# Spark DataFrames Project Exercise

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 [1]:
```
import findspark

#findspark.init takes twor aregument spark-home and location of python executable
findspark.init("c:/spark","c:/Anaconda/python.exe")

#let us import pyspark. PySpark is the collaboration of Apache Spark and Python.
#we can work with spark using scala language
import pyspark
```

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

In [2]:
```
#In order to work with dataframes, we need to start a spark session
from pyspark.sql import SparkSession

# We are going to start the spark session by applying it. The convention name it
spark = SparkSession.builder.appName('walmart').getOrCreate()
```

In [4]:
```
#read csv data file to datafram
df=spark.read.csv('data/walmart_stock.csv',inferSchema=True,header=True)
```

#### What are the column names?

In [5]:
```
#show columns names
df.columns
```

Out[5]:
```
['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close']
```

#### What does the Schema look like?

In [6]:
```
#show table's schema
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)
```

**Print out the first 5 columns.**

In [7]:
```
#print first 5 rows and seprate by empty line
for row in df.head(5):
    print(row)
    print('\n')
```

```
Row(Date=datetime.datetime(2012, 1, 3, 0, 0), Open=59.970001, High=61.060001, L
ow=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=9593300, Adj Close=52.0
78475)


Row(Date=datetime.datetime(2012, 1, 5, 0, 0), Open=59.349998, High=59.619999, L
ow=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, L
ow=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, L
ow=58.919998, Close=59.18, Volume=6679300, Adj Close=51.616215000000004)
```

**Use describe() to learn about the DataFrame.**

In [8]:
```
#show describe of dataframe
df.describe().show()
```

```
+-------+------------------+-----------------+-----------------+-----------------+--------------
--+-----------------+-----------------+
|summary|              Open|             High|              Low|            Clo
se|           Volume|        Adj Close|
+-------+------------------+-----------------+-----------------+-----------------+--------------
--+-----------------+-----------------+
|  count|              1258|             1258|             1258|             12
58|             1258|             1258|
|   mean| 72.35785375357709|72.83938807631165| 71.9186009594594|72.388449980127
26|8222093.481717011|67.23883848728146|
| stddev|  6.76809024470826|6.768186808159218|6.744075756255496|6.7568591637329
91|  4519780.8431556|6.722609449996857|
|    min|56.389998999999996|        57.060001|        56.299999|        56.4199
98|          2094900|        50.363689|
|    max|         90.800003|        90.970001|            89.25|        90.4700
01|         80898100|84.91421600000001|
+-------+------------------+-----------------+-----------------+-----------------+--------------
--+-----------------+-----------------+
```

In [10]:
```
#print describe table's schema
describe_df=df.describe()
describe_df.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 [38]:
```
from pyspark.sql.functions import format_number
describe_df.select(describe_df['summary'],
                   format_number(describe_df['open'].cast('float'),2).alias('Oper
                   format_number(describe_df['High'].cast('float'),2).alias('High
                   format_number(describe_df['Low'].cast('float'),2).alias('Low'
                   format_number(describe_df['Close'].cast('float'),2).alias('Cl
                   describe_df['Volume'].cast('int'),2).alias('Volume')
                   ).show()
```

```
  File "<ipython-input-38-20b48a196a6e>", line 8
    ).show()
    ^
IndentationError: unexpected indent
```

In [80]:

```
+-------+--------+--------+--------+--------+--------+
|summary|    Open|    High|     Low|   Close|  Volume|
+-------+--------+--------+--------+--------+--------+
|  count|1,258.00|1,258.00|1,258.00|1,258.00|    1258|
|   mean|   72.36|   72.84|   71.92|   72.39| 8222093|
| stddev|    6.77|    6.77|    6.74|    6.76| 4519781|
|    min|   56.39|   57.06|   56.30|   56.42| 2094900|
|    max|   90.80|   90.97|   89.25|   90.47|80898100|
+-------+--------+--------+--------+--------+--------+
```

**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 [40]:

```
#Create a new datafram HV_Ratio=Hight Price/ Volumn per day
HV_Ration_df=df.withColumn('HV Ratio',df['high']/df['Volume'])
HV_Ration_df.select('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 [43]:
```
# Didn't need to really do this much indexing
# Could have just shown the entire row
df.orderBy(df["High"].desc()).head(1)[0][0]
```

Out[43]:
```
datetime.datetime(2015, 1, 13, 0, 0)
```

**What is the mean of the Close column?**

In [51]:
```
#the avg for Close column
from pyspark.sql.functions import mean
df.select(mean('Close')).show()
```

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

**What is the max and min of the Volume column?**

In [52]:
```
# the max and min of the Volume column
from pyspark.sql.functions import max,min
df.select(max("Volume"),min("Volume")).show()
```

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

**How many days was the Close lower than 60 dollars?**

In [56]:
```
#How many days was the Close lower than 60 dollars
df.filter(df['Close'] < 60).count()
```

Out[56]:
```
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 [58]:
```
#What percentage of the time was the High greater than 80 dollars
(df.filter(df["High"]>80).count()/df.count())*100
```

Out[58]:
```
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.DataFrameStatF**

In [59]:
```
# the Pearson correlation between High and Volume
from pyspark.sql.functions import corr
df.select(corr("High","Volume")).show()
```

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

**What is the max High per year?**

In [66]:
```
#only get year from column Date
from pyspark.sql.functions import year
year_df = df.withColumn("Year",year(df["Date"]))
```

In [74]:
```
#the max High per year
max_year_df=year_df.groupBy('Year').max()
max_year_df.select('Year','max(High)').show()
```

```
+----+---------+
|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 [75]:
```
# create a new data column on get month from date stamp
from pyspark.sql.functions import month
month_df = df.withColumn("Month",month("Date"))
```

In [76]:
```
#show average of every month
month_avgs_df = month_df.select("Month","Close").groupBy("Month").mean()
month_avgs_df.select("Month","avg(Close)").orderBy('Month').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|
+-----+-----------------+
```