GroupBy and Aggregate Functions

Let's learn how to use GroupBy and Aggregate methods on a DataFrame. GroupBy allows you to group rows together based off some column value, for example, you could group together sales data by the day the sale occured, or group repeast customer data based off the name of the customer. Once you've performed the GroupBy operation you can use an aggregate function off that data. An aggregate function aggregates multiple rows of data into a single output, such as taking the sum of inputs, or counting the number of inputs.

Let's see some examples on an example dataset!

In [1]:
```
from pyspark.sql import SparkSession
```

In [2]:
```
# May take a little while on a local computer
spark = SparkSession.builder.appName("groupbyagg").getOrCreate()
```

Read in the customer sales data

In [3]:
```
df = spark.read.csv('sales_info.csv',inferSchema=True,header=True)
```

In [4]:
```
df.printSchema()
```

```
root
 |-- Company: string (nullable = true)
 |-- Person: string (nullable = true)
 |-- Sales: double (nullable = true)
```

In [8]:
```
df.show()
```

```
+-------+-------+-----+
|Company| Person|Sales|
+-------+-------+-----+
|   GOOG|    Sam|200.0|
|   GOOG|Charlie|120.0|
|   GOOG|  Frank|340.0|
|   MSFT|   Tina|600.0|
|   MSFT|    Amy|124.0|
|   MSFT|Vanessa|243.0|
|     FB|   Carl|870.0|
|     FB|  Sarah|350.0|
|   APPL|   John|250.0|
|   APPL|  Linda|130.0|
|   APPL|   Mike|750.0|
|   APPL|  Chris|350.0|
+-------+-------+-----+
```

Let's group together by company!

In [9]:
```
df.groupBy("Company")
```

Out[9]:
```
<pyspark.sql.group.GroupedData at 0x109915f28>
```

This returns a GroupedData object, off of which you can all various methods

In [10]:
```
# Mean
df.groupBy("Company").mean().show()
```

```
+-------+-----------------+
|Company|       avg(Sales)|
+-------+-----------------+
|   APPL|            370.0|
|   GOOG|            220.0|
|     FB|            610.0|
|   MSFT|322.3333333333333|
+-------+-----------------+
```

In [11]:
```
# Count
df.groupBy("Company").count().show()
```

```
+-------+-----+
|Company|count|
```

```
+-------+-----+
|   APPL|    4|
|   GOOG|    3|
|     FB|    2|
|   MSFT|    3|
+-------+-----+
```

In [12]:
```
# Max
df.groupBy("Company").max().show()
```

```
+-------+----------+
|Company|max(Sales)|
+-------+----------+
|   APPL|     750.0|
|   GOOG|     340.0|
|     FB|     870.0|
|   MSFT|     600.0|
+-------+----------+
```

In [13]:
```
# Min
df.groupBy("Company").min().show()
```

```
+-------+----------+
|Company|min(Sales)|
+-------+----------+
|   APPL|     130.0|
|   GOOG|     120.0|
|     FB|     350.0|
|   MSFT|     124.0|
+-------+----------+
```

In [15]:
```
# Sum
df.groupBy("Company").sum().show()
```

```
+-------+----------+
|Company|sum(Sales)|
+-------+----------+
|   APPL|    1480.0|
|   GOOG|     660.0|
|     FB|    1220.0|
|   MSFT|     967.0|
+-------+----------+
```

Check out this link for more info on other methods: http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark-sql-module (http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark-sql-module)

Not all methods need a groupby call, instead you can just call the generalized .agg() method, that will call the aggregate across all rows in the dataframe column specified. It can take in arguments as a single column, or create multiple aggregate calls all at once using dictionary notation.

For example:

In [18]:
```
# Max sales across everything
df.agg({'Sales':'max'}).show()
```

```
+----------+
|max(Sales)|
+----------+
|     870.0|
+----------+
```

In [22]:
```
# Could have done this on the group by object as well:
```

In [23]:
```
grouped = df.groupBy("Company")
```

In [25]:
```
grouped.agg({"Sales":'max'}).show()
```

```
+-------+----------+
|Company|max(Sales)|
+-------+----------+
|   APPL|     750.0|
|   GOOG|     340.0|
|     FB|     870.0|
|   MSFT|     600.0|
```

```
+-------+----------+
```

Functions

There are a variety of functions you can import from pyspark.sql.functions. Check out the documentation for the full list available: http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#module-pyspark.sql.functions (http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#module-pyspark.sql.functions)

In [36]:
```
from pyspark.sql.functions import countDistinct, avg,stddev
```

In [29]:
```
df.select(countDistinct("Sales")).show()
```

```
+----------------------+
|count(DISTINCT Sales)|
+----------------------+
|                    11|
+----------------------+
```

Often you will want to change the name, use the .alias() method for this:

In [31]:
```
df.select(countDistinct("Sales").alias("Distinct Sales")).show()
```

```
+--------------+
|Distinct Sales|
+--------------+
|            11|
+--------------+
```

In [35]:
```
df.select(avg('Sales')).show()
```

```
+-----------------+
|       avg(Sales)|
+-----------------+
|360.5833333333333|
+-----------------+
```

In [38]:
```
df.select(stddev("Sales")).show()
```

```
+------------------+
|stddev_samp(Sales)|
+------------------+
|250.08742410799007|
+------------------+
```

That is a lot of precision for digits! Let's use the format_number to fix that!

In [39]:
```
from pyspark.sql.functions import format_number
```

In [40]:
```
sales_std = df.select(stddev("Sales").alias('std'))
```

In [41]:
```
sales_std.show()
```

```
+------------------+
|               std|
+------------------+
|250.08742410799007|
+------------------+
```

In [42]:
```
# format_number("col_name",decimal places)
sales_std.select(format_number('std',2)).show()
```

```
+----------------------+
|format_number(std, 2)|
+----------------------+
|                250.09|
+----------------------+
```

Order B…

Order By

You can easily sort with the orderBy method:

In [43]:
```
# OrderBy
# Ascending
df.orderBy("Sales").show()
```

```
+-------+-------+-----+
|Company| Person|Sales|
+-------+-------+-----+
|   GOOG|Charlie|120.0|
|   MSFT|    Amy|124.0|
|   APPL|  Linda|130.0|
|   GOOG|    Sam|200.0|
|   MSFT|Vanessa|243.0|
|   APPL|   John|250.0|
|   GOOG|  Frank|340.0|
|     FB|  Sarah|350.0|
|   APPL|  Chris|350.0|
|   MSFT|   Tina|600.0|
|   APPL|   Mike|750.0|
|     FB|   Carl|870.0|
+-------+-------+-----+
```

In [47]:
```
# Descending call off the column itself.
df.orderBy(df["Sales"].desc()).show()
```

```
+-------+-------+-----+
|Company| Person|Sales|
+-------+-------+-----+
|     FB|   Carl|870.0|
```