Data And Problem Statement : Stock Data Analysis

My Role : Trader working in an angel broking company, analyzing the market trends for 4 stocks : MSFT , GOOGL , ADBE , FB

Getting the data for analysis :

A ***Python script*** to get the stocks information from NYSE is placed at a location in my computer.

Once the NYSE opens, the script runs to fetch the data relating to the stocks every minute inside a folder. I run the python script from command prompt.

The files created are ***JSON*** type and are stored at a location in my computer.

The stock information present in JSON contains fields - ***symbol , timestamp , openPrice , closePrice , highestPrice , lowestPrice*** and ***volume***.

Analysis 1:

Calculate the simple moving average closing price of the four stocks in a 5-minute sliding window for the last 10 minutes

Input -> Input will be JSON files generated by Python script. Data used from JSON is - symbol and closePrice of stock

Logic -> Since we are dealing with sliding windows mechanism, average closePrice needs to be maintained such that for any 5 minute window , with every incoming data - its average is added and for every outgoing data, its average is deducted.

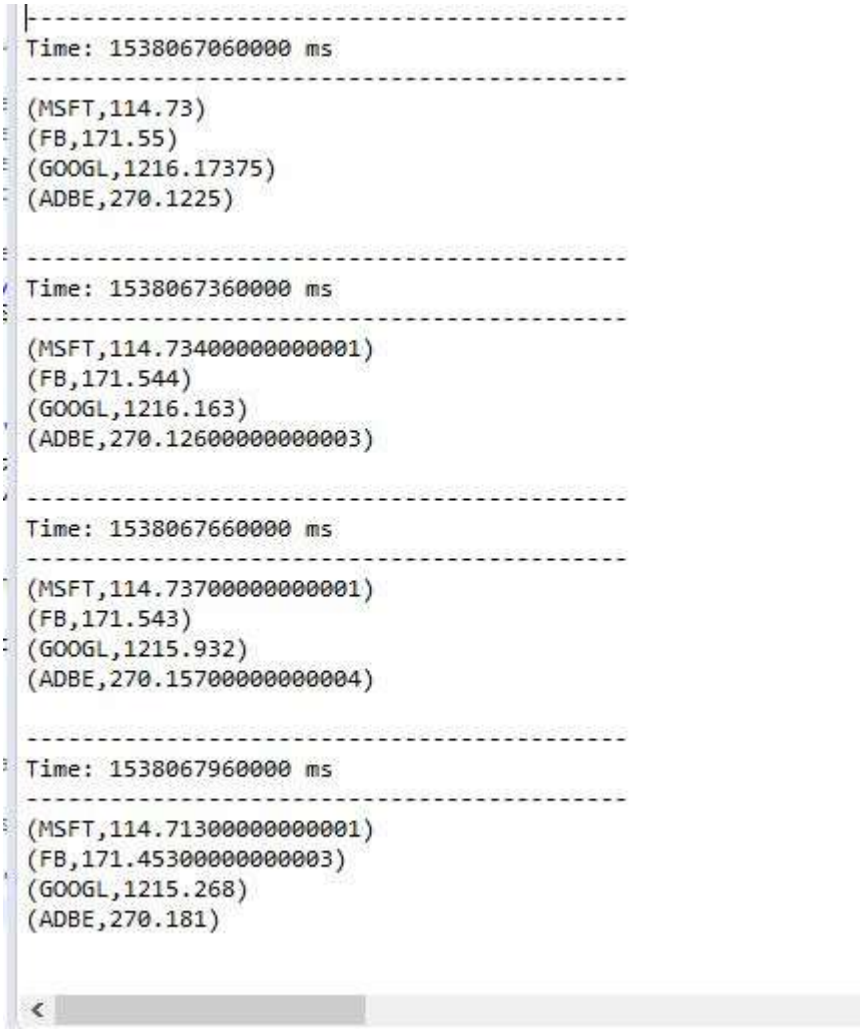
I have used reduce function, also called ***summary function*** to keep track of incoming calculation and invreduce function, also called ***inverse function*** to keep track of outgoing calculation.

These methods are called inside ***reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval)*** transformation. In this transformation, my window length and sliding interval is 10 minutes and 5 minutes respectively.

Using this transformation, the reduce value of each window is calculated incrementally using the reduce values of the previous window. This is done by reducing the new data that enters the sliding window, and "inverse reducing" the old data that leaves the window. An example would be that of "adding" and "subtracting" counts of keys as the window slides. In essence , this is going to give me moving averages for closePrice of all 4 stocks every 5 minutes. When results are emitted, they are of the form – ***(Stock , AverageClosingPrice)***

For a period of 30 minutes , below is the output generated on console every 5 minutes:

*****There were 6 outputs generated but since this is a screenshot, this is the maximum content I could capture from console in 1 single screen grab.***



Analysis 2:

Find the stock out of the four stocks giving maximum profit (average closing price - average opening price) in a 5-minute sliding window for the last 10 minutes

Input -> Input will be JSON files generated by Python script. Data used from JSON is - symbol , closePrice and openPrice of stock.

Logic -> Since we are dealing with sliding windows mechanism, Profit (average closing price - average opening price) needs to be maintained such that for any 5 minute window , with every incoming data - its profit is added and for every outgoing data, its profit is deducted.

I have used reduce function, also called ***summary function*** to keep track of incoming calculation and invreduce function, also called ***inverse function*** to keep track of outgoing calculation.

These methods are called inside ***reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval)*** transformation. In this transformation, my window length and sliding interval is 10 minutes and 5 minutes respectively.

Using this transformation, the reduce value of each window is calculated incrementally using the reduce values of the previous window. This is done by reducing the new data that enters the sliding window, and "inverse reducing" the old data that leaves the window. An example would be that of "adding" and "subtracting" counts of keys as the window slides.

When results are emitted , they are of the form – ***(Stock,Profit)*** which are then emitted out after ***sorting them in descending order of profit***

PLEASE NOTE: While I could have emitted only the highest profitable stock name, but I as a trader want to see how much are the other stocks lagging behind and hence , I choose to emit all 4 sorted in descending order of profit.

For a period of 30 minutes , below is the output generated on console every 5 minutes:

*****There were 6 outputs generated but since this is a screenshot, this is the maximum content I could capture from console in 1 single screen grab.***

```
-------------------------------------------
Time: 1538068440000 ms
-------------------------------------------
(0.033999999999991815,ADBE)
(0.01875000000003979,FB)
(0.007999999999981355,MSFT)
(-0.03499999999985448,GOOGL)

-------------------------------------------
Time: 1538068740000 ms
-------------------------------------------
(0.028333333333364408,ADBE)
(0.012000000000028876,FB)
(0.006666666666660603,MSFT)
(-0.027999999999792635,GOOGL)

-------------------------------------------
Time: 1538069040000 ms
-------------------------------------------
(0.02250000000003638,ADBE)
(0.02200000000001978,FB)
(0.007499999999993179,MSFT)
(-0.13499999999976353,GOOGL)

-------------------------------------------
Time: 1538069340000 ms
-------------------------------------------
(0.025000000000005684,FB)
(0.018333333333430346,ADBE)
(0.003333333333303017,MSFT)
(-0.11799999999971078,GOOGL)
```

Analysis 3:

Find out the Relative Strength Index or RSI of the four stocks in a 5-minute sliding window for the last 10 minutes.

Input -> Input will be JSON files generated by Python script. Data used from JSON is - symbol and closePrice

Logic -> Since we are dealing with sliding windows mechanism, RSI will be calculated based on the formulae provided for different periods. Here, ***we start calculating RSI from 11th period*** – which means after 10 periods of data generation are over. Since each period in our case is 1 minute , we see the results emitted for RSI only after 3rd sliding window (which means 11 and above minutes).

I have made use of ***window*** transformation to specify window length and sliding interval, which is 10 minutes and 5 minutes respectively in my case.

I have made use of ***updateStateByKey*** transformation because the state needs to be carried between the sliding windows. This transformation takes 1 parameter – update function, which I have implemented using the logic as below. Also, I maintain a state object which helps me store the state and pass it to subsequent periods.

1. Take close price difference of 2 periods (current period and one before)
2. If this value is positive , it means it is a gain, add it to gainSum and lossSum = 0
3. If this value is negative , it means it is a loss, add the absolute value to lossSum. gainSum will be 0
4. Until count = 10, I just keep on incrementing my lossSum or gainSum based on loss or gain respectively between 2 close prices.
5. At count = 11, get the below:

   First Average Gain = Sum of Gains over the past 10 periods / 10.
   First Average Loss = Sum of Losses over the past 10 periods / 10.
   RS = Average Gain / Average Loss
   RSI = 100 – (100 / (1 + RS))
   To simplify the calculation, if Average Gain = 0 , RSI = 0
   If Average Loss = 0 , RSI = 100
6. Coming back to count > 11,
   Average Gain = [(previous Average Gain) x 9 + current Gain] / 10.
   Average Loss = [(previous Average Loss) x 9 + current Loss] / 10
   RS = Average Gain / Average Loss
   RSI = 100 – (100 / (1 + RS))

   If you see, we always need to carry previous Average Gain and previous Average Loss, hence I used the stateful transformation for the same

When results are emitted , they are of the form – ***(Stock , RSI)***.

For a period of 30 minutes , below is the output generated on console every 5 minutes:

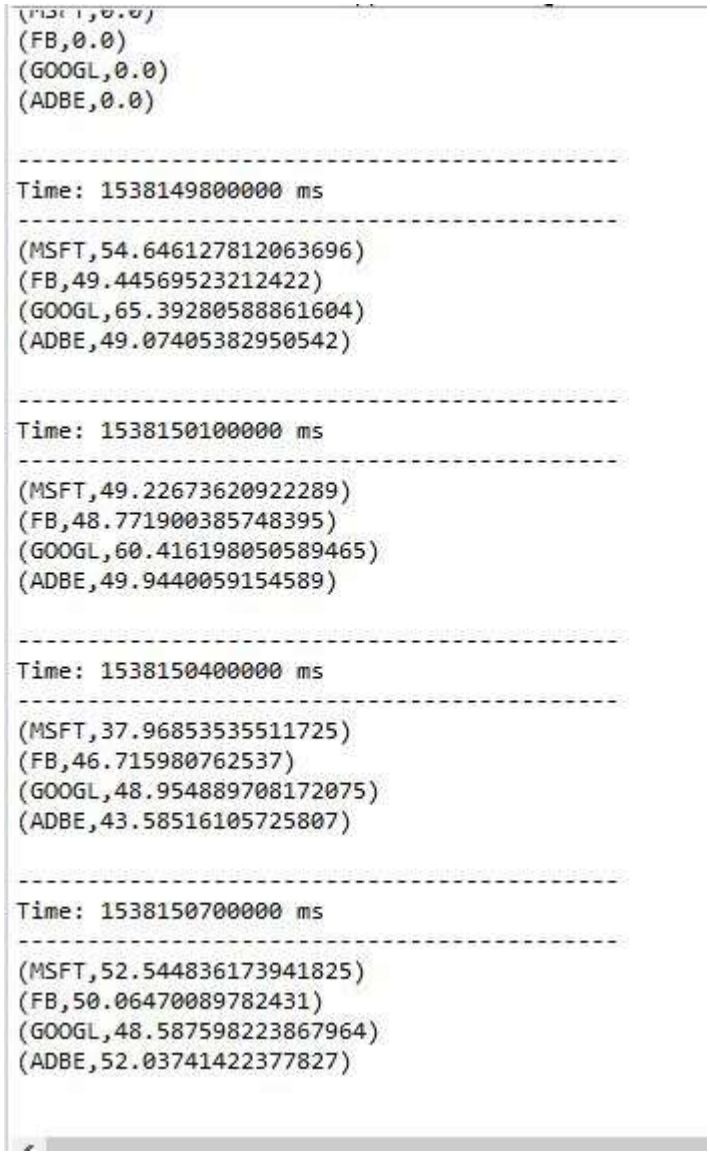
```
-------------------------------------------
Time: 1538149200000 ms
-------------------------------------------
(MSFT,0.0)
(FB,0.0)
(GOOGL,0.0)
(ADBE,0.0)

-------------------------------------------
Time: 1538149500000 ms
-------------------------------------------
(MSFT,0.0)
(FB,0.0)
(GOOGL,0.0)
(ADBE,0.0)

-------------------------------------------
Time: 1538149800000 ms
-------------------------------------------
(MSFT,54.646127812063696)
(FB,49.44569523212422)
(GOOGL,65.39280588861604)
(ADBE,49.07405382950542)

-------------------------------------------
Time: 1538150100000 ms
-------------------------------------------
(MSFT,49.22673620922289)
(FB,48.771900385748395)
(GOOGL,60.416198050589465)
(ADBE,49.9440059154589)

-------------------------------------------
Time: 1538150400000 ms
-------------------------------------------
(MSFT,37.96853535511725)
(FB,46.715980762537)
```



Analysis 4:

Calculate the trading volume of the four stocks every 10 minutes

Input -> Input will be JSON files generated by Python script. Data used from JSON is - symbol , volume of stock.

Logic -> Since we are dealing with sliding windows mechanism, volume needs to be maintained such that for any 10 minute window , with every incoming data - its volume is added and for every outgoing data, its volume is deducted.

I have used reduce function, also called ***summary function*** to keep track of incoming calculation and invreduce function, also called ***inverse function*** to keep track of outgoing calculation.

These methods are called inside ***reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval)*** transformation.

Using this transformation, the reduce value of each window is calculated incrementally using the reduce values of the previous window. This is done by reducing the new data that enters the sliding

window, and “inverse reducing” the old data that leaves the window. An example would be that of “adding” and “subtracting” counts of keys as the window slides.

PLEASE NOTE: In this transformation, my window length and sliding interval is both 10 minutes.

When results are emitted , they are of the form – (***Stock , Volume***) which are then emitted out after **sorting them in descending order of volume**.

Since, we have a 10 min long sliding window , hence I get 3 outputs at 10 min interval each.

```
-------------------------------------------
Time: 1538064900000 ms
-------------------------------------------
(196765,FB)
(81081,MSFT)
(12979,ADBE)
(9796,GOOGL)

-------------------------------------------
Time: 1538065500000 ms
-------------------------------------------
(206318,FB)
(51526,MSFT)
(22479,ADBE)
(20595,GOOGL)

-------------------------------------------
Time: 1538066100000 ms
-------------------------------------------
(192426,FB)
(65235,MSFT)
(21278,ADBE)
(14559,GOOGL)
```

How is the driver class bringing all solutions together?

The driver class is named – *StockAnalysis*.

This has the main method which calls Spark Streaming Context. ***SparkStreamingContext()*** method in turn sets up the context with ***Sparkconf*** and batch ***duration*** for the incoming D-streams (internally RDDs).

It reads the JSON coming being generated by Python script and parses it into D-Streams.

Each Analysis (Ques 1 , Ques 2 , Ques 3 , Ques 4) have individual classes whose objects are created in this main class. The createStreamingContext() method in each of these classes are invoked by their objects one by one and receive ***JavaDStreams*** as input. It is these methods in each class which are actually running the code logic.

How to run the solution?

1. Download the SparkApplication jar at some location in your computer. This is **location1**

2. Open Command prompt and navigate to this location (**location1**)
3. Ensure that python script is running and storing the JSON files generated every minute as some location in your computer. This is **location2**
4. On command prompt, type the following command:

   java -cp SparkApplication.jar com.deepika.sparkproject.StockAnalysis "<**location2**>"
5. Run the program for at least 30 minutes. You will start seeing the output generated at **location1** for each question at intervals of 5 min. [For analysis 4, this interval is 10 min]