PySpark H.W Session 1

Let's get some quick practice with your new Spark DataFrame skills, you will be asked some basic questions about some stock market data, in this case Walmart Stock from the years 2012-2017. This exercise will just ask a bunch of questions, unlike the future machine learning exercises, which will be a little looser and be in the form of "Consulting Projects", but more on that later!

For now, just answer the questions and complete the tasks below.

Use the walmart_stock.csv file to Answer and complete the tasks below!

Start a simple Spark Session

In [3]:

```
import findspark
findspark.init()

import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('Walmart_stock').getOrCreate()
sc = spark.sparkContext
```

Load the Walmart Stock CSV File, have Spark infer the data types.

In [30]:

```
v('walmart_stock.csv',header = True,inferSchema=True, timestampFormat='yyyy-MM-dd')
```

```
root
 |-- Date: timestamp (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: integer (nullable = true)
 |-- Adj Close: double (nullable = true)
```

What are the column names?

In [22]:

```
df.columns
```

Out[22]:

```
['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close']
```

What does the Schema look like?

In [32]:

```
df.printSchema()
```

```
root
 |-- Date: timestamp (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: integer (nullable = true)
 |-- Adj Close: double (nullable = true)
```

In [ ]:

```
#Required OUTPUT
```

```
root
 |-- Date: timestamp (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: integer (nullable = true)
 |-- Adj Close: double (nullable = true)
```

Print out the first 5 columns.

In [33]:

```
df.head(5)
```

Out[33]:

```
[Row(Date=datetime.datetime(2012, 1, 3, 0, 0), Open=59.970001, High=6
1.060001, Low=59.869999, Close=60.330002, Volume=12668800, Adj Close=5
2.619234999999996),
 Row(Date=datetime.datetime(2012, 1, 4, 0, 0), Open=60.20999899999999
6, High=60.349998, Low=59.470001, Close=59.709998999999996, Volume=959
3300, Adj Close=52.078475),
 Row(Date=datetime.datetime(2012, 1, 5, 0, 0), Open=59.349998, High=5
9.619999, Low=58.369999, Close=59.419998, Volume=12768200, Adj Close=5
1.825539),
 Row(Date=datetime.datetime(2012, 1, 6, 0, 0), Open=59.419998, High=5
9.450001, Low=58.869999, Close=59.0, Volume=8069400, Adj Close=51.4592
2),
 Row(Date=datetime.datetime(2012, 1, 9, 0, 0), Open=59.029999, High=5
9.549999, Low=58.919998, Close=59.18, Volume=6679300, Adj Close=51.616
215000000004)]
```

In [34]:

```
df.show(5)
```

```
+-------------------+------------------+---------+---------+----------
--------+--------+------------------+
|               Date|              Open|     High|      Low|
Close|  Volume|         Adj Close|
+-------------------+------------------+---------+---------+----------
--------+--------+------------------+
|2012-01-03 00:00:00|         59.970001|61.060001|59.869999|         6
0.330002|12668800|52.619234999999996|
|2012-01-04 00:00:00|60.209998999999996|60.349998|59.470001|59.7099989
99999996| 9593300|         52.078475|
|2012-01-05 00:00:00|         59.349998|59.619999|58.369999|         5
9.419998|12768200|         51.825539|
|2012-01-06 00:00:00|         59.419998|59.450001|58.869999|
59.0| 8069400|          51.45922|
|2012-01-09 00:00:00|         59.029999|59.549999|58.919998|
59.18| 6679300|51.616215000000004|
+-------------------+------------------+---------+---------+----------
--------+--------+------------------+
only showing top 5 rows
```

In [ ]:

```
#Required OUTPUT
```

```
Row(Date=datetime.datetime(2012, 1, 3, 0, 0), Open=59.970001, High=61.
060001, Low=59.869999, Close=60.330002, Volume=12668800, Adj Close=52.
619234999999996)


Row(Date=datetime.datetime(2012, 1, 4, 0, 0), Open=60.209998999999996,
High=60.349998, Low=59.470001, Close=59.709998999999996, Volume=959330
0, Adj Close=52.078475)


Row(Date=datetime.datetime(2012, 1, 5, 0, 0), Open=59.349998, High=59.
619999, Low=58.369999, Close=59.419998, Volume=12768200, Adj Close=51.
825539)


Row(Date=datetime.datetime(2012, 1, 6, 0, 0), Open=59.419998, High=59.
450001, Low=58.869999, Close=59.0, Volume=8069400, Adj Close=51.45922)


Row(Date=datetime.datetime(2012, 1, 9, 0, 0), Open=59.029999, High=59.
549999, Low=58.919998, Close=59.18, Volume=6679300, Adj Close=51.61621
5000000004)
```

Use describe() to learn about the DataFrame.

In [39]:

```
df.describe().show()
```

```
+-------+------------------+----------------+-----------------+------
-----------+-----------------+-----------------+
|summary|              Open|            High|              Low|
Close|           Volume|        Adj Close|
+-------+------------------+----------------+-----------------+------
-----------+-----------------+-----------------+
|  count|              1258|            1258|             1258|
1258|             1258|             1258|
|   mean| 72.35785375357709|72.83938807631165| 71.9186009594594|72.388
44998012726|8222093.481717011|67.23883848728146|
| stddev|  6.76809024470826|6.768186808159218|6.744075756255496|6.7568
59163732991|  4519780.8431556|6.722609449996857|
|    min|56.389998999999996|        57.060001|        56.299999|
56.419998|          2094900|        50.363689|
|    max|         90.800003|        90.970001|            89.25|
90.470001|         80898100|84.91421600000001|
+-------+------------------+----------------+-----------------+------
-----------+-----------------+-----------------+
```

In [ ]:

```
#Required OUTPUT
```

```
+-------+------------------+----------------+-----------------+------
-----------+-----------------+-----------------+
|summary|              Open|            High|              Low|
Close|           Volume|        Adj Close|
+-------+------------------+----------------+-----------------+------
-----------+-----------------+-----------------+
|  count|              1258|            1258|             1258|
1258|             1258|             1258|
|   mean| 72.35785375357709|72.83938807631165| 71.9186009594594|72.388
44998012726|8222093.481717011|67.23883848728146|
| stddev|  6.76809024470826|6.768186808159218|6.744075756255496|6.7568
59163732991|  4519780.8431556|6.722609449996857|
|    min|56.389998999999996|        57.060001|        56.299999|
56.419998|          2094900|        50.363689|
|    max|         90.800003|        90.970001|            89.25|
90.470001|         80898100|84.91421600000001|
+-------+------------------+----------------+-----------------+------
-----------+-----------------+-----------------+
```

Bonus Question!

There are too many decimal places for mean and stddev in the describe() dataframe. Format the numbers to just show up to two decimal places. Pay careful attention to the datatypes that .describe() returns, we didn't cover how to do this exact formatting, but we covered something very similar. Check this link for a hint (http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.Column.cast)

If you get stuck on this, don't worry, just view the solutions.

In [41]:

```
df.describe().printSchema()
```

```
root
 |-- summary: string (nullable = true)
 |-- Open: string (nullable = true)
 |-- High: string (nullable = true)
 |-- Low: string (nullable = true)
 |-- Close: string (nullable = true)
 |-- Volume: string (nullable = true)
 |-- Adj Close: string (nullable = true)
```

In [82]:

```
type(df.describe()[['summary']])
```

Out[82]:

```
pyspark.sql.dataframe.DataFrame
```

In [58]:

```
from pyspark.sql.functions import round
```

In [91]:

```
df.describe().select([round(c, 3).alias(c) for c in df.describe().columns if (c!='s
```

```
+------+------+------+------+-----------+---------+
|  Open|  High|   Low| Close|     Volume|Adj Close|
+------+------+------+------+-----------+---------+
|1258.0|1258.0|1258.0|1258.0|     1258.0|   1258.0|
|72.358|72.839|71.919|72.388|8222093.482|   67.239|
| 6.768| 6.768| 6.744| 6.757|4519780.843|    6.723|
| 56.39| 57.06|  56.3| 56.42|  2094900.0|   50.364|
|  90.8| 90.97| 89.25| 90.47|  8.08981E7|   84.914|
+------+------+------+------+-----------+---------+
```

Create a new dataframe with a column called HV Ratio that is the ratio of the High Price versus volume of stock traded for a day.

In [115]:

```
from pyspark.sql.functions import *
df.select(expr('High / Volume').alias('HV Ratio')).show()
```

```
+--------------------+
|            HV Ratio|
+--------------------+
|4.819714653321546E-6|
|6.290848613094555E-6|
|4.669412994783916E-6|
|7.367338463826307E-6|
|8.915604778943901E-6|
|8.644477436914568E-6|
|9.351828421515645E-6|
| 8.29141562102703E-6|
|7.712212102001476E-6|
|7.071764823529412E-6|
|1.015495466386981E-5|
|6.576354146362592...|
| 5.90145296180676E-6|
|8.547679455011844E-6|
|8.420709512685392E-6|
|1.041448341728929...|
|8.316075414862431E-6|
|9.721183814992126E-6|
|8.029436027707578E-6|
|6.307432259386365E-6|
+--------------------+
only showing top 20 rows
```

In [117]:

```
# Another Solution
df.select((col('High') / col('Volume')).alias('HV Ratio')).show()
```

```
+--------------------+
|            HV Ratio|
+--------------------+
|4.819714653321546E-6|
|6.290848613094555E-6|
|4.669412994783916E-6|
|7.367338463826307E-6|
|8.915604778943901E-6|
|8.644477436914568E-6|
|9.351828421515645E-6|
| 8.29141562102703E-6|
|7.712212102001476E-6|
|7.071764823529412E-6|
|1.015495466386981E-5|
|6.576354146362592...|
| 5.90145296180676E-6|
|8.547679455011844E-6|
|8.420709512685392E-6|
|1.041448341728929...|
|8.316075414862431E-6|
|9.721183814992126E-6|
|8.029436027707578E-6|
|6.307432259386365E-6|
+--------------------+
only showing top 20 rows
```

What day had the Peak High in Price?

In [189]:

```
df.select(expr('max(High)')).show()
```

```
+---------+
|max(High)|
+---------+
|90.970001|
+---------+
```

In [187]:

```
df.agg({'High':'max'}).collect()
#df.select('Date').where(df.select(col('High')) == df.select(expr('max(High)')))
```

Out[187]:

```
[Row(max(High)=90.970001)]
```

In [186]:

```
df.select('Date').where(col("High")==90.970001).collect()
```

Out[186]:

```
[Row(Date=datetime.datetime(2015, 1, 13, 0, 0))]
```

In [ ]:

```
# Required OUTPUT
```

Out[88]:

```
datetime.datetime(2015, 1, 13, 0, 0)
```

What is the mean of the Close column?

In [119]:

```
df.select(avg(col('Close'))).show()
```

```
+-----------------+
|       avg(Close)|
+-----------------+
|72.38844998012726|
+-----------------+
```

What is the max and min of the Volume column?

In [191]:

```
df.select(max(col('Volume')),min(col('Volume'))).show()
```

```
+-----------+-----------+
|max(Volume)|min(Volume)|
+-----------+-----------+
|   80898100|    2094900|
+-----------+-----------+
```

In [ ]:

```
#Required OUTPUT
```

```
+-----------+-----------+
|max(Volume)|min(Volume)|
+-----------+-----------+
|   80898100|    2094900|
+-----------+-----------+
```

How many days was the Close lower than 60 dollars?

In [196]:

```
df.select('Date').where(col("Close")<60).count()
```

Out[196]:

```
81
```

In [ ]:

```
#Required OUTPUT
```

Out[100]:

```
81
```

What percentage of the time was the High greater than 80 dollars ?

In other words, (Number of Days High>80)/(Total Days in the dataset)

In [200]:

```
df.select('Date').where(col("High")>80).count() / df[['Date']].count() *100
```

Out[200]:

```
9.141494435612083
```

In [ ]:

```
#Required OUTPUT
```

Out[107]:

```
9.141494435612083
```

What is the Pearson correlation between High and Volume?

Hint (http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrameStatFunctions.co

In [203]:

```
df.corr('High','Volume')
```

Out[203]:

```
-0.3384326061737161
```

In [ ]:

```
#Required OUTPUT
```

```
+------------------+
| corr(High, Volume)|
+------------------+
|-0.3384326061737161|
+------------------+
```

What is the max High per year?

In [256]:

```
from pyspark.sql.functions import year

df.groupby((year('Date')).alias('Year')).agg({'High': 'max'}).show()
```

```
+----+---------+
|Year|max(High)|
+----+---------+
|2015|90.970001|
|2013|81.370003|
|2014|88.089996|
|2012|77.599998|
|2016|75.190002|
+----+---------+
```

In [ ]:

```
#Required OUTPUT
```

```
+----+---------+
|Year|max(High)|
+----+---------+
|2015|90.970001|
|2013|81.370003|
|2014|88.089996|
|2012|77.599998|
|2016|75.190002|
+----+---------+
```

What is the average Close for each Calendar Month?

In other words, across all the years, what is the average Close price for Jan,Feb, Mar, etc... Your result will have a value for each of these months.

In [264]:

```
(month('Date')).alias('Month')).agg({'Close': 'avg'}).orderBy(month('Date')).show()
```

```
+-----+-----------------+
|Month|       avg(Close)|
+-----+-----------------+
|    1|71.44801958415842|
|    2|  71.306804443299|
|    3|71.77794377570092|
|    4|72.97361900952382|
|    5|72.30971688679247|
|    6| 72.4953774245283|
|    7|74.43971943925233|
|    8|73.02981855454546|
|    9|72.18411785294116|
|   10|71.57854545454543|
|   11| 72.1110893069307|
|   12|72.84792478301885|
+-----+-----------------+
```

In [ ]:

```
#Required OUTPUT
```

```
+-----+-----------------+
|Month|       avg(Close)|
+-----+-----------------+
|    1|71.44801958415842|
|    2|  71.306804443299|
|    3|71.77794377570092|
|    4|72.97361900952382|
|    5|72.30971688679247|
|    6| 72.4953774245283|
|    7|74.43971943925233|
|    8|73.02981855454546|
|    9|72.18411785294116|
|   10|71.57854545454543|
|   11| 72.1110893069307|
|   12|72.84792478301885|
+-----+-----------------+
```