| Working with Different Types of Data | | |
|---|---|---|
| **Keywork** | **Explanation** | **Example** |
| | select count | df.select(count("StockCode"))<br>spark.sql("SELECT count(StockCode) FROM dfTable") |
| count distintct | count distintct | df.select(countDistinct("StockCode"))<br>spark.sql("SELECT count(distinct(StockCode)) FROM dfTable") |
| approx_count_distinc | approx_count_distinct | df.select(approx_count_distinct("StockCode", 0.1))<br>spark.sql("SELECT approx_count_distinct(StockCode) FROM dfTable") |
| first last | first last | df.select(first("StockCode"), last("StockCode"))<br>spark.sql("SELECT first(StockCode), last(StockCode) FrOM dfTable") |
| min max | min max | df.select(min("Quantity"), max("Quantity"))<br>spark.sql("SELECT min(Quantity), max(Quantity) FrOM dfTable") |
| sum | sum | df.select(sum("Quantity"))<br>spark.sql("SELECT sum(Quantity) FrOM dfTable") |
| sumDistinct | sumDistinct | df.select(sumDistinct("Quantity")) |
| avg | avg | df.select(avg("Quantity"))<br>spark.sql("SELECT avg(Quantity) FROM dfTable") |
| selectExpr | selectExpr | df..selectExpr( "total_purchases/total_transactions") |
| var_pop var_samp<br>stddev_pop<br>stddev_samp | Variance and Standard Deviation | df.select(var_pop("Quantity"), var_samp("Quantity"), stddev_pop("Quantity"), stddev_samp("Quantity"))<br>spark.sql("""<br>SELECT var_pop(Quantity), var_samp(Quantity), stddev_pop(Quantity), stddev_samp(Quantity)<br>FROM dfTable""") |
| skewness kurtosis | skewness and kurtosis | df.select(skewness("Quantity"), kurtosis("Quantity")) |
| corr covar_samp<br>covar_pop | Covariance and Correlation | df.select(corr("InvoiceNo", "Quantity"), covar_samp("InvoiceNo", "Quantity"),<br>covar_pop("InvoiceNo", "Quantity")) |
| agg collect_set list | Aggregating to Complex Types | df.agg(collect_set("Country"), collect_list("Country")) |
| groupBy | Grouping | df.groupBy("InvoiceNo", "CustomerId").count()<br>spark.sql("SELECT count(*) FROM dfTable GROUP BY InvoiceNo, CustomerId") |
| groupBy express | Grouping with Expressions | df.groupBy("InvoiceNo").agg(<br>count("Quantity").alias("quan"),<br>expr("count(Quantity)")) |
| maps | Grouping with Maps | df.groupBy("InvoiceNo").agg("Quantity"->"avg", "Quantity"->"stddev_pop")<br>spark.sql("SELECT avg(Quantity), stddev_pop(Quantity), InvoiceNo FROM dfTable GROUP BY InvoiceNo") |
| | Windows functions | |
| rollup | rollup is a multidimensional aggregation that performs a variety of group-by style calculations | val rolledUpDF = dfNoNull.rollup("Date", "Country").agg(sum("Quantity"))<br>.selectExpr("Date", "Country", "`sum(Quantity)` as total_quantity")<br>.orderBy("Date")<br><br>rolledUpDF.where("Country IS NULL") |
| cube | Cube (ie rollup to a level deeper) - Rather than treating elements hierarchically, a cube<br>does the same thing across all dimensions. | dfNoNull.cube("Date", "Country").agg(sum(col("Quantity")))<br>.select("Date", "Country", "sum(Quantity)").orderBy("Date") |
| pivot | Pivots make it possible for you to convert a row into a column | val pivoted = dfWithDate.groupBy("date").pivot("Country").sum() |
| | User-Defined Aggregation Functions | |