Spark DataFrames Project Exercise

August 12, 2025

1 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

[2]:
```python
import findspark
findspark.init('/home/ian/spark-4.0.0-bin-hadoop3')
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('PROJECT').getOrCreate()
```

```
25/08/12 17:56:42 WARN SparkSession: Using an existing Spark session; only
runtime SQL configurations will take effect.
```

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

[5]:
```python
walmart_df = spark.read.csv('walmart_stock.csv', inferSchema= True, header=True)
```

What are the column names?

[6]:
```python
walmart_df.columns
```

[6]:
```
['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close']
```

What does the Schema look like?

[7]:
```python
walmart_df.printSchema()
```

```
root
 |-- Date: date (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.

[8]:
```
walmart_df.head(5)
```

[8]:
```
[Row(Date=datetime.date(2012, 1, 3), Open=59.970001, High=61.060001,
Low=59.869999, Close=60.330002, Volume=12668800, Adj Close=52.619234999999996),
 Row(Date=datetime.date(2012, 1, 4), Open=60.209998999999996, High=60.349998,
Low=59.470001, Close=59.709998999999996, Volume=9593300, Adj Close=52.078475),
 Row(Date=datetime.date(2012, 1, 5), Open=59.349998, High=59.619999,
Low=58.369999, Close=59.419998, Volume=12768200, Adj Close=51.825539),
 Row(Date=datetime.date(2012, 1, 6), Open=59.419998, High=59.450001,
Low=58.869999, Close=59.0, Volume=8069400, Adj Close=51.45922),
 Row(Date=datetime.date(2012, 1, 9), Open=59.029999, High=59.549999,
Low=58.919998, Close=59.18, Volume=6679300, Adj Close=51.616215000000004)]
```

Use describe() to learn about the DataFrame.

[11]:
```
walmart_df.describe().show()
```

```
25/08/12 17:59:28 WARN SparkStringUtils: Truncated the string representation of
a plan since it was too large. This behavior can be adjusted by setting
'spark.sql.debug.maxToStringFields'.

+-------+------------------+-----------------+-----------------+----------------
-+-----------------+-----------------+
|summary|              Open|             High|              Low|
Close|           Volume|        Adj Close|
+-------+------------------+-----------------+-----------------+----------------
-+-----------------+-----------------+
|  count|              1258|             1258|             1258|
1258|             1258|             1258|
|   mean| 72.35785375357709|72.83938807631165|
71.9186009594594|72.38844998012726|8222093.481717011|67.23883848728146|
| stddev|
6.76809024470826|6.768186808159218|6.744075756255496|6.756859163732991|
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|
+-------+------------------+-----------------+-----------------+----------------
-+-----------------+-----------------+
```

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

[12]:
```
stats_df = walmart_df.describe()
```

[24]:
```
stats_df.printSchema()

## convert to numeric
from pyspark.sql.types import DoubleType
cols_to_change = stats_df.columns[1:]

for col in cols_to_change:
    stats_df = stats_df.withColumn(col, stats_df[col].cast(DoubleType()))
```

```
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)
```

[26]:
```
## check types now
stats_df.printSchema()
```

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

[28]:
```
from pyspark.sql.functions import format_number, mean

for col in cols_to_change:
    stats_df = stats_df.withColumn(col, format_number(col, 2))

stats_df.show()
```

```
+-------+--------+--------+--------+--------+-------------+---------+
|summary|    Open|    High|     Low|   Close|       Volume|Adj Close|
+-------+--------+--------+--------+--------+-------------+---------+
|  count|1,258.00|1,258.00|1,258.00|1,258.00|     1,258.00| 1,258.00|
|   mean|   72.36|   72.84|   71.92|   72.39| 8,222,093.48|    67.24|
| stddev|    6.77|    6.77|    6.74|    6.76| 4,519,780.84|     6.72|
|    min|   56.39|   57.06|   56.30|   56.42| 2,094,900.00|    50.36|
|    max|   90.80|   90.97|   89.25|   90.47|80,898,100.00|    84.91|
+-------+--------+--------+--------+--------+-------------+---------+
```

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.

[32]:
```
walmart_df = walmart_df.withColumn('HV Ratio', walmart_df['High']/
  ↪walmart_df['Volume'])
```

What day had the Peak High in Price?

[42]:
```
max_high = walmart_df.agg({'High':'max'}).collect()[0][0]
walmart_df.filter(walmart_df['High'] == max_high).select('Date').show()
```

```
+----------+
|      Date|
+----------+
|2015-01-13|
+----------+
```

What is the mean of the Close column?

[44]:
```
walmart_df.select(mean('Close')).show()
```

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

What is the max and min of the Volume column?

[51]:
```
from pyspark.sql.functions import min, max

walmart_df.select(min('Volume'), max('Volume')).show()
```

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

How many days was the Close lower than 60 dollars?

[54]:
```
walmart_df.filter(walmart_df['Close'] < 60).show()
```

```
+----------+------------------+------------------+------------------+-----------
-------+--------+------------------+--------------------+
|      Date|              Open|              High|               Low|
Close|  Volume|         Adj Close|            HV Ratio|
+----------+------------------+------------------+------------------+-----------
-------+--------+------------------+--------------------+
|2012-01-04|60.209998999999996|         60.349998|
59.470001|59.709998999999996| 9593300|         52.078475|6.290848613094555E-6|
|2012-01-05|         59.349998|         59.619999|         58.369999|
59.419998|12768200|         51.825539|4.669412994783916E-6|
|2012-01-06|         59.419998|         59.450001|         58.869999|
59.0| 8069400|          51.45922|7.367338463826307E-6|
|2012-01-09|         59.029999|         59.549999|         58.919998|
59.18| 6679300|51.616215000000004|8.915604778943901E-6|
|2012-01-10|             59.43|59.709998999999996|
58.98|59.040001000000004| 6907300|         51.494109|8.644477436914568E-6|
|2012-01-11|         59.060001|         59.529999|59.040001000000004|
59.400002| 6365600|         51.808098|9.351828421515645E-6|
|2012-01-12|59.790001000000004|              60.0|         59.400002|
59.5| 7236400|51.895315999999994| 8.29141562102703E-6|
|2012-01-13|
59.18|59.610001000000004|59.009997999999996|59.540001000000004|
7729300|51.930203999999996|7.712212102001476E-6|
|2012-01-17|         59.869999|60.110001000000004|             59.52|
59.849998| 8500000|         52.200581|7.071764823529412E-6|
|2012-02-22|         59.580002|         59.900002|         58.369999|
58.599998|28630200|         51.110343|2.092196421960028E-6|
|2012-02-23|             58.59|
58.900002|58.209998999999996|58.540001000000004|14880300|
51.058014|3.958253664240641E-6|
|2012-02-24|             58.75|         58.950001|
58.5|58.790001000000004| 9925900|         51.276061|5.939008150394423E-6|
|2012-02-27|         58.700001|
58.779999|58.290001000000004|58.459998999999996|12258800|
50.988237|4.794922749371879…|
|2012-02-28|         58.439999|         59.099998|         58.349998|
58.93|10761900|         51.398167|5.491595164422639E-6|
|2012-02-29|             58.84|         59.330002|         58.720001|
59.080002|11484400|         51.528997|5.166138587997632E-6|
|2012-03-01|59.360001000000004|         59.419998|58.639998999999996|
58.82|16283900|         51.302226| 3.64900288014542E-6|
|2012-03-02|58.990002000000004|         59.279999|
58.799999|59.009997999999996| 9848100|51.467940999999996|6.019435119464668E-6|
|2012-03-05|58.959998999999996|              59.59|             58.75|
59.400002| 9651000|         51.808098|6.174489690187545E-6|
|2012-03-06|59.040001000000004|         59.220001|             58.75|
58.970001| 9057100|         51.433056|6.538516854180698E-6|
|2012-03-
```

```
07|59.110001000000004|59.860001000000004|59.110001000000004|59.860001000000004|1
4916900|         52.209305|4.012898189302067…|
+----------+------------------+------------------+------------------+-----------
-------+--------+------------------+--------------------+
only showing top 20 rows
```

[55]:

```python
walmart_df.filter(walmart_df['Close'] < 60).count()
```

[55]:

```
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)

[57]:

```python
total_days = walmart_df.count()
total_days
```

[57]:

```
1258
```

[58]:

```python
days_more_than_80 = walmart_df.filter(walmart_df['High'] > 80).count()
days_more_than_80
```

[58]:

```
115
```

[61]:

```python
print(f'{(days_more_than_80/total_days)*100:.2f}%')
```

```
9.14%
```

What is the Pearson correlation between High and Volume?

Hint

[62]:

```python
from pyspark.sql.functions import corr

walmart_df.select(
    corr("High", "Volume")).show()
```

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

[ ]:

```python
## it seems that the more the volume traded, the lower the high price
```

What is the max High per year?

[71]:

```python
from pyspark.sql.functions import year, max
```

```
## generate date column
walmart_df = walmart_df.withColumn('year', year('Date'))

walmart_df.groupby('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.

[72]:

```
from pyspark.sql.functions import month

## generate month column
walmart_df = walmart_df.withColumn('month', month('Date'))

walmart_df.groupby('month').mean('Close').show()
```

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

2 Great Job!

2.1 OTHER INTERESTING INSIGHTS

2.1.1 1. 7 day moving average

[73]:

```
from pyspark.sql.window import Window
from pyspark.sql.functions import avg, col
```

[82]:

```
windowSpec = Window.orderBy('Date').rowsBetween(-6,0)
```

[83]:

```
walmart_with_avg = walmart_df.withColumn(
    "7_day_moving_avg", avg(col("Close")).over(windowSpec)
)
```

[88]:

```
walmart_with_avg.select("Date", "Close", "7_day_moving_avg").show()
```

```
25/08/12 18:34:56 WARN WindowExec: No Partition Defined for Window operation!
Moving all data to a single partition, this can cause serious performance
degradation.
25/08/12 18:34:56 WARN WindowExec: No Partition Defined for Window operation!
Moving all data to a single partition, this can cause serious performance
degradation.
25/08/12 18:34:56 WARN WindowExec: No Partition Defined for Window operation!
Moving all data to a single partition, this can cause serious performance
degradation.
25/08/12 18:34:56 WARN WindowExec: No Partition Defined for Window operation!
Moving all data to a single partition, this can cause serious performance
degradation.
25/08/12 18:34:56 WARN WindowExec: No Partition Defined for Window operation!
Moving all data to a single partition, this can cause serious performance
degradation.
```

```
+----------+-----------------+------------------+
|      Date|            Close|  7_day_moving_avg|
+----------+-----------------+------------------+
|2012-01-03|        60.330002|         60.330002|
|2012-01-04|59.70999899999996|60.020000499999995|
|2012-01-05|        59.419998| 59.81999966666666|
|2012-01-06|             59.0|59.614999749999996|
|2012-01-09|            59.18|59.527999799999996|
|2012-01-10|59.040001000000004|59.446666666666665|
|2012-01-11|        59.400002| 59.44000028571429|
|2012-01-12|             59.5| 59.32142857142857|
|2012-01-13|59.540001000000004| 59.29714314285714|
|2012-01-17|        59.849998| 59.35857171428571|
|2012-01-18|60.009997999999996| 59.50285714285714|
|2012-01-19|60.610001000000004| 59.70714300000001|
|2012-01-20|61.009997999999996| 59.98857114285714|
|2012-01-23|             60.91| 60.20428514285714|
|2012-01-24|61.389998999999996|60.474284999999995|
|2012-01-25|         61.470001| 60.74999928571429|
|2012-01-26|         60.970001|60.909999714285725|
|2012-01-27|60.709998999999996| 61.00999985714286|
|2012-01-30|         61.299999|         61.108571|
|2012-01-31|61.360001000000004| 61.15857142857143|
+----------+------------------+------------------+
only showing top 20 rows
```

2.1.2 2. Year over year growth

[89]:

```python
from pyspark.sql.functions import year, lag, col
from pyspark.sql.window import Window

# Add a 'year' column to the DataFrame
walmart_with_year = walmart_df.withColumn('year', year('Date'))

# Get the last closing price of each year
last_close_of_year = walmart_with_year.groupBy('year').agg({'Close': 'last'}).
 ↪withColumnRenamed('last(Close)', 'Close')

# Use a window function to get the previous year's close price
windowSpec_year = Window.orderBy('year')
last_close_of_year_with_prev = last_close_of_year.withColumn('prev_year_close',␣
 ↪lag(col('Close'), 1).over(windowSpec_year))

# Calculate the year-over-year growth percentage
yoy_growth = last_close_of_year_with_prev.withColumn(
    'yoy_growth_percent',
    ((col('Close') - col('prev_year_close')) / col('prev_year_close')) * 100
)

# Show the results
yoy_growth.select('year', 'Close', 'yoy_growth_percent').show()
```

```
25/08/12 18:37:55 WARN WindowExec: No Partition Defined for Window operation!
Moving all data to a single partition, this can cause serious performance
degradation.
25/08/12 18:37:55 WARN WindowExec: No Partition Defined for Window operation!
Moving all data to a single partition, this can cause serious performance
degradation.
25/08/12 18:37:55 WARN WindowExec: No Partition Defined for Window operation!
Moving all data to a single partition, this can cause serious performance
degradation.
25/08/12 18:37:55 WARN WindowExec: No Partition Defined for Window operation!
Moving all data to a single partition, this can cause serious performance
degradation.
```

```
25/08/12 18:37:55 WARN WindowExec: No Partition Defined for Window operation!
Moving all data to a single partition, this can cause serious performance
degradation.
```

```
+----+---------+------------------+
|year|    Close|yoy_growth_percent|
+----+---------+------------------+
|2012|68.230003|              NULL|
|2013|78.690002|15.330497640458862|
|2014|85.879997| 9.137113759382032|
|2015|61.299999|-28.62133076227285|
|2016|69.120003|12.756939849215982|
+----+---------+------------------+
```

```
25/08/12 18:37:55 WARN WindowExec: No Partition Defined for Window operation!
Moving all data to a single partition, this can cause serious performance
degradation.
25/08/12 18:37:55 WARN WindowExec: No Partition Defined for Window operation!
Moving all data to a single partition, this can cause serious performance
degradation.
```

[ ]: