replacing a value with another value. DataFrame.replace() and DataFrameNaFunctions.replace() are aliases of each other. Values to\_replace and value must have the same type and can only be numerics, booleans, or strings. Value can have None. When replacing, the new value will be cast to the type of the existing column. For numeric replacements all values to be replaced should have unique floating point representation. In case of conflicts (for example with {42: -1, 42.0: 1}) and arbitrary replacement will be used.  df4.na.replace(10, 20).show()  df4.na.replace('Alice', None).show()  df4.na.replace({'Alice': None}).show()  df4.na.replace(['Alice', 'Bob'], ['A', 'B'], 'name').show() |

|  |
| --- |
| pyspark.sql.DataFrame.rollup(\*cols)  Create a multi-dimensional rollup for the current DataFrame using the specified columns, so we can run aggregation on them.  df.rollup("name", df.age).count().orderBy("name", "age").show() |

|  |
| --- |
| pyspark.sql.DataFrame.sameSemantics(other)  … |

|  |
| --- |
| pyspark.sql.DataFrame.sample(withReplacement=None, fraction=None, seed=None)  Returns a sampled subset of this DataFrame.  fraction is required and, withReplacement and seed are optional. |

|  |
| --- |
| pyspark.sql.DataFrame.sampleBy(col, fractions, seed=None)  Returns a stratified sample without replacement based on the fraction given on each stratum.  from pyspark.sql.functions import col  dataset = sqlContext.range(0, 100).select((col("id") % 3).alias("key"))  sampled = dataset.sampleBy("key", fractions={0: 0.1, 1: 0.2}, seed=0)  sampled.groupBy("key").count().orderBy("key").show() |

|  |
| --- |
| pyspark.sql.DataFrame.schema  Returns the schema of this DataFrame as a pyspark.sql.types.StructType.  >>>df.schema  StructType(List(StructField(age,IntegerType,true),StructField(name,StringType,true))) |

|  |
| --- |
| **pyspark.sql.DataFrame.select(cols)**  Projects a set of expressions and returns a new DataFrame. Since DataFrame is immutable, this creates a new DataFrame with selected columns. show() function is used to show the Dataframe contents.  df.select("firstname","lastname").show()  df.select(df.firstname,df.lastname).show()  df.select(df["firstname"],df["lastname"]).show()  df.select(df.name, (df.age + 10).alias('age')).collect()  # Select All columns from List  df.select(\*columns).show()  # Select All columns  df.select([col for col in df.columns]).show()  df.select("\*").show()  #Selects first 3 columns and top 3 rows  df.select(df.columns[:3]).show(3)  #Selects columns 2 to 4 and top 3 rows  df.select(df.columns[2:4]).show(3)  #By using col() function  from pyspark.sql.functions import col  df.select(col("firstname"),col("lastname")).show()  #Select columns by regular expression  df.select(df.colRegex("`^.\*name\*`")).show()  # If there is nested structure  df2.select("name").show(truncate=False)  +----------------------+  |name |  +----------------------+  |[James,, Smith] |  df2.select("name.firstname","name.lastname")  +---------+--------+  |firstname|lastname|  +---------+--------+  |James |Smith |  df2.select("name.\*")  +---------+----------+--------+  |firstname|middlename|lastname|  +---------+----------+--------+  |James |null |Smith |  **Complete code example:**  import pyspark  from pyspark.sql import SparkSession  spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()  data = [("James","Smith","USA","CA"),  ("Michael","Rose","USA","NY"),  ("Robert","Williams","USA","CA"),  ("Maria","Jones","USA","FL")  ]  columns = ["firstname","lastname","country","state"]  df = spark.createDataFrame(data = data, schema = columns)  df.show(truncate=False)  df.select("firstname").show()  df.select("firstname","lastname").show()  #Using Dataframe object name  df.select(df.firstname,df.lastname).show()  # Using col function  from pyspark.sql.functions import col  df.select(col("firstname"),col("lastname")).show() |

|  |
| --- |
| pyspark.sql.DataFrame.selectExpr(\*expr)  Projects a set of SQL expressions and returns a new DataFrame.  This is a variant of select() that accepts SQL expressions.  df.selectExpr("age \* 2", "abs(age)").collect() |

|  |
| --- |
| pyspark.sql.DataFrame.show(n=20, truncate=True, vertical=False)  Prints the first n rows to the console. |

|  |
| --- |
| pyspark.sql.DataFrame.sortWithinPartitions(cols, ascending)  Returns a new DataFrame with each partition sorted by the specified column(s).  df.sortWithinPartitions("age", ascending=False).show() |

|  |
| --- |
| pyspark.sql.DataFrame.stat  Returns a DataFrameStatFunctions for statistic functions. |

|  |
| --- |
| pyspark.sql.DataFrame.storageLevel  Get the DataFrame’s current storage level.  >>>df.storageLevel  StorageLevel(False, False, False, False, 1) |

|  |
| --- |
| pyspark.sql.DataFrame.subtract(other)  Return a new DataFrame containing rows in this DataFrame but not in another DataFrame.  This is equivalent to EXCEPT DISTINCT in SQL. |

|  |
| --- |
| pyspark.sql.DataFrame.summary(\*statistics)  Computes specified statistics for numeric and string columns. |

|  |
| --- |
| pyspark.sql.DataFrame.tail(n)  Returns the last num rows as a list of Row. |

|  |
| --- |
| pyspark.sql.DataFrame.take(n)  Returns the first num rows as a list of Row. |

|  |
| --- |
| pyspark.sql.DataFrame.toDF(cols)  Returns a new DataFrame that with new specified column names  >>>df.toDF('f1', 'f2').collect()  [Row(f1=2, f2='Alice'), Row(f1=5, f2='Bob')] |

|  |
| --- |
| pyspark.sql.DataFrame.toJSON(use\_unicode=True)  Converts a DataFrame into a RDD of string.  Each row is turned into a JSON document as one element in the returned RDD.  >>>df.toJSON().first()  '{"age":2,"name":"Alice"}' |

|  |
| --- |
| pyspark.sql.DataFrame.toLocalIterator(prefetchPartitions=False) |

|  |
| --- |
| pyspark.sql.DataFrame.toPandas()  Returns the contents of this DataFrame as Pandas pandas.DataFrame.  This is only available if Pandas is installed and available.  This method should only be used if the resulting Pandas’s DataFrame is expected to be small, as all the data is loaded into the driver’s memory.  df.toPandas() |

|  |
| --- |
| pyspark.sql.DataFrame.transform(func)  Returns a new DataFrame. Concise syntax for chaining custom transformations.  **func*function***  a function that takes and returns a [**DataFrame**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html#pyspark.sql.DataFrame).  from pyspark.sql.functions import col  df = spark.createDataFrame([(1, 1.0), (2, 2.0)], ["int", "float"])  def cast\_all\_to\_int(input\_df):  return input\_df.select([col(col\_name).cast("int") for col\_name in input\_df.columns])  def sort\_columns\_asc(input\_df):  return input\_df.select(\*sorted(input\_df.columns))  df.transform(cast\_all\_to\_int).transform(sort\_columns\_asc).show() |

|  |
| --- |
| **pyspark.sql.DataFrame.union(other)**  **pyspark.sql.DataFrame.unionAll(other)**  PySpark union() and unionAll() transformations are used to merge two or more DataFrame’s of the same schema or structure.  If schemas are not the same it returns an error.  unionDF = df.union(df2)  unionDF.show(truncate=False)  # DataFrame unionAll() method is deprecated since PySpark “2.0.0” version and  # recommends using the union() method.  unionAllDF = df.unionAll(df2)  unionAllDF.show(truncate=False)  Merge without Duplicates:  disDF = df.union(df2).distinct()  disDF.show(truncate=False) |

|  |
| --- |
| **pyspark.sql.DataFrame.unionByName(other, allowMissingColumns=False)**  Returns a new DataFrame containing union of rows in this and another DataFrame.  The difference between this function and union() is that this function resolves columns by name (not by position).  df1 = spark.createDataFrame(  [[1, 2, 3]],  ["col0", "col1", "col2"]  )  df2 = spark.createDataFrame(  [[4, 5, 6]],  ["col1", "col2", "col0"]  )  df1.unionByName(df2).show()  +----+----+----+  |col0|col1|col2|  +----+----+----+  | 1| 2| 3|  | 6| 4| 5| |

|  |
| --- |
| pyspark.sql.DataFrame.unpersist(blocking=False)  Marks the DataFrame as non-persistent, and remove all blocks for it from memory and disk. |

|  |
| --- |
| **pyspark.sql.DataFrame.where(condition)**  PySpark filter() function is used to filter the rows from RDD/DataFrame based on the given condition or SQL expression, you can also use where() clause instead of the filter() if you are coming from an SQL background, both these functions operate exactly the same.  Current dataframe:  +----------------------+------------------+-----+------+  |name |languages |state|gender|  +----------------------+------------------+-----+------+  |[James, , Smith] |[Java, Scala, C++]|OH |M |  |[Anna, Rose, ] |[Spark, Java, C++]|NY |F |  DataFrame filter() with Column Condition  # Using equals condition  df.filter(df.state == "OH").show(truncate=False)  # not equals condition  df.filter(df.state != "OH").show(truncate=False)  df.filter(~(df.state == "OH")).show(truncate=False)  #Using SQL col() function  from pyspark.sql.functions import col  df.filter(col("state") == "OH") \  .show(truncate=False)  #Using SQL Expression  df.filter("gender == 'M'").show()  #For not equal  df.filter("gender != 'M'").show()  df.filter("gender <> 'M'").show()  # Filter multiple condition  df.filter( (df.state == "OH") & (df.gender == "M") )  Filter Based on List Values  li=["OH","CA","DE"]  #Filter IS IN List values  df.filter(df.state.isin(li)).show()  # Filter NOT IS IN List values  # These show all records with NY (NY is not part of the list)  df.filter(~df.state.isin(li)).show()  df.filter(df.state.isin(li)==False).show()  Filter Based on Starts With, Ends With, Contains  # Using startswith  df.filter(df.state.startswith("N")).show()  #using endswith  df.filter(df.state.endswith("H")).show()  #contains  df.filter(df.state.contains("H")).show()  PySpark Filter like and rlike  df2.filter(df2.name.like("%rose%")).show()  df2.filter(df2.name.rlike("(?i)^\*rose$")).show()  Filter on an Array column: When you want to filter rows from DataFrame based on value present in an array collection column. array\_contains() checks if a value contains in an array if present it returns true otherwise false.  from pyspark.sql.functions import array\_contains  df.filter(array\_contains(df.languages,"Java")).show(truncate=False)  +----------------+------------------+-----+------+  |name |languages |state|gender|  +----------------+------------------+-----+------+  |[James, , Smith]|[Java, Scala, C++]|OH |M |  Filtering on Nested Struct columns  df.filter(df.name.lastname == "Williams").show(truncate=False) |

|  |
| --- |
| **pyspark.sql.DataFrame.withColumn(colName, col(‘’))**  PySpark withColumn() is a transformation function of DataFrame which is used to change the value, convert the datatype of an existing column, create a new column, and many more.  The column expression must be an expression over this DataFrame; attempting to add a column from some other DataFrame will raise an error. |
| Change DataType using PySpark withColumn()  df.withColumn("salary",col("salary").cast("Integer"))  Update The Value of an Existing Column  df.withColumn("salary",col("salary")\*100).show()  df.withColumn('age2', df.age + 2).collect()  Create a Column from an Existing  df.withColumn("CopiedColumn",col("salary")\* -1).show()  Add a New Column using withColumn()   * PySpark lit() function is used to add a constant value to a DataFrame column * can also chain in order to add multiple columns. * Make sure this new column not already present on DataFrame, if it presents it updates the value of that column   df.withColumn("Country", lit("USA")).show()  df.withColumn("Country", lit("USA")) \  .withColumn("anotherColumn",lit("anotherValue")) \  .show()  Rename Column Name  df.withColumnRenamed("gender","sex") \  .show(truncate=False)  Drop Column From PySpark DataFrame  df.drop("salary").show()  Note that all of these functions return the new DataFrame after applying the functions instead of updating DataFrame.  import pyspark  from pyspark.sql import SparkSession  from pyspark.sql.functions import col, lit  from pyspark.sql.types import StructType, StructField, StringType,IntegerType  spark = SparkSession.builder.getOrCreate()  data = [('James','','Smith','1991-04-01','M',3000),  ('Michael','Rose','','2000-05-19','M',4000),  ]  columns = ["firstname","middlename","lastname","dob","gender","salary"]  df = spark.createDataFrame(data=data, schema = columns)  df.printSchema()  df.show(truncate=False)  df2 = df.withColumn("salary",col("salary").cast("Integer"))  df2.printSchema()  df2.show(truncate=False)  df3 = df.withColumn("salary",col("salary")\*100)  df3.printSchema()  df3.show(truncate=False)  df4 = df.withColumn("CopiedColumn",col("salary")\* -1)  df4.printSchema()  df5 = df.withColumn("Country", lit("USA"))  df5.printSchema()  df6 = df.withColumn("Country", lit("USA")) \  .withColumn("anotherColumn",lit("anotherValue"))  df6.printSchema()  df.withColumnRenamed("gender","sex") \  .show(truncate=False)    df4.drop("CopiedColumn") \  .show(truncate=False) |

|  |
| --- |
| **pyspark.sql.DataFrame.withColumnRenamed(existing, new)**  Returns a new DataFrame by renaming an existing column. This is a no-op if schema doesn’t contain the given column name.  >>>df.withColumnRenamed('age', 'age2').collect()  [Row(age2=2, name='Alice'), Row(age2=5, name='Bob')] |

|  |
| --- |
| pyspark.sql.DataFrame.withWatermark(eventTime, delayThreshold) |

|  |
| --- |
| pyspark.sql.DataFrame.write  Interface for saving the content of the non-streaming DataFrame out into external storage. |

|  |
| --- |
| pyspark.sql.DataFrame.writeStream  Interface for saving the content of the streaming DataFrame out into external storage. |

|  |
| --- |
| pyspark.sql.DataFrame.writeTo(table)  Create a write configuration builder for v2 sources.  This builder is used to configure and execute write operations.  For example, to append or create or replace existing tables.  df.writeTo("catalog.db.table").append()  df.writeTo(  "catalog.db.table"  ).partitionedBy("col").createOrReplace() |

|  |
| --- |
| pyspark.sql.DataFrame.to\_pandas\_on\_spark(index\_col=None)  Converts the existing DataFrame into a pandas-on-Spark DataFrame.  If a pandas-on-Spark DataFrame is converted to a Spark DataFrame and then back to pandas-on-Spark, it will lose the index information and the original index will be turned into a normal column.  This is only available if Pandas is installed and available.  df.to\_pandas\_on\_spark()  df.to\_pandas\_on\_spark(index\_col="Col1"): |

|  |
| --- |
| pyspark.sql.DataFrameNaFunctions.drop(how='any', thresh=None, subset=None)  Returns a new DataFrame omitting rows with null values. DataFrame.dropna() and DataFrameNaFunctions.drop() are aliases of each other.  df4.na.drop().show() |

|  |
| --- |
| pyspark.sql.DataFrameNaFunctions.fill(value, subset=None)  Replace null values, alias for na.fill(). DataFrame.fillna() and DataFrameNaFunctions.fill() are aliases of each other.  df4.na.fill(50).show()  df5.na.fill(False).show()  df4.na.fill({'age': 50, 'name': 'unknown'}).show() |

|  |
| --- |
| pyspark.sql.DataFrameNaFunctions.replace(to\_replace, value=<no value>, subset=None) |

|  |
| --- |
| pyspark.sql.DataFrameStatFunctions.approxQuantile(col, probabilities, relativeError) |

|  |
| --- |
| pyspark.sql.DataFrameStatFunctions.corr(col1, col2, method=None) |

|  |
| --- |
| pyspark.sql.DataFrameStatFunctions.cov(col1, col2) |