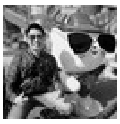
Cheatography

Spark Scala API v2.3 Cheat Sheet

by Ryan (ryan2002) via cheatography.com/108946/cs/21509/



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.
Page 1 of 100.

Data Sources - read

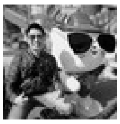
format	"csv", "text", "json", "parquet" (default), "orc", "jdbc"

Data Sources - read (cont)

option	csv	**sep** (default ,): sets a single character as a separator for each field and value. **quote** (default "): sets a single character used for escaping quoted values where the separator can be part of the value. If you woul need to set not null but an empty string. This behaviour is different from com.databricks.spark.csv. **escape** (default \): sets a single character used for escaping quotes inside an already quoted value. charToEscapeQuoteEscaping single character used for escaping the escape for the quote character. The default value is escape character when escape and quo otherwise. comment (default empty string): sets a single character used for skipping lines beginning with this character. By default, **header** (default false): uses the first line as names of columns. **inferSchema** (default false): infers the input schema automatically from data. It requires one extra pass over the data. **mode** (default PERMISSIVE): allows a mode for dealing with corrupt records during parsing. It supports the following case-insensiti *PERMISSIVE* : sets other fields to null when it meets a corrupted record, and puts the malformed string into a field configured by To keep corrupt records, an user can set a string type field named columnNameOfCorruptRecord in an user-defined schema. If a s drops corrupt records during parsing. When a length of parsed CSV tokens is shorter than an expected length of a schema, it sets *DROPMALFORMED* : ignores the whole corrupted records. *FAILFAST* : throws an exception when it meets corrupted records. **nullValue** (default empty string): sets the string representation of a null value. Since 2.0.1, this applies to all supported types includi **nanValue** (default NaN): sets the string representation of a non-number" value. **dateFormat** (default yyyy-MM-dd): sets the string that indicates a date format. Custom date formats follow the formats at java.text.S to date type. **timestampFormat** (default yyyy-MM-dd'T'HH:mm:ss.SSSXXX): sets the string that indicates a timestamp format. Custom date form ext.SimpleDateFormat. This applies to timestamp type. **maxColumns** (default 20480): defines a hard limit of how many columns a record can have. maxCharsPerColumn (default -1): defi characters allowed for any given value being read. By default, it is -1 meaning unlimited length **multiLine** (default false): parse one record, which may span multiple lines. **encoding** (default UTF-8): decodes the CSV files by the given encoding type. **ignoreLeadingWhiteSpace** (default false): a flag indicating whether or not leading whitespaces from values being read should be s **ignoreTrailingWhiteSpace** (default false): a flag indicating whether or not trailing whitespaces from values being read should be sk **positiveInf** (default Inf): sets the string representation of a positive infinity value. **negativeInf** (default -Inf): sets the string representation of a negative infinity value. **columnNameOfCorruptRecord** (default is the value specified in spark.sql.columnNameOfCorruptRecord): allows renaming the ne created by PERMISSIVE mode. This overrides spark.sql.columnNameOfCorruptRecord.

Data Sources - read (cont)

text	**wholetext**(default false)



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Data Sources - read (cont)

json

mode (default PERMISSIVE): allows a mode for dealing with corrupt records during parsing.

PERMISSIVE : sets other fields to null when it meets a corrupted record, and puts the malformed string into a field configured by column corrupt records, an user can set a string type field named columnNameOfCorruptRecord in an user-defined schema. If a schema does not records during parsing. When inferring a schema, it implicitly adds a columnNameOfCorruptRecord field in an output schema.

DROPMALFORMED : ignores the whole corrupted records.

FAILFAST : throws an exception when it meets corrupted records.

columnNameOfCorruptRecord (default is the value specified in spark.sql.columnNameOfCorruptRecord): allows renaming the new field l by PERMISSIVE mode. This overrides spark.sql.columnNameOfCorruptRecord.

dateFormat (default yyyy-MM-dd): sets the string that indicates a date format. Custom date formats follow the formats at java.text.SimpleD type.

timestampFormat (default yyyy-MM-dd'T'HH:mm:ss.SSSXXX): sets the string that indicates a timestamp format. Custom date formats foll mpleDateFormat. This applies to timestamp type.

multiLine (default false): parse one record, which may span multiple lines, per file

primitivesAsString (default false): infers all primitive values as a string type

prefersDecimal (default false): infers all floating-point values as a decimal type. If the values do not fit in decimal, then it infers them as dou

allowComments (default false): ignores Java/C++ style comment in JSON records

allowUnquotedFieldNames (default false): allows unquoted JSON field names

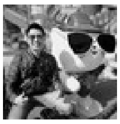
allowSingleQuotes (default true): allows single quotes in addition to double quotes

allowNumericLeadingZeros (default false): allows leading zeros in numbers (e.g. 00012)

allowBackslashEscapingAnyCharacter (default false): allows accepting quoting of all character using backslash quoting mechanism

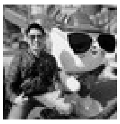
allowUnquotedControlChars (default false): allows JSON Strings to contain unquoted control characters (ASCII characters with value les feed characters) or not.

Data Sources - read (cont)	
parquet	**mergeSchema** (default is the value specified in spark.sql.parquet.mergeSchema): sets whether we should merge schemas collected fr will override spark.sql.parquet.mergeSchema.



Data Sources - read (cont)

	orc

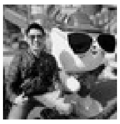


By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Data Sources - read (cont)

jdbc	**url**: The JDBC URL for Spark to connect to. At the minimum, it should contain the host, port, and database name. For MySQL, it may look s l://localhost:3306/sakila. **dbtable**: The name of a database table for Spark to read data from or write data to. **user** **password** **driver**: The class name of the JDBC driver that Spark will instantiate to connect to the previous URL. Consult the JDBC driver documentati MySQL Connector/J driver, the class name is com.mysql.jdbc.Driver.



Data Sources - read (cont)

schema

ϟ can use """...""" define he schema, need use the scala data type.

e.g. `schema("""stockticker STRING, tradedate INT, openprice FLOAT""")`

```
// Mode 1
val movieSchema = StructType(Array(StructField("stockticker", StringType, true),
  StructField("tradedate", IntegerType, true),
  StructField("openprice", FloatType, true)))

// Mode 2: equivalent to mode 1
val movieSchema = """stockticker STRING, tradedate INT, openprice FLOAT"""
```

`DataFrameReader.format(...).option("key", "value").schema(...).load(paths: String*)`

can give multiple paths, can give directory path to read all files in the directory, can use wildcard "*" in the path

To get a DataFrameReader, use `spark.read`

Two ways to define Schema

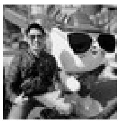
Define a schema programmatically:	`val schema = StructType(Array(StructField("author", StringType, false),`
	`StructField("title", StringType, false),`
	`StructField("pages", IntegerType, false)))`

Two ways to define Schema (cont)

Define a schema with a DDL String	`val schema = “author STRING, title STRING, pages INT”`

Data Source - write	
format	"csv", "text", "json", "parquet" (default), "orc", "jdbc"

Data Source - write (cont)	
mode	"overwrite", "append", "ignore", "error/errorIfExists" (default)



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Data Source - write (cont)

option	csv	**sep** (default ,): sets a single character as a separator for each field and value. **quote** (default "): sets a single character used for escaping quoted values where the separator can be part of the value. If an empty character). **escape** (default \): sets a single character used for escaping quotes inside an already quoted value. charToEscapeQuoteEscaping single character used for escaping the escape for the quote character. The default value is escape character when escape and quo otherwise. **escapeQuotes** (default true): a flag indicating whether values containing quotes should always be enclosed in quotes. Default is to quote character. **quoteAll** (default false): a flag indicating whether all values should always be enclosed in quotes. Default is to only escape values **header** (default false): writes the names of columns as the first line. **nullValue** (default empty string): sets the string representation of a null value. **compression** (default null): compression codec to use when saving to file. This can be one of the known case-insensitive shorten snappy and deflate). **dateFormat** (default yyyy-MM-dd): sets the string that indicates a date format. Custom date formats follow the formats at java.text.S to date type. **timestampFormat** (default yyyy-MM-dd'T'HH:mm:ss.SSSXXX): sets the string that indicates a timestamp format. Custom date form ext.SimpleDateFormat. This applies to timestamp type. **ignoreLeadingWhiteSpace** (default true): a flag indicating whether or not leading whitespaces from values being written should be **ignoreTrailingWhiteSpace** (default true): a flag indicating defines whether or not trailing whitespaces from values being written sho

Data Source - write (cont)

text	**compression** (default null): compression codec to use when saving to file. This can be one of the known case-insensitive shorten names (and deflate).

Data Source - write (cont)	
json	**compression** (default null): compression codec to use when saving to file. This can be one of the known case-insensitive shorten names (and deflate). **dateFormat** (default yyyy-MM-dd): sets the string that indicates a date format. Custom date formats follow the formats at java.text.SimpleD type. **timestampFormat** (default yyyy-MM-dd'T'HH:mm:ss.SSSXXX): sets the string that indicates a timestamp format. Custom date formats follo mpleDateFormat. This applies to timestamp type.

Data Source - write (cont)	
parquet	**compression** (default is the value specified in spark.sql.parquet.compression.codec): compression codec to use when saving to file. Th case-insensitive shorten names(none, snappy, gzip, and lzo). This will override spark.sql.parquet.compression.codec.

Data Source - write (cont)

orc	**compression** (default is the value specified in spark.sql.orc.compression.codec): compression codec to use when saving to file. This can be nsitive shorten names(none, snappy, zlib, and lzo). This will override orc.compress and spark.sql.orc.compression.codec. If orc.compress is c.compression.codec.

Data Source - write (cont)

jdbc	**truncate** (default false): use TRUNCATE TABLE instead of DROP TABLE. In case of failures, users should turn off truncate option to use DROP TABLE again. Also, due to the different behavior of TRUNCATE TAB safe to use this. MySQLDialect, DB2Dialect, MsSqlServerDialect, DerbyDialect, and OracleDialect supports this while PostgresDialect and unknown and unsupported JDBCDirect, the user option truncate is ignored.

Data Source - write (cont)

saveAsTable(tableName: String): Unit

Saves the content of the 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 t
mode is Overwrite, the schema of the DataFrame does not need to be the same as that of the existing table.

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 th
doesn't need to be same as that of the existing table. Unlike insertInto, saveAsTable will use the column names to find the correct

```
scala> Seq((1, 2)).toDF("i", "j").write.mode("overwrite").saveAsTable("t1") scala> Seq((3, 4)).
ode("append").saveAsTable("t1")
scala> sql("select * from t1").show
+---+---+
|   i|   j|
+---+---+
|   1|   2|
|   4|   3|
+---+---+
```

In this method, save mode is used to determine the behavior if the data source table exists in Spark catalog. We will always overw
source (e.g. a table in JDBC data source) if the table doesn't exist in Spark catalog, and will always append to the underlying data
already exists.

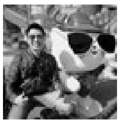
When the DataFrame is created from a non-partitioned HadoopFsRelation with a single input path, and the data source provider c
Hive builtin SerDe (i.e. ORC and Parquet), the table is persisted in a Hive compatible format, which means other systems like Hiv
Otherwise, the table is persisted in a Spark SQL specific format.

Data Source - write (cont)

insertInto(tableName: String): Unit	Inserts the content of the DataFrame to the specified table. It requires that the schema of the DataFrame is the table.
	
	Unlike saveAsTable, insertInto ignores the column names and just uses position-based resolution. For exampl
	
	`scala> Seq((1, 2)).toDF("i", "j").write.mode("overwrite").saveAsTable("t1")`
	`scala> Seq((3, 4)).toDF("j", "i").write.insertInto("t1")`
	`scala> Seq((5, 6)).toDF("a", "b").write.insertInto("t1")`
	`scala> sql("select * from t1").show`
	+---+---+
	| i| j|
	+---+---+
	| 5| 6|
	| 3| 4|
	| 1| 2|
	+---+---+
	
	Because it inserts data to an existing table, format or options will be ignored.

```
DataFrameWriter.format(...).mode(...).option(...).partitionBy(colNames: String ).bucketBy(numBuckets: Int, co
colNames: String).sortBy(colName: String, colNames: String*).save(path: String)
DataFrameWriter.format(...).mode(...).option(...).partitionBy(colNames: String ).bucketBy(numBuckets: Int, co
colNames: String).sortBy(colName: String, colNames: String*).saveAsTable/insertInto(tableName: String)
```

To get DataFrameWriter, use `dataFrame.write`



Data Type

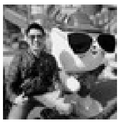
Spark	Scala	Java

Data Type (cont)

ByteType	Byte	byte or Byte

Data Type (cont)

ShortType	Short	short or Short

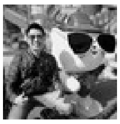


Data Type (cont)

IntegerType	Int	int or Integer

Data Type (cont)

LongType	Long	long or Long



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Data Type (cont)

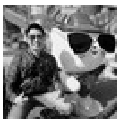
FloatType	Float	float or Float

Data Type (cont)

DoubleType	Double	double or Double

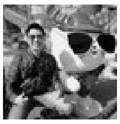
Data Type (cont)

DecimalType	java.math.BigDecimal	java.,math.BigDecimal



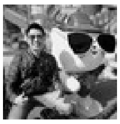
Data Type (cont)

StringType	String	String



Data Type (cont)

BinaryType	Array[Byte]	byte[]



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

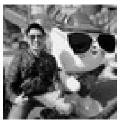
Published 31st January, 2020.
Last updated 5th February, 2020.
Page 31 of 100.

Data Type (cont)

BooleanType	Boolean	boolean or Boolean

Data Type (cont)

DateType	java.sql.Date	java.sql.Date



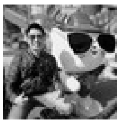
By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Data Type (cont)

TimestampType	java.sql.Timestamp	java.sql.Timestamp

Data Type (cont)

ArrayType	scala.collection.Seq	java.util.List



Data Type (cont)

MapType	scala.collection.Map	java.util.Map

Data Type (cont)

StructType	org.apache.spark.sql.Row	org.apache.spark.sql.Row

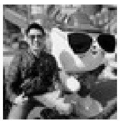
Data Type (cont)
StructField

By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Expressions

Computational expressions	`(((col("someCol") + 5) * 200) - 6) < col("otherCol")`



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Expressions (cont)	
Relational expressions	`expr("(((someCol + 5) * 200) - 6) < otherCol")`

An expression is a set of transformations on one or more values in a record in a DataFrame. Think of it like a function that takes as input one or mo them, and then potentially applies more expressions to create a single value for each record in the dataset. Importantly, this “single value” can actu Map or Array.

Converting to Spark Types - functions

lit(literal: Any): Column	Creates a Column of literal value. The passed in object is returned directly if it is already a Column. If the object is a Scala Symbol, it is converted into a Column is created to represent the literal value.
org.apache.spark.sql.functions	

Change the Column Data Type - Column

cast(to: String): Column	Casts the column to a different data type, using the canonical string representation of the type. The supported types are:**string** **long, float, double, decimal, date, timestamp**.
	`// Casts colA to integer.`
	`df.select(col("colA").cast("int"))`

Change the Column Data Type - Column (cont)

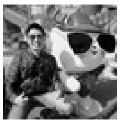
cast(to: DataType): Column	Casts the column to a different data type.
	`// Casts colA to IntegerType.`
	`import org.apache.spark.sql.types.IntegerType`
	`df.select(col("colA").cast(IntegerType))`
	
	`// equivalent to`
	`df.select(df("colA").cast("int"))`

e.g. `df.withColumn("id", col("id").cast("string"))`

org.apache.spark.sql.Dataset - others	
first(): T	Returns the first row. Alias for head().

org.apache.spark.sql.Dataset - others (cont)

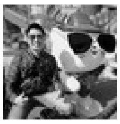
head(): T	Returns the first row.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

org.apache.spark.sql.Dataset - others (cont)

head(n: Int): Array[T]	Returns the first n rows.

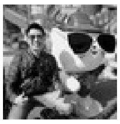


By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.
Page 46 of 100.

org.apache.spark.sql.Dataset - others (cont)

take(n: Int): Array[T]	Returns the first n rows in the Dataset.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

org.apache.spark.sql.Dataset - others (cont)

takeAsList(n: Int): List[T]	Returns the first n rows in the Dataset as a list.

org.apache.spark.sql.Dataset - others (cont)

collect(): Array[T]	Returns an array that contains all rows in this Dataset.

org.apache.spark.sql.Dataset - others (cont)	
collectAsList(): List[T]	Returns a Java list that contains all rows in this Dataset.

org.apache.spark.sql.Dataset - others (cont)

count(): Long	Returns the number of rows in the Dataset.

org.apache.spark.sql.Dataset - others (cont)

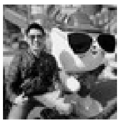
show(): Unit	Displays the top 20 rows of Dataset in a tabular form. Strings more than 20 characters will be truncated, and all cells will be align

org.apache.spark.sql.Dataset - others (cont)

show(numRows: Int): Unit	Displays the Dataset in a tabular form. Strings more than 20 characters will be truncated, and all cells will be alig

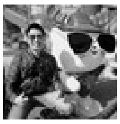
org.apache.spark.sql.Dataset - others (cont)

show(truncate: Boolean): Unit	Displays the top 20 rows of Dataset in a tabular form.



org.apache.spark.sql.Dataset - others (cont)

show(numRows: Int, truncate: Boolean): Unit	Displays the Dataset in a tabular form.

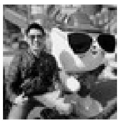


org.apache.spark.sql.Dataset - others (cont)

printSchema(): Unit	Prints the schema to the console in a nice tree format.

org.apache.spark.sql.Dataset - others (cont)

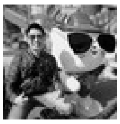
explain(): Unit	Prints the physical plan to the console for debugging purposes.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

org.apache.spark.sql.Dataset - others (cont)	
explain(extended: Boolean): Unit	Prints the plans (logical and physical) to the console for debugging purposes.



org.apache.spark.sql.Dataset - others (cont)

schema: StructType	Returns the schema of this Dataset.

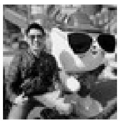
org.apache.spark.sql.Dataset - others (cont)	
columns: Array[String]	Returns all column names as an array.

org.apache.spark.sql.Dataset - others (cont)

describe(cols: String*): DataFrame

Computes basic statistics for numeric and string columns, including count, mean, stddev, min, and

```
ds.describe("age", "height").show()

// output:
// summary age   height
// count   10.0  10.0
// mean    53.3  178.05
// stddev  11.6  15.7
// min     18.0  163.0
// max     92.0  192.0
```



org.apache.spark.sql.Dataset - others (cont)

summary(statistics: String*): DataFrame

Computes specified statistics for numeric and string columns. Available statistics are:
- count - mean - stddev - min - max - arbitrary approximate percentiles specified as a percentag

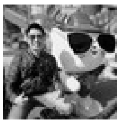
```
ds.summary().show()

// output:
// summary age   height
// count   10.0  10.0
// mean    53.3  178.05
// stddev  11.6  15.7
// min     18.0  163.0
// 25%     24.0  176.0
// 50%     24.0  176.0
// 75%     32.0  180.0
// max     92.0  192.0

ds.summary("count", "min", "25%", "75%", "max").show()

// output:
// summary age   height
// count   10.0  10.0
// mean    53.3  178.05
// min     18.0  163.0
// 25%     24.0  176.0
// 75%     32.0  180.0
// max     92.0  192.0
```

To do a summary for specific columns first select them:

```
ds.select("age", "height").summary().show()
```

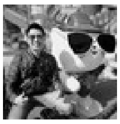
org.apache.spark.sql.Dataset - others (cont)

cache(): Dataset.this.type	Persist this Dataset with the default storage level (MEMORY_AND_DISK).



org.apache.spark.sql.Dataset - others (cont)

persist(): Dataset.this.type	Persist this Dataset with the default storage level (MEMORY_AND_DISK).



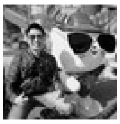
By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

org.apache.spark.sql.Dataset - others (cont)

persist(newLevel: StorageLevel): Dataset.this.type	Persist this Dataset with the given storage level. newLevel One of: MEMORY_ONLY, MEMORY_AND_DISK, MEMORY_ONLY_SER, MEMORY_AND_DISK_SER LY_2, MEMORY_AND_DISK_2, etc.

org.apache.spark.sql.Dataset - others (cont)	
unpersist(): Dataset.this.type	Mark the Dataset as non-persistent, and remove all blocks for it from memory and disk.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

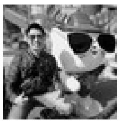
org.apache.spark.sql.Dataset - others (cont)	
unpersist(blocking: Boolean): Dataset.this.type	Mark the Dataset as non-persistent, and remove all blocks for it from memory and blocking: Whether to block until all blocks are deleted.

org.apache.spark.sql.Dataset - others (cont)

storageLevel: StorageLevel	Get the Dataset's current storage level, or StorageLevel.NONE if not persisted.

org.apache.spark.sql.Dataset - others (cont)

rdd: RDD[T]	Represents the content of the Dataset as an RDD of T.

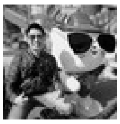


By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

org.apache.spark.sql.Dataset - others (cont)

toDF(): DataFrame	Converts this strongly typed collection of data to generic Dataframe.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

org.apache.spark.sql.Dataset - others (cont)	

toDF(colNames: String*): DataFrame	Converts this strongly typed collection of data to generic DataFrame with columns renamed.
	`val rdd: RDD[(Int, String)] = ...`
	`rdd.toDF()  // this implicit conversion creates a DataFrame with column name`
	`rdd.toDF("id", "name")  // this creates a DataFrame with column name "id" ar`

org.apache.spark.sql.Dataset - others (cont)	
coalesce(numPartitions: Int): Dataset[T]	Returns a new Dataset that has exactly numPartitions partitions, when the fewer partitions are

org.apache.spark.sql.Dataset - others (cont)

repartition(numPartitions: Int): Dataset[T]	Returns a new Dataset that has exactly numPartitions partitions.

org.apache.spark.sql.Dataset - others (cont)	
repartition(numPartitions: Int, partitionExprs: Column*): Dataset[T]	Returns a new Dataset partitioned by the given partitioning expressions into numParti hash partitioned.

org.apache.spark.sql.Dataset - others (cont)

repartition(partitionExprs: Column*): Dataset[T]	Returns a new Dataset partitioned by the given partitioning expressions, using spark.sql.shuffle.p

org.apache.spark.sql - Transformations

select(col: String, cols: String*): DataFrame	Selects a set of columns. `ds.select("colA", "colB")`

org.apache.spark.sql - Transformations (cont)

select(cols: Column*): DataFrame	Selects a set of column based expressions. `ds.select($"colA", $"colB")`

org.apache.spark.sql - Transformations (cont)

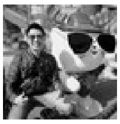
selectExpr(exprs: String*): DataFrame	Selects a set of SQL expressions.
	`// The following are equivalent:`
	`ds.selectExpr("colA", "colB as newName", "abs(colC)")`
	`ds.select(expr("colA"), expr("colB as newName"), expr("abs(colC)"))`
	
	`df.selectExpr("*","(produced_year - (produced_year % 10)) as decade"`

org.apache.spark.sql - Transformations (cont)

where(conditionExpr: String): Dataset[T]	Filters rows using the given SQL expression. To filter a DataFrame, you can also just specify a Boolean column: `df.where("isExpens` `peopleDs.where("age > 15")`

org.apache.spark.sql - Transformations (cont)

where(condition: Column): Dataset[T]	Filters rows using the given condition. `peopleDs.where($"age" > 15)`



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

org.apache.spark.sql - Transformations (cont)

filter(conditionExpr: String): Dataset[T]	Filters rows using the given SQL expression. `peopleDs.filter("age > 15")`

org.apache.spark.sql - Transformations (cont)

filter(condition: Column): Dataset[T]	Filters rows using the given condition.
	// The following are equivalent:
	`peopleDs.filter($"age" > 15)`
	`peopleDs.where($"age" > 15)`

org.apache.spark.sql - Transformations (cont)

filter(func: (T) ⇒ Boolean): Dataset[T]	Returns a new Dataset that only contains elements where func returns true.

org.apache.spark.sql - Transformations (cont)

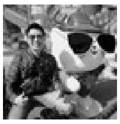
orderBy(sortExprs: Column*): Dataset[T]	Returns a new Dataset sorted by the given expressions. This is an alias of the sort functi `movieTitles.orderBy('title_length.desc, 'produced_year)`

org.apache.spark.sql - Transformations (cont)

orderBy(sortCol: String, sortCols: String*): Dataset[T]	Returns a new Dataset sorted by the given expressions. This is an alias of the

org.apache.spark.sql - Transformations (cont)

sort(sortExprs: Column*): Dataset[T]	Returns a new Dataset sorted by the given expressions. e.g. `ds.sort($"col1", $"col2".desc)`

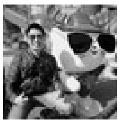


org.apache.spark.sql - Transformations (cont)

sort(sortCol: String, sortCols: String*): Dataset[T]	Returns a new Dataset sorted by the specified column, all in ascending ord
	`// The following 3 are equivalent`
	`ds.sort("sortcol")`
	`ds.sort($"sortcol")`
	`ds.sort($"sortcol".asc)`

org.apache.spark.sql - Transformations (cont)

distinct(): Dataset[T]	Returns a new Dataset that contains only the unique rows from this Dataset. This is an alias for dropDuplicates.



org.apache.spark.sql - Transformations (cont)

dropDuplicates(): Dataset[T]	Returns a new Dataset that contains only the unique rows from this Dataset. This is an alias for distinct.

org.apache.spark.sql - Transformations (cont)	

dropDuplicates(col1: String, cols: String*): Dataset[T]	Returns a new Dataset with duplicate rows removed, considering only the subset of colur
	`// The following are equivalent`
	`movies.select("movie_title").distinct.selectExpr("count(movie_titl`
	`movies.dropDuplicates("movie_title").selectExpr("count(movie_title`

org.apache.spark.sql - Transformations (cont)

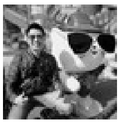
dropDuplicates(colNames: Seq[String]): Dataset[T]	Returns a new Dataset with duplicate rows removed, considering only the subset

org.apache.spark.sql - Transformations (cont)

dropDuplicates(colNames: Array[String]): Dataset[T]	Returns a new Dataset with duplicate rows removed, considering only the subse

org.apache.spark.sql - Transformations (cont)

limit(n: Int): Dataset[T]	Returns a new Dataset by taking the first n rows. The difference between this function and head is that head is an action and re query execution) while limit returns a new Dataset.



org.apache.spark.sql - Transformations (cont)	
withColumn(colName: String, col: Column): DataFrame	Returns a new Dataset by adding a column or replacing the existing column that has the sar However, if the given column name matches one of the existing ones, then that column is re expression. `// adding a new column based on a certain column expression` `movies.withColumn("decade", ('produced_year - 'produced_year % 10))`

org.apache.spark.sql - Transformations (cont)

withColumnRenamed(existingName: String, newName: String): DataFrame	Returns a new Dataset with a column renamed. This is a no-op if schema does Notice that if the provided existingColName doesn’t exist in the schema, Spark silently do nothing. `movies.withColumnRenamed("actor_name", "actor")`

org.apache.spark.sql - Transformations (cont)

drop(colName: String): DataFrame	Returns a new Dataset with columns dropped. This is a no-op if schema doesn't contain column nar `movies.drop("actor_name")`

org.apache.spark.sql - Transformations (cont)

drop(colNames: String*): DataFrame	Returns a new Dataset with columns dropped. This is a no-op if schema doesn't contain column name(s). You can specify one or more column names to drop, but only the ones that exist in the schema will be dropped silently ignored. `movies.drop("actor_name", "me")`

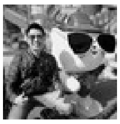
Cheatography

Spark Scala API v2.3 Cheat Sheet

by Ryan (ryan2002) via cheatography.com/108946/cs/21509/

org.apache.spark.sql - Transformations (cont)

drop(col: Column): DataFrame	Returns a new Dataset with a column dropped. This version of drop accepts a Column rather than a name. This is a no- column with an equivalent expression. `movies.drop($"actor_name")`



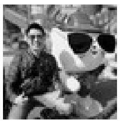
org.apache.spark.sql - Transformations (cont)

union(other: Dataset[T]): Dataset[T]	Returns a new Dataset containing union of rows in this Dataset and another Dataset. This is equivalent to UNION ALL in SQL. To do a SQL-style set union (that does deduplication of elements), use this function fo Notice that the column positions in the schema aren't necessarily matched with the fields in the strongly typed objects in a Data columns by their positions in the schema, not the fields in the strongly typed objects. Use unionByName to resolve columns by

```
val df1 = Seq((1, 2, 3)).toDF("col0", "col1", "col2")
val df2 = Seq((4, 5, 6)).toDF("col1", "col2", "col0")
df1.union(df2).show

// output:
// +----+----+----+
// |col0|col1|col2|
// +----+----+----+
// |   1|   2|   3|
// |   4|   5|   6|
// +----+----+----+
```

org.apache.spark.sql - Transformations (cont)

unionByName(other: Dataset[T]): Dataset[T]	Returns a new Dataset containing union of rows in this Dataset and another Dataset.
	This is different from both UNION ALL and UNION DISTINCT in SQL. To do a SQL-style set union (that does
	use this function followed by a distinct.
	The difference between this function and union is that this function resolves columns by name (not by positio
	`val df1 = Seq((1, 2, 3)).toDF("col0", "col1", "col2")`
	`val df2 = Seq((4, 5, 6)).toDF("col1", "col2", "col0")`
	`df1.unionByName(df2).show`
	
	`// output:`
	`// +----+----+----+`
	`// |col0|col1|col2|`
	`// +----+----+----+`
	`// |   1|   2|   3|`
	`// |   6|   4|   5|`
	`// +----+----+----+`



org.apache.spark.sql - Transformations (cont)

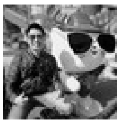
intersect(other: Dataset[T]): Dataset[T]	Returns a new Dataset containing rows only in both this Dataset and another Dataset. This is equivale

Working with Booleans - Column

===(other: Any): Column	Equality test. `df.where(col("InvoiceNo") === 536365)`

Working with Booleans - Column (cont)

equalTo(other: Any): Column	Equality test. `df.where(col("InvoiceNo").equalTo(536365))`

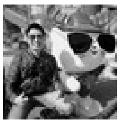


Working with Booleans - Column (cont)

<=>(other: Any): Column	Equality test that is safe for null values.

Working with Booleans - Column (cont)

=!=(other: Any): Column	Inequality test. `df.where(col("InvoiceNo") =!= 536365)`



Working with Booleans - Column (cont)

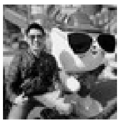
<(other: Any): Column	Less than.

Working with Booleans - Column (cont)

<=(other: Any): Column	Less than or equal to.

Working with Booleans - Column (cont)

>(other: Any): Column	Greater than.



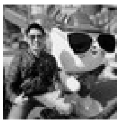
By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Booleans - Column (cont)	
>=(other: Any): Column	Greater than or equal to an expression.

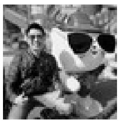
Working with Booleans - Column (cont)

&&(other: Any): Column	Boolean AND.



Working with Booleans - Column (cont)

||(other: Any): Column	Boolean OR.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

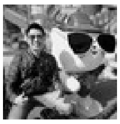
Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Booleans - Column (cont)

isNaN: Column	True if the current expression is NaN.

Working with Booleans - Column (cont)

isNotNull: Column	True if the current expression is NOT null.



Working with Booleans - Column (cont)

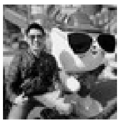
isNull: Column	True if the current expression is null.

Working with Booleans - Column (cont)

isin(list: Any*): Column	A boolean expression that is evaluated to true if the value of this expression is contained by the evaluated values of th
	According to documentation, isin takes a vararg, not a list. List is actually a confusing name here.
	`val items = List("a", "b", "c")`
	`df.filter($"c1".isin(items:_*))`
	or
	`df.filter($"c1".isin("a", "b", "c"))`

Working with Booleans - Column (cont)

like(literal: String): Column	SQL like expression. Returns a boolean column based on a SQL LIKE match. **SQL Wildcards** **%**: Represents zero or more characters, e.g. `bl%` finds bl, black, blue, and blob _: Represents a single character, e.g. `h_t` finds hot, hat, and hit **[]**: Represents any single character within the brackets, e.g. `h[oa]t` finds hot and hat, but not hit **^**: Represents any character not in the brackets, e.g. `h[^oa]t` finds hit, but not hot and hat **-**: Represents a range of characters, e.g. `c[a-b]t` finds cat and cbt



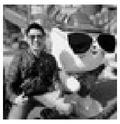
By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Working with Booleans - Column (cont)

rlike(literal: String): Column	SQL RLIKE expression (LIKE with Regex).

Working with Booleans - Column (cont)

startsWith(literal: String): Column	String starts with another string literal. Returns a boolean column based on a string match.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Booleans - Column (cont)

startsWith(other: Column): Column	String starts with.

Working with Booleans - Column (cont)

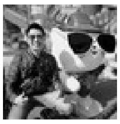
endsWith(literal: String): Column	String ends with another string literal. Returns a boolean column based on a string match.

Working with Booleans - Column (cont)

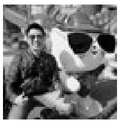
endsWith(other: Column): Column	String ends with. Returns a boolean column based on a string match.

Working with Booleans - Column (cont)

contains(other: Any): Column	Contains the other element. Returns a boolean column based on a string match.
org.apache.spark.sql.Column	

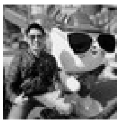


Working with Booleans - functions	
not(e: Column): Column	Inversion of boolean expression, i.e. NOT.

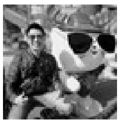


Working with Booleans - functions (cont)

isnan(e: Column): Column	Return true iff the column is NaN.



Working with Booleans - functions (cont)	
isnull(e: Column): Column	Return true iff the column is null.
org.apache.spark.sql.functions	

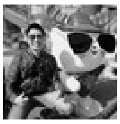


Working with Numbers - Column

+(other: Any): Column	Sum of this expression and another expression.

Working with Numbers - Column (cont)

-(other: Any): Column	Subtraction. Subtract the other expression from this expression.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Numbers - Column (cont)

*(other: Any): Column	Multiplication of this expression and another expression.

Working with Numbers - Column (cont)

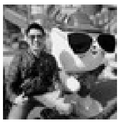
/(other: Any): Column	Division this expression by another expression.

Working with Numbers - Column (cont)

%(other: Any): Column	Modulo (a.k.a. , e.g.`11 mod 4 = 3`

org.apache.spark.sql.Column

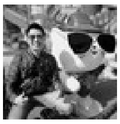
Working with Numbers - functions	
abs(e: Column): Column	Computes the absolute value.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Working with Numbers - functions (cont)

round(e: Column): Column	Returns the value of the column e rounded to 0 decimal places with HALF_UP round mode.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Working with Numbers - functions (cont)

round(e: Column, scale: Int): Column	Round the value of e to scale decimal places with HALF_UP round mode if scale is greater than or equal to 0 less than 0.

Working with Numbers - functions (cont)

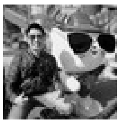
bround(e: Column): Column	Returns the value of the column e rounded to 0 decimal places with HALF_EVEN round mode. HALF_EVEN round towards the "nearest neighbor" unless both neighbors are equidistant, in which case, round to

Working with Numbers - functions (cont)

bround(e: Column, scale: Int): Column	Round the value of e to scale decimal places with HALF_EVEN round mode if scale is greater than or equal t scale is less than 0.

Working with Numbers - functions (cont)

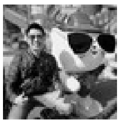
pow(l: Double, rightName: String): Column	Returns the value of the first argument raised to the power of the second argument.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Numbers - functions (cont)	
pow(l: Double, r: Column): Column	Returns the value of the first argument raised to the power of the second argument.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

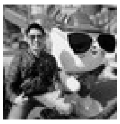
Working with Numbers - functions (cont)

pow(leftName: String, r: Double): Column	Returns the value of the first argument raised to the power of the second argument.

Working with Numbers - functions (cont)	
pow(l: Column, r: Double): Column	Returns the value of the first argument raised to the power of the second argument.

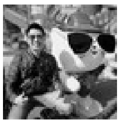
Working with Numbers - functions (cont)

pow(leftName: String, rightName: String): Column	Returns the value of the first argument raised to the power of the second argu



Working with Numbers - functions (cont)

pow(leftName: String, r: Column): Column	Returns the value of the first argument raised to the power of the second argument.

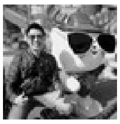


By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Numbers - functions (cont)

pow(l: Column, rightName: String): Column	Returns the value of the first argument raised to the power of the second argument



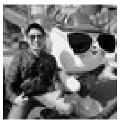
Working with Numbers - functions (cont)

pow(l: Column, r: Column): Column	Returns the value of the first argument raised to the power of the second argument.

org.apache.spark.sql.functions

Working with Strings - Column

contains(other: Any): Column	Contains the other element. Returns a boolean column based on a string match.



Working with Strings - Column (cont)

startsWith(literal: String): Column	String starts with another string literal. Returns a boolean column based on a string match.

Working with Strings - Column (cont)

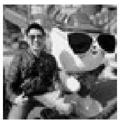
startsWith(other: Column): Column	String starts with. Returns a boolean column based on a string match.

Working with Strings - Column (cont)

endsWith(literal: String): Column	String ends with another string literal. Returns a boolean column based on a string match.

Working with Strings - Column (cont)

endsWith(other: Column): Column	String ends with. Returns a boolean column based on a string match.



Working with Strings - Column (cont)

substr(startPos: Int, len: Int): Column	An expression that returns a substring. startPos begins with **1**. *In scala, String has also a function substring(int beginIndex, int endIndex), here the beginIndex star*

Working with Strings - Column (cont)	
substr(startPos: Column, len: Column): Column	An expression that returns a substring.

Working with Strings - Column (cont)

like(literal: String): Column	SQL like expression. Returns a boolean column based on a SQL LIKE match. **SQL Wildcards** **%**: Represents zero or more characters, e.g. `bl%` finds bl, black, blue, and blob _: Represents a single character, e.g. `h_t` finds hot, hat, and hit **[]**: Represents any single character within the brackets, e.g. `h[oa]t` finds hot and hat, but not hit **^**: Represents any character not in the brackets, e.g. `h[^oa]t` finds hit, but not hot and hat **-**: Represents a range of characters, e.g. `c[a-b]t` finds cat and cbt

Working with Strings - Column (cont)

rlike(literal: String): Column	SQL RLIKE expression (LIKE with Regex).

org.apache.spark.sql.Column

Working with Strings - functions

initcap(e: Column): Column	Returns a new string column by converting the first letter of each word to uppercase.

Working with Strings - functions (cont)

lower(e: Column): Column	Converts a string column to lower case.

Working with Strings - functions (cont)

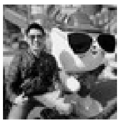
upper(e: Column): Column	Converts a string column to upper case.

Working with Strings - functions (cont)

trim(e: Column): Column	Trim the spaces from both ends for the specified string column.

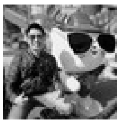
Working with Strings - functions (cont)

trim(e: Column, trimString: String): Column	Trim the specified character from both ends for the specified string column.



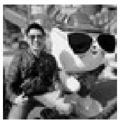
Working with Strings - functions (cont)

ltrim(e: Column): Column	Trim the spaces from left end for the specified string value.



Working with Strings - functions (cont)

ltrim(e: Column, trimString: String): Column	Trim the specified character string from left end for the specified string column.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.
Page 159 of 100.

Working with Strings - functions (cont)

rtrim(e: Column): Column	Trim the spaces from right end for the specified string value.

Working with Strings - functions (cont)

rtrim(e: Column, trimString: String): Column	Trim the specified character string from right end for the specified string column.

Working with Strings - functions (cont)

lpad(str: Column, len: Int, pad: String): Column	Left-pad the string column with pad to a length of len. If the string column is longer than len, the retu characters.

Working with Strings - functions (cont)

rpad(str: Column, len: Int, pad: String): Column	Right-pad the string column with pad to a length of len. If the string column is longer than len, the re characters.

Working with Strings - functions (cont)

substring(str: Column, pos: Int, len: Int): Column	Substring starts at pos and is of length len when str is String type or returns the slice of byte array that st length len when str is Binary type. ***Note:*** *The position is not zero based, but 1 based index.**1**.* *In scala, String has also a function substring(int beginIndex, int endIndex), here the beginIndex starts fro*

Working with Strings - functions (cont)

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

Working with Strings - functions (cont)	

regexp_extract(e: Column, exp: String, groupIdx: Int): Column	Extract a specific group matched by a Java regex, from the specified string column. If the regex di group did not match, an empty string is returned.
	`val rhymeDF = Seq(("A fox saw a crow sitting on a tree singing \"Caw! Ca yme")`
	`rhymeDF.select(regexp_extract('rhyme, "[a-z]*o[xw]",0).as("substring")).`
	There could be multiple matches of the pattern in a string; therefore, the group index (starts with 0 one.

Working with Strings - functions (cont)

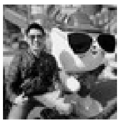
regexp_replace(e: Column, pattern: Column, replacement: Column): Column	Replace all substrings of the specified string value that m

Working with Strings - functions (cont)

regexp_replace(e: Column, pattern: String, replacement: String): Column	Replace all substrings of the specified string value that match regexp with rep.
	`val rhymeDF = Seq(("A fox saw a crow sitting on a tree singi Caw!\"")).toDF("rhyme")`
	`rhymeDF.select(regexp_replace('rhyme, "fox|crow", "animal").a (false)`
	`rhymeDF.select(regexp_replace('rhyme, "[a-z]*o[xw]", "animal how(false)`

Working with Strings - functions (cont)

repeat(str: Column, n: Int): Column	Repeats a string column n times, and returns it as a new string column.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

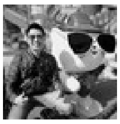
Published 31st January, 2020.
Last updated 5th February, 2020.
Page 169 of 100.

Working with Strings - functions (cont)

reverse(str: Column): Column	Reverses the string column and returns it as a new string column.

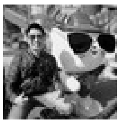
Working with Strings - functions (cont)

split(str: Column, pattern: String): Column	Splits str around pattern (pattern is a regular expression).



Working with Strings - functions (cont)

length(e: Column): Column	Computes the character length of a given string or number of bytes of a binary string. The length of character strings incluc of binary strings includes binary zeros.



Working with Strings - functions (cont)

translate(src: Column, matchingString: String, replaceString: String): Column	Translate any character in the src by a character in replaceString. The characters in replaceString corre matchingString. The translate will happen when any character in the string matches the character in the

Working with Strings - functions (cont)

concat(exprs: Column*): Column	Concatenates multiple input columns together into a single column. If all inputs are binary, concat returns an outp returns as string.

Working with Strings - functions (cont)	
concat_ws(sep: String, exprs: Column*): Column	Concatenates multiple input string columns together into a single string column, using th

Working with Strings - functions (cont)

instr(str: Column, substring: String): Column	Locate the position of the first occurrence of substr column in the given string. Returns null if eithe ***Note:*** *The position is not zero based, but 1 based index. Returns 0 if substr could not be found in*

Working with Strings - functions (cont)

locate(substr: String, str: Column, pos: Int): Column	Locate the position of the first occurrence of substr in a string column, after position ***Note:*** *The position is not zero based, but 1 based index. Returns 0 if substr could n*

Working with Strings - functions (cont)	
locate(substr: String, str: Column): Column	Locate the position of the first occurrence of substr in a string column, after position pos. ***Note:*** *The position is not zero based, but 1 based index. Returns 0 if substr could not be fo*
org.apache.spark.sql.functions	

Working with Date/Time - functions	
current_date(): Column	Returns the current date as a date column.

Working with Date/Time - functions (cont)

current_timestamp(): Column	Returns the current timestamp as a timestamp column.

Working with Date/Time - functions (cont)

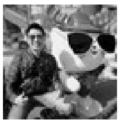
date_add(start: Column, days: Int): Column	Returns the date that is days days after start

Working with Date/Time - functions (cont)

date_sub(start: Column, days: Int): Column	Returns the date that is days days before start

Working with Date/Time - functions (cont)

datediff(end: Column, start: Column): Column	Returns the number of days from start to end.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.
Page 183 of 100.

Working with Date/Time - functions (cont)

add_months(startDate: Column, numMonths: Int): Column	Returns the date that is numMonths after startDate

Working with Date/Time - functions (cont)

months_between(date1: Column, date2: Column): Column	Returns number of months between dates date1 and da

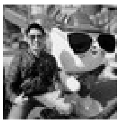
Working with Date/Time - functions (cont)

year(e: Column): Column	Extracts the year as an integer from a given date/timestamp/string.

Working with Date/Time - functions (cont)	
quarter(e: Column): Column	Extracts the quarter as an integer from a given date/timestamp/string.

Working with Date/Time - functions (cont)	
month(e: Column): Column	Extracts the month as an integer from a given date/timestamp/string.

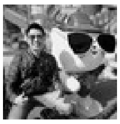
Working with Date/Time - functions (cont)	
weekofyear(e: Column): Column	Extracts the week number as an integer from a given date/timestamp/string.



Working with Date/Time - functions (cont)	
dayofyear(e: Column): Column	Extracts the day of the year as an integer from a given date/timestamp/string.

Working with Date/Time - functions (cont)

dayofmonth(e: Column): Column	Extracts the day of the month as an integer from a given date/timestamp/string.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

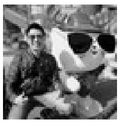
Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Date/Time - functions (cont)

dayofweek(e: Column): Column	Extracts the day of the week as an integer from a given date/timestamp/string.

Working with Date/Time - functions (cont)

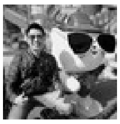
hour(e: Column): Column	Extracts the hours as an integer from a given date/timestamp/string.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

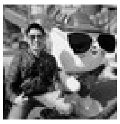
Published 31st January, 2020.
Last updated 5th February, 2020.
Page 193 of 100.

Working with Date/Time - functions (cont)	
minute(e: Column): Column	Extracts the minutes as an integer from a given date/timestamp/string.



Working with Date/Time - functions (cont)

second(e: Column): Column	Extracts the seconds as an integer from a given date/timestamp/string.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.
Page 195 of 100.

Working with Date/Time - functions (cont)

to_date(e: Column): Column	Converts the column into DateType by casting rules to DateType.

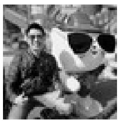
Working with Date/Time - functions (cont)

to_date(e: Column, fmt: String): Column	Converts the column into a DateType with a specified format (see [http://docs.oracle.com/javase/tutorial/i18n return null if fail. The format here is the format, which is used by Date to be saved in DF. DF.show() will display the date in default format yyyy-MM-dd.

Working with Date/Time - functions (cont)	
to_timestamp(s: Column): Column	Convert time string to a Unix timestamp (in seconds) by casting rules to TimestampType.

Working with Date/Time - functions (cont)

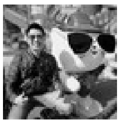
to_timestamp(s: Column, fmt: String): Column	Convert time string to a Unix timestamp (in seconds) with a specified format (see [http://docs.oracle.com/javase/ teFormat.html]) to Unix timestamp (in seconds), return null if fail. The format here is the format, which is used by timestamp to be saved in DF. DF.show() will display the timestamp in default format yyyy-MM-dd HH:mm:ss



Working with Date/Time - functions (cont)

date_format(dateExpr: Column, format: String): Column	Converts a date/timestamp/string to a value of string in the format specified by the date format given by the seco dd.MM.yyyy would return a string like 18.03.1993. All pattern letters of java.text.SimpleDateFormat can be used.

Working with Date/Time - functions (cont)	
unix_timestamp(): Column	Returns the current Unix timestamp (in seconds).



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

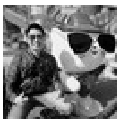
Working with Date/Time - functions (cont)

unix_timestamp(s: Column): Column	Converts time string in format yyyy-MM-dd HH:mm:ss to Unix timestamp (in seconds), using the default timezo Returns null if fails.

Working with Date/Time - functions (cont)

unix_timestamp(s: Column, p: String): Column	Converts time string with given pattern to Unix timestamp (in seconds). Returns null

Working with Date/Time - functions (cont)	
from_unixtime(ut: Column, f: String): Column	Converts the number of seconds from unix epoch (1970-01-01 00:00:00 UTC) to a string representing the time current system time zone in the given format.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

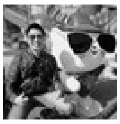
Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Date/Time - functions (cont)

from_unixtime(ut: Column): Column	Converts the number of seconds from unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestam system time zone in the given format.

Working with Date/Time - functions (cont)

last_day(e: Column): Column	Given a date column, returns the last day of the month which the given date belongs to. For example, input "2015-07-27" 31 is the last day of the month in July 2015.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Date/Time - functions (cont)

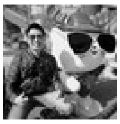
next_day(date: Column, dayOfWeek: String): Column	Given a date column, returns the first date which is later than the value of the date column th week. For example, next_day('2015-07-27', "Sunday") returns 2015-08-02 because that is the first S Day of the week parameter is case insensitive, and accepts: "Mon", "Tue", "Wed", "Thu", "Fr

org.apache.spark.sql.functions

The default date format these functions use is yyyy-MM-dd HH:mm:ss.

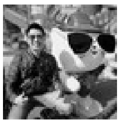
Working with Null/NaN - Column

isNull: Column	True if the current expression is null.

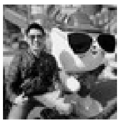


Working with Null/NaN - Column (cont)

isNotNull: Column	True if the current expression is NOT null.



Working with Null/NaN - Column (cont)	
isNaN: Column	True if the current expression is NaN.
org.apache.spark.sql.Column	

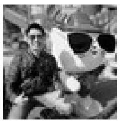


Working with Null/NaN - functions

isnull(e: Column): Column	Return true iff the column is null.

Working with Null/NaN - functions (cont)

isnan(e: Column): Column	Return true iff the column is NaN.

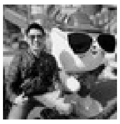


By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Null/NaN - functions (cont)

nanvl(col1: Column, col2: Column): Column	Returns col1 if it is not NaN, or col2 if col1 is NaN. Both inputs should be floating point columns (D



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Null/NaN - functions (cont)

coalesce(e: Column*): Column	Returns the first column that is not null, or null if all inputs are null. For example, coalesce(a, b, c) will return a if a is not r null, or c if both a and b are null but c is not null.
	`// create a movie with null title`
	`case class Movie(actor_name:String, movie_title:String, produced_year:Long)`
	`val badMoviesDF = Seq( Movie(null, null, 2018L), Movie("John Doe", "Awesome Movie", 2018I`
	`// use coalese to handle null value in title column`
	`badMoviesDF.select(coalesce('actor_name, lit("no_name")).as("new_title")).show`
	+----------+
	| new_title|
	+----------+
	| no_name|
	| John Doe|
	+----------+

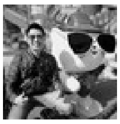
org.apache.spark.sql.functions

Working with Null/NaN - DataFrameNaFunctions

drop(): DataFrame	Returns a new DataFrame that drops rows containing **any** null or NaN values.

Working with Null/NaN - DataFrameNaFunctions (cont)

drop(how: String): DataFrame	Returns a new DataFrame that drops rows containing null or NaN values. If how is **"any"**, then drop rows containing any null or NaN values. If how is **"all"**, then drop rows only if every colu

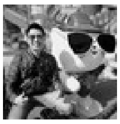


By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.
Page 216 of 100.

Working with Null/NaN - DataFrameNaFunctions (cont)

drop(cols: Seq[String]): DataFrame	(Scala-specific) Returns a new DataFrame that drops rows containing any null or NaN values in the spe



Working with Null/NaN - DataFrameNaFunctions (cont)	
drop(how: String, cols: Seq[String]): DataFrame	(Scala-specific) Returns a new DataFrame that drops rows containing null or NaN values in the specified columns. If hov containing any null or NaN values in the specified columns. If how is "all", then drop rows only if every specified column i

Working with Null/NaN - DataFrameNaFunctions (cont)

drop(minNonNulls: Int): DataFrame	Returns a new DataFrame that drops rows containing less than minNonNulls non-null and non-NaN

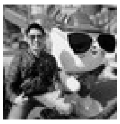
Working with Null/NaN - DataFrameNaFunctions (cont)

drop(minNonNulls: Int, cols: Seq[String]): DataFrame	(Scala-specific) Returns a new DataFrame that drops rows containing less than minNonNulls non specified columns.

Working with Null/NaN - DataFrameNaFunctions (cont)

fill(value: String/Boolean/Double/Long): DataFrame	Returns a new DataFrame that replaces null values in string/boolean columns (or null or NaN v value.

Working with Null/NaN - DataFrameNaFunctions (cont)	
fill(value: String/Boolean/Double/Long, cols: Seq[String]): DataFrame	(Scala-specific) Returns a new DataFrame that replaces null values in specifie columns.



Working with Null/NaN - DataFrameNaFunctions (cont)

fill(valueMap: Map[String, Any]): DataFrame	(Scala-specific) Returns a new DataFrame that replaces null values. The key of the map is the column name, and the valu value. The value must be of the following type: Int, Long, Float, Double, String, Boolean. Replacement values are cast to t e.g. `df.na.fill(Map( "A" -> "unknown", "B" -> 1.0 ))`

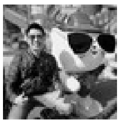
Working with Null/NaN - DataFrameNaFunctions (cont)

replaceT: DataFrame	(Scala-specific) Replaces values matching keys in replacement map.
	col name of the column to apply the value replacement. If col is "*", replacement is applied on all string, numeric or boolean c
	replacement value replacement map. Key and value of replacement map must have the same type, and can only be doubles
	value can have nulls.
	
	// Replaces all occurrences of 1.0 with 2.0 in column "height".
	df.na.replace("height", Map(1.0 -> 2.0));
	
	// Replaces all occurrences of "UNKNOWN" with "unnamed" in column "name".
	df.na.replace("name", Map("UNKNOWN" -> "unnamed"));
	
	// Replaces all occurrences of "UNKNOWN" with "unnamed" in all string columns.
	df.na.replace("*", Map("UNKNOWN" -> "unnamed"));

Working with Null/NaN - DataFrameNaFunctions (cont)

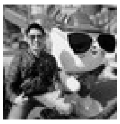
replaceT: DataFrame	(Scala-specific) Replaces values matching keys in replacement map.
	
	// Replaces all occurrences of 1.0 with 2.0 in column "height" and "weight".
	df.na.replace("height" :: "weight" :: Nil, Map(1.0 -> 2.0));
	
	// Replaces all occurrences of "UNKNOWN" with "unnamed" in column "firstname" and "lastname".
	df.na.replace("firstname" :: "lastname" :: Nil, Map("UNKNOWN" -> "unnamed"));

org.apache.spark.sql.DataFrameNaFunctions
use `df.na` to get DataFrameNaFunctions



Working with Sorting - Column

asc: Column	Returns a sort expression based on ascending order of the column.
	
	// Scala: sort a DataFrame by age column in ascending order.
	`df.sort(df("age").asc)`



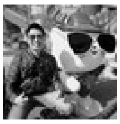
By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Working with Sorting - Column (cont)

asc_nulls_first: Column	Returns a sort expression based on ascending order of the column, and null values return before non-null values.

Working with Sorting - Column (cont)

asc_nulls_last: Column	Returns a sort expression based on ascending order of the column, and null values appear after non-null values.

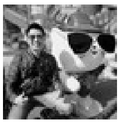


By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Sorting - Column (cont)

desc: Column	Returns a sort expression based on the descending order of the column.

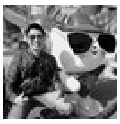


By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Sorting - Column (cont)

desc_nulls_first: Column	Returns a sort expression based on the descending order of the column, and null values appear before non-null v


By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.
Page 230 of 100.

Working with Sorting - Column (cont)

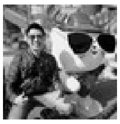
desc_nulls_last: Column	Returns a sort expression based on the descending order of the column, and null values appear after non-null va
org.apache.spark.sql.Column	

Working with Sorting - functions

asc(columnName: String): Column	Returns a sort expression based on ascending order of the column. `df.sort(asc("dept"), desc("age"))`

Working with Sorting - functions (cont)

asc_nulls_first(columnName: String): Column	Returns a sort expression based on ascending order of the column, and null values return bef



Working with Sorting - functions (cont)

asc_nulls_last(columnName: String): Column	Returns a sort expression based on ascending order of the column, and null values appear a

Working with Sorting - functions (cont)

desc(columnName: String): Column	Returns a sort expression based on the descending order of the column.

Working with Sorting - functions (cont)	
desc_nulls_first(columnName: String): Column	Returns a sort expression based on the descending order of the column, and null values appe

Working with Sorting - functions (cont)	
desc_nulls_last(columnName: String): Column	Returns a sort expression based on the descending order of the column, and null values appe
org.apache.spark.sql.functions	

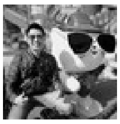
Cheatography

Spark Scala API v2.3 Cheat Sheet

by Ryan (ryan2002) via cheatography.com/108946/cs/21509/

Working with Aggregate functions

count(columnName: String): TypedColumn[Any, Long]	Aggregate function: returns the number of items in a group. count("*"): count null values count(<column_name>): not count null values



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Aggregate functions (cont)

count(e: Column): Column	Aggregate function: returns the number of items in a group.

Working with Aggregate functions (cont)

countDistinct(columnName: String, columnNames: String*): Column	Aggregate function: returns the number of distinct items

Working with Aggregate functions (cont)

countDistinct(expr: Column, exprs: Column*): Column	Aggregate function: returns the number of distinct items in a grou

Working with Aggregate functions (cont)

first(columnName: String): Column	Aggregate function: returns the first value of a column in a group.The function by default returns the first values it sees. It it sees when ignoreNulls is set to true. If all values are null, then null is returned.

Working with Aggregate functions (cont)

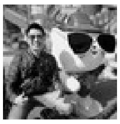
first(e: Column): Column	Aggregate function: returns the first value in a group.

Working with Aggregate functions (cont)

first(columnName: String, ignoreNulls: Boolean): Column	Aggregate function: returns the first value of a column in a gro

Working with Aggregate functions (cont)

first(e: Column, ignoreNulls: Boolean): Column	Aggregate function: returns the first value in a group.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Aggregate functions (cont)

last(columnName: String): Column	Aggregate function: returns the last value of the column in a group.The function by default returns the last values it sees. I value it sees when ignoreNulls is set to true. If all values are null, then null is returned.

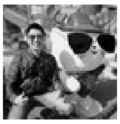
Working with Aggregate functions (cont)

last(e: Column): Column	Aggregate function: returns the last value in a group.

Working with Aggregate functions (cont)

last(columnName: String, ignoreNulls: Boolean): Column	Aggregate function: returns the last value of the column in a gr

Working with Aggregate functions (cont)	
last(e: Column, ignoreNulls: Boolean): Column	Aggregate function: returns the last value in a group.



Working with Aggregate functions (cont)

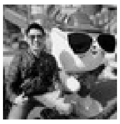
min(columnName: String): Column	Aggregate function: returns the minimum value of the column in a group.

Working with Aggregate functions (cont)

min(e: Column): Column	Aggregate function: returns the minimum value of the expression in a group.

Working with Aggregate functions (cont)

max(columnName: String): Column	Aggregate function: returns the maximum value of the column in a group.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Aggregate functions (cont)

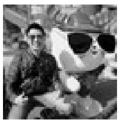
max(e: Column): Column	Aggregate function: returns the maximum value of the expression in a group.

Working with Aggregate functions (cont)

sum(columnName: String): Column	Aggregate function: returns the sum of all values in the given column.

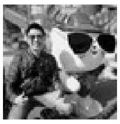
Working with Aggregate functions (cont)

sum(e: Column): Column	Aggregate function: returns the sum of all values in the expression.



Working with Aggregate functions (cont)

sumDistinct(columnName: String): Column	Aggregate function: returns the sum of distinct values in the expression.

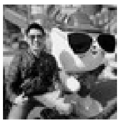


By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Aggregate functions (cont)

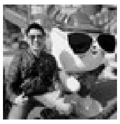
sumDistinct(e: Column): Column	Aggregate function: returns the sum of distinct values in the expression.



Working with Aggregate functions (cont)

avg(columnName: String): Column	Aggregate function: returns the average of the values in a group.

Working with Aggregate functions (cont)	
avg(e: Column): Column	Aggregate function: returns the average of the values in a group.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Aggregate functions (cont)

mean(columnName: String): Column	Aggregate function: returns the average of the values in a group. Alias for avg.

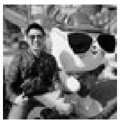
Working with Aggregate functions (cont)

mean(e: Column): Column	Aggregate function: returns the average of the values in a group. Alias for avg.

Working with Aggregate functions (cont)	
variance(columnName: String): Column	Aggregate function: alias for var_samp.

Working with Aggregate functions (cont)

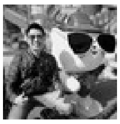
variance(e: Column): Column	Aggregate function: alias for var_samp



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Working with Aggregate functions (cont)

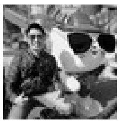
var_samp(columnName: String): Column	Aggregate function: returns the unbiased variance of the values in a group.



Working with Aggregate functions (cont)	
var_samp(e: Column): Column	Aggregate function: returns the unbiased variance of the values in a group.

Working with Aggregate functions (cont)

var_pop(columnName: String): Column	Aggregate function: returns the population variance of the values in a group.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Aggregate functions (cont)

var_pop(e: Column): Column	Aggregate function: returns the population variance of the values in a group.

Working with Aggregate functions (cont)	
stddev(columnName: String): Column	Aggregate function: alias for stddev_samp.

Working with Aggregate functions (cont)

stddev(e: Column): Column	Aggregate function: alias for stddev_samp.

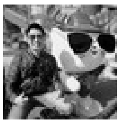
Working with Aggregate functions (cont)	
stddev_samp(columnName: String): Column	Aggregate function: returns the sample standard deviation of the expression in a gro

Working with Aggregate functions (cont)

stddev_samp(e: Column): Column	Aggregate function: returns the sample standard deviation of the expression in a group.

Working with Aggregate functions (cont)

stddev_pop(columnName: String): Column	Aggregate function: returns the population standard deviation of the expression in a gro



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Working with Aggregate functions (cont)

stddev_pop(e: Column): Column	Aggregate function: returns the population standard deviation of the expression in a group.

Working with Aggregate functions (cont)	
skewness(columnName: String): Column	Aggregate function: returns the skewness of the values in a group.

Working with Aggregate functions (cont)

skewness(e: Column): Column	Aggregate function: returns the skewness of the values in a group.

Working with Aggregate functions (cont)

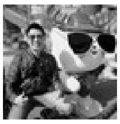
kurtosis(columnName: String): Column	Aggregate function: returns the kurtosis of the values in a group.

Cheatography

Spark Scala API v2.3 Cheat Sheet

by Ryan (ryan2002) via cheatography.com/108946/cs/21509/

Working with Aggregate functions (cont)	
kurtosis(e: Column): Column	Aggregate function: returns the kurtosis of the values in a group.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

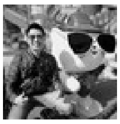
Working with Aggregate functions (cont)	
corr(columnName1: String, columnName2: String): Column	Aggregate function: returns the Pearson Correlation Coefficient for tw

Working with Aggregate functions (cont)

corr(column1: Column, column2: Column): Column	Aggregate function: returns the Pearson Correlation Coefficient for two colu

Working with Aggregate functions (cont)

covar_samp(columnName1: String, columnName2: String): Column	Aggregate function: returns the sample covariance for t



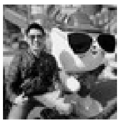
Cheatography

Spark Scala API v2.3 Cheat Sheet

by Ryan (ryan2002) via cheatography.com/108946/cs/21509/

Working with Aggregate functions (cont)

covar_samp(column1: Column, column2: Column): Column	Aggregate function: returns the sample covariance for two co



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Aggregate functions (cont)

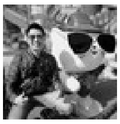
covar_pop(columnName1: String, columnName2: String): Column	Aggregate function: returns the population covariance for t

Working with Aggregate functions (cont)

covar_pop(column1: Column, column2: Column): Column	Aggregate function: returns the population covariance for two col

Working with Aggregate functions (cont)

collect_list(columnName: String): Column	Aggregate function: returns a list of objects with duplicates.



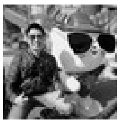
By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Aggregate functions (cont)	
collect_list(e: Column): Column	Aggregate function: returns a list of objects with duplicates.

Working with Aggregate functions (cont)

collect_set(columnName: String): Column	Aggregate function: returns a set of objects with duplicate elements eliminated.



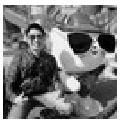
Working with Aggregate functions (cont)

collect_set(e: Column): Column	Aggregate function: returns a set of objects with duplicate elements eliminated.

org.apache.spark.sql.functions

Working with Aggregate - RelationalGroupedDataset

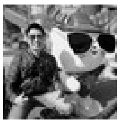
agg(expr: Column, exprs: Column*): DataFrame	Compute aggregates by specifying a series of aggregate columns. Note that this function by default retains the g
	To not retain grouping columns, set spark.sql.retainGroupColumns to false.
	
	`import org.apache.spark.sql.functions._`
	`df.groupBy("department").agg(max("age"), sum("expense"))`



Working with Aggregate - RelationalGroupedDataset (cont)	
agg(exprs: Map[String, String]): DataFrame	(Scala-specific) Compute aggregates by specifying a map from column name to aggregate methods. The re contain the grouping columns. `df.groupBy("department").agg(Map( "age" -> "max", "expense" -> "sum" ))`

Working with Aggregate - RelationalGroupedDataset (cont)

count(): DataFrame	Count the number of rows for each group. The resulting DataFrame will also contain the grouping columns.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Aggregate - RelationalGroupedDataset (cont)

max(colNames: String*): DataFrame	Compute the max value for each numeric columns for each group. The resulting DataFrame will also contain the grou columns are given, only compute the max values for them.

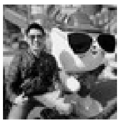
Working with Aggregate - RelationalGroupedDataset (cont)	
min(colNames: String*): DataFrame	Compute the min value for each numeric column for each group. The resulting DataFrame will also contain the groupi columns are given, only compute the min values for them.

Working with Aggregate - RelationalGroupedDataset (cont)

sum(colNames: String*): DataFrame	Compute the sum for each numeric columns for each group. The resulting DataFrame will also contain the grouping columns are given, only compute the sum for them.

Working with Aggregate - RelationalGroupedDataset (cont)

avg(colNames: String*): DataFrame	Compute the mean value for each numeric columns for each group.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Aggregate - RelationalGroupedDataset (cont)

mean(colNames: String*): DataFrame	Compute the average value for each numeric columns for each group. This is an alias for avg. The resulting DataFrame columns. When specified columns are given, only compute the average values for them.

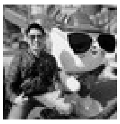
Working with Aggregate - RelationalGroupedDataset (cont)	
pivot(pivotColumn: String): RelationalGr-oupedDataset	Pivots a column of the current DataFrame and performs the specified aggregation. There are two versions of pivot functi specify the list of distinct values to pivot on, and one that does not. The latter is more concise but less efficient, because list of distinct values internally.

Working with Aggregate - RelationalGroupedDataset (cont)

pivot(pivotColumn: String, values: Seq[Any]): RelationalGroupedDataset	Pivots a column of the current DataFrame and performs the specified aggregation. There are two versions of pivot caller to specify the list of distinct values to pivot on, and one that does not. The latter is more concise but less effi first compute the list of distinct values internally.

org.apache.spark.sql.RelationalGroupedDataset
Use `df.groupBy("xxx")` to get RelationalGroupedDataset

Working with Collection - functions	
size(e: Column): Column	Returns length of array or map.



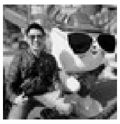
By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.
Page 298 of 100.

Working with Collection - functions (cont)	
array_contains(column: Column, value: Any): Column	Returns null if the array is null, true if the array contains value, and false o

Working with Collection - functions (cont)

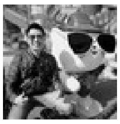
sort_array(e: Column): Column	Sorts the input array for the given column in ascending order, according to the natural ordering of the array



Working with Collection - functions (cont)

sort_array(e: Column, asc: Boolean): Column	Sorts the input array for the given column in ascending or descending order, according to the natur elements.

Working with Collection - functions (cont)	
explode(e: Column): Column	Creates a new row for each element in the given array or map column.

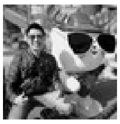


By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Working with Collection - functions (cont)

explode_outer(e: Column): Column	Creates a new row for each element in the given array or map column. Unlike explode, if the array/map is nu produced.

org.apache.spark.sql.functions



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

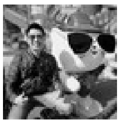
Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Window - functions

rank(): Column	Window function: returns the rank of rows within a window partition. The difference between rank and dense_rank is that dense_rank leaves no gaps in ranking sequence when there are ties. That is, if you using dense_rank and had three people tie for second place, you would say that all three were in second place and that the next person me sequential numbers, making the person that came in third place (after the ties) would register as coming in fifth. e.g. 1, 2, 2, 2, 5

Working with Window - functions (cont)

dense_ rank(): Column	Window function: returns the rank of rows within a window partition, without any gaps. The difference between rank and dense_rank is that denseRank leaves no gaps in ranking sequence when there are ties. That is, if you using dense_rank and had three people tie for second place, you would say that all three were in second place and that the next person me sequential numbers, making the person that came in third place (after the ties) would register as coming in fifth. e.g. 1, 2, 2, 2, 3



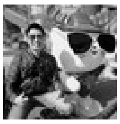
By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.

Working with Window - functions (cont)	
percent_rank(): Column	Window function: returns the relative rank (i.e. percentile) of rows within a window partition. This is computed by: (rank of row in its partition - 1) / (number of rows in the partition - 1)

Working with Window - functions (cont)

row_number(): Column	Window function: returns a sequential number starting at 1 within a window partition.



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.
Page 307 of 100.

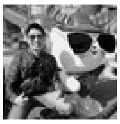
Working with Window - functions (cont)

cume_dist(): Column	Window function: returns the cumulative distribution of values within a window partition, i.e.
	`N = total number of rows in the partition`
	`cumeDist(x) = number of values before (and including) x / N`

```
import org.apache.spark.sql.expressions.Window
import org.apache.spark.sql.functions.colval
windowSpec = Window.partitionBy("CustomerId", "date").orderBy(col("Quantity").desc).rowsBetween(Window.unboundedPreceding, Window.curre

val purchaseRank = rank().over(windowSpec)

dfWithDate.where("CustomerId IS NOT NULL").orderBy("CustomerId")
.select(
col("CustomerId"),
col("date"),
col("Quantity"),
purchaseRank.alias("quantityRank")).show()
```

org.apache.spark.sql.expresseions.WindowSpec

rowBetween	todo



By **Ryan** (ryan2002)
cheatography.com/ryan2002/

Published 31st January, 2020.
Last updated 5th February, 2020.
Page 309 of 100.

org.apache.spark.sql.expresseions.WindowSpec (cont)	
rangeBetween	todo