23B3307-E10-2

November 19, 2024

0.1 E10-2

Nirav Bhattad (23B3307)

This Notebook illustrates the use of SPARK Dataframe functions to process nse-data.csv

- Review Part-1 to understand the code by referring to SPARK documentation.
- Add your comment to each cell, to explain its purpose
- Add code / create additional cells for debugging purpose, and comment them too
- Write SPARK code to solve the problem stated in Part-2 (do not use the createTempView function in your solution!)

Submission - Create and upload a PDF of this Notebook. - BEFORE CONVERTING TO PDF ENSURE THAT YOU REMOVE / TRIM LENGTHY DEBUG OUTPUTS . - Short debug outputs of up to 5 lines are acceptable.

0.2 Part 1

[1]:
```
import findspark # here we are importing findspark module to locate the spark␣
↪in the system
findspark.init() # here we are initializing the spark
```

[2]:
```
import pyspark # here we are importing pyspark module
from pyspark.sql.types import * # here we are importing all the classes from␣
↪the module
from pyspark.sql import functions as F # here we are importing functions from␣
↪the module
```

[ ]:
```
sc = pyspark.SparkContext(appName="E10-2") # here we are creating a spark␣
↪context
```

[4]:
```
ss = pyspark.sql.SparkSession(sc) # here we are creating a spark session
```

[5]:
```
dfr = ss.read # here we are reading the data from the file
```

[6]:
```
schemaStruct = StructType()
schemaStruct.add("SYMBOL", StringType(), True)
schemaStruct.add("SERIES", StringType(), True)
```

```
schemaStruct.add("OPEN", DoubleType(), True)
schemaStruct.add("HIGH", DoubleType(), True)
schemaStruct.add("LOW", DoubleType(), True)
schemaStruct.add("CLOSE", DoubleType(), True)
schemaStruct.add("LAST", DoubleType(), True)
schemaStruct.add("PREVCLOSE", DoubleType(), True)
schemaStruct.add("TOTTRDQTY", LongType(), True)
schemaStruct.add("TOTTRDVAL", DoubleType(), True)
schemaStruct.add("TIMESTAMP", StringType(), True)
schemaStruct.add("ADDNL", StringType(), True)

# here we are reading the data from the file, and we are providing the schema␣
↪to the data as well
```

[6]: StructType([StructField('SYMBOL', StringType(), True), StructField('SERIES', StringType(), True), StructField('OPEN', DoubleType(), True), StructField('HIGH', DoubleType(), True), StructField('LOW', DoubleType(), True), StructField('CLOSE', DoubleType(), True), StructField('LAST', DoubleType(), True), StructField('PREVCLOSE', DoubleType(), True), StructField('TOTTRDQTY', LongType(), True), StructField('TOTTRDVAL', DoubleType(), True), StructField('TIMESTAMP', StringType(), True), StructField('ADDNL', StringType(), True)])

[7]:
```
df = dfr.csv("./nsedata.csv", schema=schemaStruct, header=True) # here we are␣
↪reading the data from the file
```

0.2.1 Basics : Using SPARK for analysis

[8]:
```
def create_subset_from_df(company_code):
    """
    This function takes a company code as input and returns a subset of the␣
↪dataframe with the columns OPEN, HIGH, LOW, CLOSE and TIMESTAMP for the␣
↪given company code.

    Parameters:
        company_code: str: A string representing the company code.

    Returns:
        df_subset: DataFrame: A subset of the dataframe with the columns OPEN,␣
↪HIGH, LOW, CLOSE and TIMESTAMP for the given company code.
    """
    tcode = company_code.lower()
    df_subset = df.select(\
                    F.col("OPEN").alias("OPEN_" + tcode),\
                    F.col("HIGH").alias("HIGH_"+ tcode),\
                    F.col("LOW").alias("LOW_"+ tcode),\
                    F.col("CLOSE").alias("CLOSE_" + tcode),\
```

```
                    F.col("TIMESTAMP")).\
                    where(F.col("SYMBOL") == company_code)
    return(df_subset)
```

[9]:
```
# Why do we need to use the alias function, above? What happens if we do not␣
 ↪alias / rename the columns?
# Answer: We need to use alias function to rename the columns, because if we do␣
 ↪not rename the columns, then the columns will have the same name as the␣
 ↪original dataframe, and it will be difficult to differentiate between the␣
 ↪columns of the original dataframe and the subset dataframe.
```

[ ]:
```
df_infy = create_subset_from_df("INFY")
df_infy.show(5)
df_infy.describe().show()
```

[ ]:
```
df_tcs = create_subset_from_df("TCS")
df_tcs.show(5)
df_tcs.describe().show()
```

[12]:
```
df_join = df_tcs.join(df_infy,"TIMESTAMP").
 ↪select("TIMESTAMP","CLOSE_tcs","CLOSE_infy")
df_join.show(5)
```

```
24/11/19 20:51:58 WARN GarbageCollectionMetrics: To enable non-built-in garbage
collector(s) List(G1 Concurrent GC), users should configure it(them) to
spark.eventLog.gcMetrics.youngGenerationGarbageCollectors or
spark.eventLog.gcMetrics.oldGenerationGarbageCollectors
[Stage 9:==>                                                    (1 + 19) / 20]

+-----------+---------+----------+
|  TIMESTAMP|CLOSE_tcs|CLOSE_infy|
+-----------+---------+----------+
|02-FEB-2012|   1148.0|    2757.0|
|01-AUG-2013|   1815.4|   2974.65|
|01-DEC-2014|  2692.95|   4349.85|
|01-OCT-2014|   2775.6|    3847.3|
|01-JUL-2013|  1492.35|    2451.0|
+-----------+---------+----------+
only showing top 5 rows
```

[13]:
```
df_join.select(F.abs(df_join["CLOSE_tcs"] - df_join["CLOSE_infy"]).
 ↪alias("PriceDiff")).describe().show()
```

```
+-------+-----------------+
|summary|        PriceDiff|
```

```
+-------+------------------+
|  count|              1025|
|   mean|1163.6446341463422|
| stddev| 366.9897015322771|
|    min|150.95000000000027|
|    max|            1804.9|
+-------+------------------+
```

[14]:
```python
df_join.filter(F.abs(df_join["CLOSE_tcs"] - df_join["CLOSE_infy"]) < 180).show()
```

```
[Stage 24:>                                                         (0 + 20) / 20]

+-----------+---------+----------+
|  TIMESTAMP|CLOSE_tcs|CLOSE_infy|
+-----------+---------+----------+
|10-FEB-2015|  2441.15|    2278.3|
|11-FEB-2015|   2459.9|   2284.85|
|12-FEB-2015|  2462.15|    2311.2|
+-----------+---------+----------+
```

[15]:
```python
from pyspark.sql.functions import col, date_format, to_date

df1 = df.withColumn("TIMESTAMP2", date_format(to_date(col("TIMESTAMP"),␣
 ↪"dd-MMM-yyyy"), "yyyy-MM"))
```

[ ]:
```python
df1.printSchema()
```

[17]:
```python
df1.show(5)
```

```
24/11/19 20:52:02 WARN CSVHeaderChecker: Number of column in CSV header is not
equal to number of fields in the schema:
 Header length: 14, schema size: 12
CSV file: file:///home/nirav24/Desktop/Academia/IITB%20Courses/Sem%203/DS203/Exe
rcise-10/nsedata.csv

+----------+------+------+-------+------+------+------+---------+---------+-----
--------+-----------+-----+----------+
|    SYMBOL|SERIES|  OPEN|   HIGH|   LOW| CLOSE|  LAST|PREVCLOSE|TOTTRDQTY|
TOTTRDVAL|  TIMESTAMP|ADDNL|TIMESTAMP2|
+----------+------+------+-------+------+------+------+---------+---------+-----
--------+-----------+-----+----------+
| 20MICRONS|    EQ| 37.75|  37.75| 36.35| 37.45|  37.3|    37.15|    38638|
1420968.1|01-APR-2011|    0|   2011-04|
|3IINFOTECH|    EQ| 43.75|   45.3| 43.75|  44.9|  44.8|    43.85|
1239690|5.531120435E7|01-APR-2011|    0|   2011-04|
|   3MINDIA|    EQ|3374.0|3439.95|3338.0|3397.5|3400.0|   3364.7|      871|
```

```
2941547.35|01-APR-2011|    0|   2011-04|
|    A2ZMES|    EQ| 281.8| 294.45| 279.8| 289.2| 287.2|    281.3|   140643|
4.02640755E7|01-APR-2011|    0|   2011-04|
|AARTIDRUGS|    EQ| 127.0|  132.0|126.55| 131.3| 130.6|    127.6|     2972|
384468.2|01-APR-2011|    0|   2011-04|
+----------+------+------+-------+------+------+------+---------+---------+-----
--------+-----------+-----+----------+
only showing top 5 rows
```

[18]:
```python
from pyspark.sql import functions as F

df_t1 = df1.groupBy("SYMBOL","TIMESTAMP2").agg(F.min("OPEN"), F.max("OPEN"), F.
 ↪avg("OPEN"),\
                                       F.stddev("OPEN"), F.count("OPEN"))
```

[19]:
```python
df_t1.show(5)
```

```
[Stage 30:=====================>                              (8 + 12) / 20]

+----------+----------+---------+---------+------------------+------------------
+-----------+
|    SYMBOL|TIMESTAMP2|min(OPEN)|max(OPEN)|         avg(OPEN)|
stddev(OPEN)|count(OPEN)|
+----------+----------+---------+---------+------------------+------------------
+-----------+
|  AREVAT&D|   2011-04|   246.15|   292.95|274.73055555555555|
12.60998832495714|         18|
| CHEMPLAST|   2011-04|      6.3|     8.25| 7.172222222222223|
0.55709916273872|         18|
|FIRSTLEASE|   2011-04|     68.3|   106.05| 93.05277777777776|
10.68782254033041|         18|
|    FORTIS|   2011-04|    152.0|
163.4|159.50833333333333|2.7349723087102285|         18|
| GOLDINFRA|   2011-04|    16.85|    20.15|
17.925|0.7857648952379039|         18|
+----------+----------+---------+---------+------------------+------------------
+-----------+
only showing top 5 rows
```

[20]:
```python
df_t2 = df_t1.sort(F.asc("SYMBOL"), F.asc("TIMESTAMP2"))
```

[21]:
```python
df_t2.show(5)
```

```
[Stage 35:===========================>                       (11 + 10) / 21]

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

```
+-----------+
|   SYMBOL|TIMESTAMP2|min(OPEN)|max(OPEN)|         avg(OPEN)|
stddev(OPEN)|count(OPEN)|
+---------+----------+---------+---------+------------------+------------------
+-----------+
|20MICRONS|   2010-08|     51.6|     54.0| 52.81666666666667|0.9266876496425305|
9|
|20MICRONS|   2010-09|     54.9|     64.3| 59.11428571428571| 2.514614426564381|
21|
|20MICRONS|   2010-10|    55.05|     60.0|57.166666666666664|1.3035848009751174|
21|
|20MICRONS|   2010-11|     53.6|    61.75| 55.98809523809524|2.2001650370997607|
21|
|20MICRONS|   2010-12|     38.8|     61.0| 45.66590909090908| 5.796599708606603|
22|
+---------+----------+---------+---------+------------------+------------------
+-----------+
only showing top 5 rows
```

[22]:
```python
# Uncomment the following statement to generate the output, and analyze it
# Write your observations in the next cell

df_t2.write.csv("monthly_stats.csv")
```

[23]:
```python
# Your observation related to monthly_stats
# The monthly_stats.csv file contains the monthly statistics of the stock␣
 ↪market data, and it contains the minimum, maximum, average, standard␣
 ↪deviation and count of the opening price of the stocks for each company for␣
 ↪each month.
# it is sorted in order by the company code and the timestamp2.
```

0.2.2 SPARK based solutions for stock analysis and portfolio management: An Example

0.3 Problem Statement

Based on equity (EQ) data contained in nsedata.csv, you are tasked with the responsibility to identify a set of 10 stocks to invest in based on the following steps:

- You have to process the data for one entire year, and then make investment decisions for the following year. You can shoose 2012 as the past year and make recommendations for 2013.
- Assume that you are doing this analysis on Jan 1, 2013.
- You are required to draw up an initial list of 10 stocks based on the following preliminary analysis:

    - The stocks should be liquid. That is, they should be traded in large volumes almost every day and the trading volume should be high.
    - You have to filter those stocks that have shown maximum overall growth over the past year. The hope is that they will continue to grow in the future.
- Select 5 pairs of stocks from these filtered stocks based on the following further analysis.
    - You should ensure that volatility and negative market movements in the coming year will not adversely affect the total investment, substantially.
    - One way to achieve this involves selecting stock pairs that are negatively correlated, so that if one stock loses value its partner will most likely gain value - thereby reducing the overall impact of fall in stock prices. As all these stocks are high growth stocks, anyway, the expectation is that there also will be overall growth of the portfolio.
    - Purchase 1 unit of each of these stock pairs on the first trading day of the next year (i.e. 2013)
- Once you have selected the 5 pairs and made the above investments, you should further do the following
    - Report the performance of your portfolio as on 31/12/2013 (or the nearest traded date, if 31/12/2013 was a non traded day) in terms of the:
        * Overall growth of your portfolio
        * Report which stocks in your portfolio grew in value, which of them reduced in value, an whether the pairing strategy worked.
        * How did the overall market perform during the same period? This can be assessed as follows:
            · If you had blindly selected 1 stock each of the top 25 highly traded, high growth stocks, what would have been the performance of this portfolio
            · How did the implemented strategy of selecting highly traded, high growth stocks, but in pairs having negative correlation, perform in comparion? Did the strategy work?

[24]:
```
# Here are some suggested steps to solve the problem

# First of all select only EQUITY related data
# Create a dataframe of stocks that have traded in during the year 2012
# Find out the average total traded quantity of each of these stocks
# Identify stocks that high trade volumes: average daily volume ranging between␣
↪5L and 10L
# Find out the price difference in each of these stocks between the 'last␣
↪traded day of 2012' and 'first traded day of 2012'
# Sort the stocks in descending order using traded quantity and price␣
↪difference as the criteria
# Select the top 10 stocks for further analysis
# Create a new dataframe containing pairs of stocks traded on the same day
#   - join the selected stocks by using the criteria that stock names in the␣
↪resulting dataframe are different
# Sort the dataframe in ascending order
# Establish the criteria for selecting the final pairs of stocks, and select␣
↪them
# Calculate your total investment value
```

```
# ... likewise state and complete the rest of the steps
```

[25]:
```
df_2012 = df.filter("SERIES=='EQ'").filter("TIMESTAMP like '%2012'")
```

[ ]:
```
df_2012_avgqty = df_2012.groupBy("SYMBOL").avg("TOTTRDQTY")\
                        .filter(F.col("avg(TOTTRDQTY)") < 10000000)\
                        .filter(F.col("avg(TOTTRDQTY)") > 500000)\
                        .orderBy("avg(TOTTRDQTY)", ascending=False)
df_2012_avgqty.show(10)
```

[27]:
```
top10 = df_2012_avgqty.limit(10)
```

[28]:
```
t1 = top10.select("SYMBOL").rdd.flatMap(lambda x: x).collect()
t2 = df_2012.filter(F.col("SYMBOL").isin(t1))
t3 = t2.select(F.col("SYMBOL").alias("S1"), F.col("CLOSE").alias("Close1"),␣
 ↪"TIMESTAMP")
t4 = t2.select(F.col("SYMBOL").alias("S2"), F.col("CLOSE").alias("Close2"),␣
 ↪"TIMESTAMP")
df_for_corr = t3.join(t4,"TIMESTAMP")
```

[29]:
```
df_for_corr.show(5)
```

```
+-----------+--------+------+------+------+
|  TIMESTAMP|      S1|Close1|    S2|Close2|
+-----------+--------+------+------+------+
|02-FEB-2012|ALOKTEXT|  20.2|  STER| 124.1|
|02-FEB-2012|ALOKTEXT|  20.2|RENUKA| 38.85|
|02-FEB-2012|ALOKTEXT|  20.2|  NHPC| 20.85|
|02-FEB-2012|ALOKTEXT|  20.2|   ITC|199.05|
|02-FEB-2012|ALOKTEXT|  20.2|  IDFC|131.45|
+-----------+--------+------+------+------+
only showing top 5 rows
```

[30]:
```
wrklist = df_for_corr.select("S1","S2").filter("S1 != S2").distinct().collect()
wrklist[0:10]
```

[30]:
```
[Row(S1='IDFC', S2='NHPC'),
 Row(S1='RENUKA', S2='IDFC'),
 Row(S1='ALOKTEXT', S2='ASHOKLEY'),
 Row(S1='STER', S2='ITC'),
 Row(S1='RENUKA', S2='STER'),
 Row(S1='HINDALCO', S2='IDFC'),
 Row(S1='ITC', S2='ALOKTEXT'),
 Row(S1='ALOKTEXT', S2='NHPC'),
 Row(S1='DLF', S2='ALOKTEXT'),
```

```
Row(S1='GMRINFRA', S2='ITC')]
```

[31]:

```
print(len(wrklist))
```

```
90
```

[ ]:

```
# THIS CELL TAKES QUITE SOME TIME TO EXECUTE - BE PATIENT!
tcorr = []
tlen = len(wrklist)
for i in range(tlen):
    s1 = wrklist[i][0]
    s2 = wrklist[i][1]
    corr = df_for_corr.filter((F.col('S1') == s1) & (F.col('S2') == s2)).
  ↪corr("Close1","Close2")
    tcorr.append([s1,s2,corr])
    if((i+1)%10 ==0):
        print(f"Processed: {i+1} of {tlen}", end='')
```

[33]:

```
from pyspark.sql.types import StructType, StructField, StringType, FloatType
from pyspark.sql import Row

schema = StructType([
    StructField("Symbol1", StringType(), True),
    StructField("Symbol2", StringType(), True),
    StructField("Corr", FloatType(), True)
])

rdd = sc.parallelize(tcorr)
df_corr = ss.createDataFrame(rdd.map(lambda x: Row(Symbol1=x[0], Symbol2=x[1],␣
  ↪Corr=float(x[2]))), schema)

df_corr.show(5)
```

```
+--------+--------+----------+
| Symbol1| Symbol2|      Corr|
+--------+--------+----------+
|    IDFC|    NHPC| 0.7452768|
|  RENUKA|    IDFC| 0.2969912|
|ALOKTEXT|ASHOKLEY|0.47407645|
|    STER|     ITC|-0.3065829|
|  RENUKA|    STER| 0.6944879|
+--------+--------+----------+
only showing top 5 rows
```

[34]:

```
df_corr_neg = df_corr.filter(F.col("Corr") <= 0.0).dropDuplicates(["Corr"]).
  ↪orderBy(F.col("Corr").asc())
df_corr_neg.count()
```

[34]: 12

[35]:
```
df_corr_neg.show()
```

```
+--------+--------+-------------+
| Symbol1| Symbol2|         Corr|
+--------+--------+-------------+
|     ITC|ALOKTEXT|  -0.90314275|
|GMRINFRA|     ITC|   -0.7135044|
|    IDFC|ALOKTEXT|   -0.6409445|
|HINDALCO|     ITC|  -0.62534785|
|ALOKTEXT|    NHPC|  -0.33097458|
|     ITC|ASHOKLEY|   -0.3144176|
|    STER|     ITC|   -0.3065829|
|GMRINFRA|    IDFC|  -0.28986531|
|     ITC|  RENUKA|  -0.21256758|
|     DLF|ALOKTEXT|  -0.16802602|
|GMRINFRA|    NHPC| -0.048641354|
|HINDALCO|    IDFC|-0.0068381117|
+--------+--------+-------------+
```

[36]:
```
df_2013 = df.filter("SERIES=='EQ'").filter("TIMESTAMP like '%2013'")

first_day_2013 = (df_2013.select("TIMESTAMP").filter("TIMESTAMP like␣
↪'%JAN-2013'").distinct().orderBy("TIMESTAMP").first())[0]
last_day_2013 = (df_2013.select("TIMESTAMP").filter("TIMESTAMP like␣
↪'%DEC-2013'").distinct().orderBy("TIMESTAMP",ascending=False).first())[0]

print(first_day_2013,last_day_2013)
```

```
[Stage 887:==>                                                  (1 + 19) / 20]
```

```
01-JAN-2013 31-DEC-2013
```

[37]:
```
def get_price_on_day(loc_stock, loc_date):
    loc_price = df_2013.where(F.col("TIMESTAMP")==loc_date).where(F.
↪col("SYMBOL")==loc_stock).select("CLOSE").collect()[0]
    return((loc_price)[0])
```

[38]:
```
# Selected stocks, based on the analysis
# |     ITC|ALOKTEXT|  -0.90314275|
# |GMRINFRA|     ITC|   -0.7135044|
# |    IDFC|ALOKTEXT|   -0.6409445|
# |HINDALCO|     ITC|  -0.62534785|
# |ALOKTEXT|    NHPC|  -0.33097458|
```

```
stock_list = ["ITC","ALOKTEXT","GMRINFRA","IDFC","HINDALCO","NHPC"]
multiplier = [3,3,1,1,1,1]

prices = []
total_profit = 0
total_investment = 0
for the_stock,the_multiplier in zip(stock_list,multiplier):
    first_day_price = get_price_on_day(the_stock,first_day_2013)
    last_day_price  = get_price_on_day(the_stock,last_day_2013)
    diff = (last_day_price - first_day_price)
    total_diff = diff * the_multiplier
    total_profit += total_diff
    total_investment += (first_day_price * the_multiplier)
    prices.append([the_stock,first_day_price,last_day_price,diff,total_diff])
```

[39]:
```
prices
```

[39]:
```
[['ITC', 287.25, 321.85, 34.60000000000002, 103.80000000000007],
 ['ALOKTEXT', 11.35, 8.45, -2.9000000000000004, -8.700000000000001],
 ['GMRINFRA', 20.3, 24.8, 4.5, 4.5],
 ['IDFC', 173.65, 109.6, -64.05000000000001, -64.05000000000001],
 ['HINDALCO', 134.15, 122.6, -11.550000000000011, -11.550000000000011],
 ['NHPC', 25.35, 19.55, -5.800000000000001, -5.800000000000001]]
```

[40]:
```
print(total_investment, total_profit)
```

```
1249.25 18.200000000000042
```

[ ]:
```
top25 = df_2012_avgqty.limit(25)
t1 = top25.select("SYMBOL").rdd.flatMap(lambda x: x).collect()
t2 = df_2013.filter(F.col("SYMBOL").isin(t1))
t2.show(10)
```

[42]:
```
first_day_overall = t2.where(F.col("TIMESTAMP")==first_day_2013).
 ↪select("CLOSE").agg(F.sum("CLOSE")).collect()
last_day_overall = t2.where(F.col("TIMESTAMP")==last_day_2013).select("CLOSE").
 ↪agg(F.sum("CLOSE")).collect()
total_profit_overall = last_day_overall[0][0] - first_day_overall[0][0]
```

[43]:
```
print(f"Amount: {first_day_overall[0][0]} invested on the first trading day of␣
 ↪2013\n\
has a value: {last_day_overall[0][0]} on the last trading day of 2013\n\
The profit/loss is : {total_profit_overall:.2f} corresponding to␣
 ↪{total_profit_overall/first_day_overall[0][0]*100:.2f}%")
```

```
Amount: 5119.3 invested on the first trading day of 2013
has a value: 4226.45 on the last trading day of 2013
```

```
The profit/loss is : -892.85 corresponding to -17.44%
```

Performance of the strategy

- If we had invested in all the top 25 stocks, without implementing the negative correlation strategy, There would have been a loss of 892 on an investment of 5119 (17.5% loss)
- As against that, by implementing the 'select based on negative correlation' strategy, a profit of 18.2 on an investment of 1249 (1.5% profit) has been achieved
- In conclusion, the strategy has definitely prevented portfolio value loss during a bad year. It has, in fact, preserved capital.

0.4 Part 2 : Problem to solve

1. Which of the following is better, if you have 10 Lakhs to invest for a year:
   - identify 5 top performing stocks of the previous year and invest in them, or
   - Spread your investment across a basket of 25 stocks, with investments equally distributed among them
   - Employing strategies like 'negative correlation' to select your stocks
   - What if you use 'positive correlation' instead, carry out analysis to understand the portfolio's performance?
2. Do your analysis over multiple years (2011-2012, 2012-2013, etc.) to make your final recommendations

[ ]:
```
# print the all of the distinct series
df.select("SERIES").distinct().show()
```

[45]:
```
from pyspark.sql import functions as F
from pyspark.sql.types import StructType, StructField, StringType, FloatType

# Step 1: Filter and preprocess data for 2011 and 2012
df_2011 = df.filter("SERIES == 'EQ'").filter("TIMESTAMP like '%2011'")
df_2012 = df.filter("SERIES == 'EQ'").filter("TIMESTAMP like '%2012'")

# Step 2: Calculate yearly performance for 2011
first_day_2011 = (df_2011.select("TIMESTAMP").filter("TIMESTAMP like␣
↪'%JAN-2011'")
                  .distinct().orderBy("TIMESTAMP").first())[0]
last_day_2011 = (df_2011.select("TIMESTAMP").filter("TIMESTAMP like␣
↪'%DEC-2011'")
                 .distinct().orderBy("TIMESTAMP", ascending=False).first())[0]

# Rename CLOSE column in first_prices_2011
first_prices_2011 = df_2011.filter(F.col("TIMESTAMP") == first_day_2011) \
    .select(F.col("SYMBOL"), F.col("CLOSE").alias("CLOSE_x"))

# Rename CLOSE column in last_prices_2011
last_prices_2011 = df_2011.filter(F.col("TIMESTAMP") == last_day_2011) \
    .select(F.col("SYMBOL"), F.col("CLOSE").alias("CLOSE_y"))
```

```python
# Perform the join
performance_2011 = first_prices_2011.join(last_prices_2011, "SYMBOL") \
    .withColumn("Percent_Change", ((F.col("CLOSE_y") - F.col("CLOSE_x")) / F.col("CLOSE_x")) * 100) \
    .select("SYMBOL", "Percent_Change") \
    .orderBy(F.col("Percent_Change").desc())

# Select top 5 performing stocks
top_stocks_2011 = performance_2011.limit(10).select("SYMBOL").rdd.flatMap(lambda x: x).collect()

print("Top 10 performing stocks in 2011:")
print(top_stocks_2011)
performance_2011.show(10)

# Step 3: Simulate investments in 2012
first_day_2012 = (df_2012.select("TIMESTAMP").filter("TIMESTAMP like '%JAN-2012'")
                  .distinct().orderBy("TIMESTAMP").first())[0]
last_day_2012 = (df_2012.select("TIMESTAMP").filter("TIMESTAMP like '%DEC-2012'")
                 .distinct().orderBy("TIMESTAMP", ascending=False).first())[0]

# Function to get price on a specific day
def get_price_on_day(stock, date):
    price = df_2012.filter(F.col("TIMESTAMP") == date).filter(F.col("SYMBOL") == stock).select("CLOSE").collect()
    return price[0][0] if price else None

# Simulate investments
investment_results = []
total_investment = 0
total_profit = 0

for stock in top_stocks_2011:
    first_price = get_price_on_day(stock, first_day_2012)
    last_price = get_price_on_day(stock, last_day_2012)

    if first_price and last_price:
        diff = last_price - first_price
        investment_results.append([stock, first_price, last_price, diff])
        total_investment += first_price
        total_profit += diff

    if(len(investment_results) == 5):
        break
```

```python
# Display results
for result in investment_results:
    print(f"Stock: {result[0]}, First Price: {result[1]:.2f}, Last Price:␣
↪{result[2]:.2f}, Change: {result[3]:.2f}")

print(f"Total Investment: {total_investment:.2f}")
print(f"Total Profit: {total_profit:.2f}")
print(f"Profit Percentage: {total_profit / total_investment * 100:.2f}%")
```

```
Top 10 performing stocks in 2011:
['AVTNPL', 'ALFALAVAL', 'INSECTICID', 'UTVSOF', 'VSTIND', 'BHARATRAS',
'PAGEIND', 'GUJFLUORO', 'IFBAGRO', 'TTKPRESTIG']
```

```
+----------+------------------+
|    SYMBOL|    Percent_Change|
+----------+------------------+
|    AVTNPL|131.03164817098232|
| ALFALAVAL| 79.80894839973875|
|INSECTICID| 76.00891861761426|
|    UTVSOF|  73.5927665987058|
|    VSTIND| 72.42339832869081|
| BHARATRAS| 60.31105990783411|
|   PAGEIND| 57.10582444626744|
| GUJFLUORO| 55.81867388362652|
|   IFBAGRO| 54.09927495817066|
|TTKPRESTIG| 52.88052074461614|
+----------+------------------+
only showing top 10 rows
```

```
Stock: AVTNPL, First Price: 285.55, Last Price: 35.05, Change: -250.50
Stock: INSECTICID, First Price: 394.85, Last Price: 408.65, Change: 13.80
Stock: VSTIND, First Price: 1071.70, Last Price: 1950.05, Change: 878.35
Stock: BHARATRAS, First Price: 142.60, Last Price: 171.55, Change: 28.95
Stock: PAGEIND, First Price: 2400.15, Last Price: 3424.35, Change: 1024.20
Total Investment: 4294.85
Total Profit: 1694.80
Profit Percentage: 39.46%
```

[46]:
```python
from pyspark.sql import functions as F

# Step 1: Filter and preprocess data for 2011 and 2012
df_2011 = df.filter("SERIES == 'EQ'").filter("TIMESTAMP like '%2011'")
```

```python
df_2012 = df.filter("SERIES == 'EQ'").filter("TIMESTAMP like '%2012'")

# Step 2: Calculate yearly performance for 2011
first_day_2011 = (df_2011.select("TIMESTAMP").filter("TIMESTAMP like '%JAN-2011'")
                  .distinct().orderBy("TIMESTAMP").first())[0]
last_day_2011 = (df_2011.select("TIMESTAMP").filter("TIMESTAMP like '%DEC-2011'")
                 .distinct().orderBy("TIMESTAMP", ascending=False).first())[0]

# Rename CLOSE column in first_prices_2011
first_prices_2011 = df_2011.filter(F.col("TIMESTAMP") == first_day_2011) \
    .select(F.col("SYMBOL"), F.col("CLOSE").alias("CLOSE_x"))

# Rename CLOSE column in last_prices_2011
last_prices_2011 = df_2011.filter(F.col("TIMESTAMP") == last_day_2011) \
    .select(F.col("SYMBOL"), F.col("CLOSE").alias("CLOSE_y"))

# Perform the join
performance_2011 = first_prices_2011.join(last_prices_2011, "SYMBOL") \
    .withColumn("Percent_Change", ((F.col("CLOSE_y") - F.col("CLOSE_x")) / F.col("CLOSE_x")) * 100) \
    .select("SYMBOL", "Percent_Change") \
    .orderBy(F.col("Percent_Change").desc())

# Select top 25 performing stocks
top_stocks_2011 = performance_2011.limit(30).select("SYMBOL").rdd.flatMap(lambda x: x).collect()

print("Top 30 performing stocks in 2011:")
print(top_stocks_2011)
performance_2011.show(30)

# Step 3: Simulate investments in 2012
first_day_2012 = (df_2012.select("TIMESTAMP").filter("TIMESTAMP like '%JAN-2012'")
                  .distinct().orderBy("TIMESTAMP").first())[0]
last_day_2012 = (df_2012.select("TIMESTAMP").filter("TIMESTAMP like '%DEC-2012'")
                 .distinct().orderBy("TIMESTAMP", ascending=False).first())[0]

# Function to get price on a specific day
def get_price_on_day(stock, date):
    price = df_2012.filter(F.col("TIMESTAMP") == date).filter(F.col("SYMBOL") == stock).select("CLOSE").collect()
    return price[0][0] if price else None
```

```
# Simulate investments
investment_results = []
total_investment = 0
total_profit = 0

# Total money to invest equally across 25 stocks
total_money = 1000000  # Example total investment amount (e.g., 1 million)
investment_per_stock = total_money / 25  # Equal investment for each of the 25␣
↪stocks

# Loop through each stock in the top 25
for stock in top_stocks_2011:
    first_price = get_price_on_day(stock, first_day_2012)
    last_price = get_price_on_day(stock, last_day_2012)

    if first_price and last_price:
        diff = last_price - first_price
        investment_results.append([stock, first_price, last_price, diff])
        total_investment += investment_per_stock
        total_profit += diff * (investment_per_stock / first_price)

    if len(investment_results) == 25:
        break

# Display results for each stock in the basket
for result in investment_results:
    print(f"Stock: {result[0]}, First Price: {result[1]:.2f}, Last Price:␣
↪{result[2]:.2f}, Change: {result[3]:.2f}")

print(len(investment_results))
print(f"Total Investment: {total_investment:.2f}")
print(f"Total Profit: {total_profit:.2f}")
print(f"Profit Percentage: {total_profit / total_investment * 100:.2f}%")
```

```
Top 30 performing stocks in 2011:
['AVTNPL', 'ALFALAVAL', 'INSECTICID', 'UTVSOF', 'VSTIND', 'BHARATRAS',
'PAGEIND', 'GUJFLUORO', 'IFBAGRO', 'TTKPRESTIG', 'VISASTEEL', 'INEABS',
'TATACOFFEE', 'GITANJALI', 'BATAINDIA', 'SURANAIND', 'BLUEDART', 'GRUH',
'RAJTV', 'GRAVITA', 'AMTEKINDIA', 'REPRO', 'SOLARINDS', 'AJANTPHARM',
'KAJARIACER', 'BRFL', '20MICRONS', 'LAOPALA', 'HINDUNILVR', 'RELGOLD']
```

```
+----------+------------------+
|    SYMBOL|    Percent_Change|
+----------+------------------+
```

```
|    AVTNPL|131.03164817098232|
| ALFALAVAL| 79.80894839973875|
|INSECTICID| 76.00891861761426|
|    UTVSOF|  73.5927665987058|
|    VSTIND| 72.42339832869081|
| BHARATRAS| 60.31105990783411|
|   PAGEIND| 57.10582444626744|
| GUJFLUORO| 55.81867388362652|
|   IFBAGRO| 54.09927495817066|
|TTKPRESTIG| 52.88052074461614|
| VISASTEEL|51.574803149606296|
|    INEABS| 48.62431849879548|
|TATACOFFEE| 48.42865508491119|
| GITANJALI| 43.04337520739512|
| BATAINDIA| 42.08551132555956|
| SURANAIND|42.078011736278924|
|  BLUEDART| 41.31220709663021|
|      GRUH|  39.2198404785643|
|     RAJTV| 39.00000000000001|
|   GRAVITA| 38.42215057841836|
|AMTEKINDIA| 37.87110789283129|
|     REPRO|37.643555933645274|
| SOLARINDS|  37.4910007199424|
|AJANTPHARM|  37.3419176822258|
|KAJARIACER| 35.65629228687414|
|      BRFL| 35.55555555555555|
| 20MICRONS| 33.44051446945338|
|   LAOPALA| 30.16949152542373|
|HINDUNILVR|30.097397413380172|
|   RELGOLD|29.900636942675167|
+----------+------------------+
only showing top 30 rows
```

```
Stock: AVTNPL, First Price: 285.55, Last Price: 35.05, Change: -250.50
Stock: INSECTICID, First Price: 394.85, Last Price: 408.65, Change: 13.80
Stock: VSTIND, First Price: 1071.70, Last Price: 1950.05, Change: 878.35
Stock: BHARATRAS, First Price: 142.60, Last Price: 171.55, Change: 28.95
Stock: PAGEIND, First Price: 2400.15, Last Price: 3424.35, Change: 1024.20
Stock: GUJFLUORO, First Price: 356.05, Last Price: 333.00, Change: -23.05
Stock: IFBAGRO, First Price: 138.40, Last Price: 183.20, Change: 44.80
Stock: TTKPRESTIG, First Price: 2500.90, Last Price: 3379.00, Change: 878.10
Stock: VISASTEEL, First Price: 57.90, Last Price: 47.60, Change: -10.30
Stock: TATACOFFEE, First Price: 761.75, Last Price: 1407.65, Change: 645.90
Stock: GITANJALI, First Price: 312.25, Last Price: 531.65, Change: 219.40
Stock: BATAINDIA, First Price: 529.95, Last Price: 866.95, Change: 337.00
Stock: SURANAIND, First Price: 418.00, Last Price: 139.20, Change: -278.80
```

```
Stock: BLUEDART, First Price: 1590.20, Last Price: 2047.55, Change: 457.35
Stock: GRUH, First Price: 548.45, Last Price: 237.45, Change: -311.00
Stock: RAJTV, First Price: 83.70, Last Price: 195.45, Change: 111.75
Stock: GRAVITA, First Price: 402.20, Last Price: 183.35, Change: -218.85
Stock: AMTEKINDIA, First Price: 97.00, Last Price: 106.05, Change: 9.05
Stock: REPRO, First Price: 163.90, Last Price: 218.95, Change: 55.05
Stock: SOLARINDS, First Price: 762.65, Last Price: 958.45, Change: 195.80
Stock: AJANTPHARM, First Price: 301.75, Last Price: 382.15, Change: 80.40
Stock: KAJARIACER, First Price: 96.60, Last Price: 231.95, Change: 135.35
Stock: BRFL, First Price: 270.95, Last Price: 243.75, Change: -27.20
Stock: 20MICRONS, First Price: 62.50, Last Price: 156.95, Change: 94.45
Stock: LAOPALA, First Price: 96.45, Last Price: 268.85, Change: 172.40
25
Total Investment: 1000000.00
Total Profit: 344879.68
Profit Percentage: 34.49%
```

[50]:

```python
from pyspark.sql import functions as F
from pyspark.sql.types import StructType, StructField, StringType, FloatType
from pyspark.sql import Row

# Step 1: Filter for 2011 data to train the model
df_2011 = df.filter("SERIES == 'EQ'").filter("TIMESTAMP like '%2011'")
df_2011_avgqty = df_2011.groupBy("SYMBOL").avg("TOTTRDQTY")\
                        .filter(F.col("avg(TOTTRDQTY)") < 10000000)\
                        .filter(F.col("avg(TOTTRDQTY)") > 500000)\
                        .orderBy("avg(TOTTRDQTY)", ascending=False)
df_2011_avgqty.show(10)

# Step 2: Select the top 10 stocks for training
top10_2011 = df_2011_avgqty.limit(10)
t1 = top10_2011.select("SYMBOL").rdd.flatMap(lambda x: x).collect()

# Step 3: Filter the data for 2012 testing based on selected stocks
df_2012 = df.filter("SERIES == 'EQ'").filter("TIMESTAMP like '%2012'")
t2 = df_2012.filter(F.col("SYMBOL").isin(t1))

# Step 4: Prepare for correlation calculation (join data for pairwise␣
↪correlations)
t3 = t2.select(F.col("SYMBOL").alias("S1"), F.col("CLOSE").alias("Close1"),␣
↪"TIMESTAMP")
t4 = t2.select(F.col("SYMBOL").alias("S2"), F.col("CLOSE").alias("Close2"),␣
↪"TIMESTAMP")
df_for_corr = t3.join(t4, "TIMESTAMP")
df_for_corr.show(5)

# Step 5: Get distinct pairs for correlation calculation
```

```python
wrklist = df_for_corr.select("S1", "S2").filter("S1 != S2").distinct().collect()

# Step 6: Calculate correlation for each pair
tcorr = []
tlen = len(wrklist)
for i in range(tlen):
    s1 = wrklist[i][0]
    s2 = wrklist[i][1]
    corr = df_for_corr.filter((F.col('S1') == s1) & (F.col('S2') == s2)).corr("Close1", "Close2")
    tcorr.append([s1, s2, corr])
    # if ((i + 1) % 10 == 0):
        # print(f"Processed: {i + 1} of {tlen}", end='')

# Step 7: Create a DataFrame for correlations
schema = StructType([
    StructField("Symbol1", StringType(), True),
    StructField("Symbol2", StringType(), True),
    StructField("Corr", FloatType(), True)
])

rdd = sc.parallelize(tcorr)
df_corr = ss.createDataFrame(rdd.map(lambda x: Row(Symbol1=x[0], Symbol2=x[1], Corr=float(x[2]))), schema)

df_corr.show(5)

# Step 8: Filter negative correlations
df_corr_neg = df_corr.filter(F.col("Corr") <= 0.0).dropDuplicates(["Corr"]).orderBy(F.col("Corr").asc())
df_corr_neg.count()

df_corr_neg.show()

# Step 9: Get first and last days of 2012 for selected stocks (testing period)
df_2012_first_day = df_2012.select("TIMESTAMP").filter("TIMESTAMP like '%JAN-2012'").distinct().orderBy("TIMESTAMP").first()[0]
df_2012_last_day = df_2012.select("TIMESTAMP").filter("TIMESTAMP like '%DEC-2012'").distinct().orderBy("TIMESTAMP", ascending=False).first()[0]

print(f"First day of 2012: {df_2012_first_day}, Last day of 2012: {df_2012_last_day}")

# Step 10: Function to get price on a specific day
def get_price_on_day(loc_stock, loc_date):
```

```
    loc_price = df_2012.where(F.col("TIMESTAMP") == loc_date).where(F.
↪col("SYMBOL") == loc_stock).select("CLOSE").collect()[0]
    return (loc_price)[0]
```

```
+----------+-----------------+
|    SYMBOL|   avg(TOTTRDQTY)|
+----------+-----------------+
|  ALOKTEXT|8677144.275303643|
|    GVKPIL|8149384.765182186|
|  HINDALCO|7992351.906882592|
|      RCOM|7713584.137651822|
|    RENUKA|7459392.910931174|
|SHREEASHTA|7339390.076923077|
|       ITC|7325373.246963562|
|      IDFC|7102852.137651822|
|      HDIL|6585712.372469636|
|TATAMOTORS|6425309.267206478|
+----------+-----------------+
only showing top 10 rows
```

```
+-----------+--------+------+----------+------+
|  TIMESTAMP|      S1|Close1|        S2|Close2|
+-----------+--------+------+----------+------+
|02-FEB-2012|ALOKTEXT|  20.2|TATAMOTORS|246.45|
|02-FEB-2012|ALOKTEXT|  20.2|SHREEASHTA|   4.0|
|02-FEB-2012|ALOKTEXT|  20.2|    RENUKA| 38.85|
|02-FEB-2012|ALOKTEXT|  20.2|      RCOM|  96.7|
|02-FEB-2012|ALOKTEXT|  20.2|       ITC|199.05|
+-----------+--------+------+----------+------+
only showing top 5 rows
```

```
+----------+----------+-----------+
|   Symbol1|   Symbol2|       Corr|
+----------+----------+-----------+
|       ITC|SHREEASHTA| -0.5946509|
|    GVKPIL|      IDFC|0.025669474|
|    RENUKA|      IDFC|  0.2969912|
|      RCOM|    GVKPIL|  0.8123094|
|TATAMOTORS|       ITC| 0.25560105|
+----------+----------+-----------+
only showing top 5 rows
```

```
+----------+----------+-------------+
|   Symbol1|   Symbol2|         Corr|
+----------+----------+-------------+
|       ITC|  ALOKTEXT|  -0.90314275|
|       ITC|      RCOM|   -0.6903315|
|      IDFC|  ALOKTEXT|   -0.6409445|
|  HINDALCO|       ITC|  -0.62534785|
|       ITC|SHREEASHTA|   -0.5946509|
|    GVKPIL|       ITC|  -0.50507504|
|  ALOKTEXT|      HDIL|   -0.3032723|
|       ITC|    RENUKA|  -0.21256758|
|SHREEASHTA|      IDFC|  -0.14311746|
|      RCOM|      IDFC|  -0.11569301|
|  ALOKTEXT|TATAMOTORS|  -0.09804849|
|SHREEASHTA|TATAMOTORS|-0.0090005705|
|  HINDALCO|      IDFC|-0.0068381117|
+----------+----------+-------------+
```

```
First day of 2012: 02-JAN-2012, Last day of 2012: 31-DEC-2012
```

[53]:

```
# Step 11: Selected stocks, based on the analysis
stock_list = ["ITC", "SHREEASHTA", "GVKPIL", "IDFC", "RENUKA", "RCOM",␣
 ↪"TATAMOTORS"]
multiplier = [2, 1, 2, 2, 1, 1, 1]

# Step 12: Simulate the investment strategy based on 2012 data (testing period)
prices = []
total_profit = 0
total_investment = 0
for the_stock, the_multiplier in zip(stock_list, multiplier):
    try:
        first_day_price = get_price_on_day(the_stock, df_2012_first_day)
        last_day_price = get_price_on_day(the_stock, df_2012_last_day)
        diff = (last_day_price - first_day_price)
        total_diff = diff * the_multiplier
        total_profit += total_diff
        total_investment += (first_day_price * the_multiplier)
        prices.append([the_stock, first_day_price, last_day_price, diff,␣
 ↪total_diff])
    except:
        pass

# print(prices)
```

```
# Step 13: Display results
print("Investment Summary:")
print(f"Total Investment: {total_investment:.2f}")
print(f"Total Profit: {total_profit:.2f}")
print(f"Profit Percentage: {total_profit / total_investment * 100:.2f}%")

# Display prices for each stock
for result in prices:
    print(f"Stock: {result[0]}, First Price: {result[1]:.2f}, Last Price:␣
↪{result[2]:.2f}, Change: {result[3]:.2f}, Total Profit: {result[4]:.2f}")
```

```
Investment Summary:
Total Investment: 886.80
Total Profit: 474.80
Profit Percentage: 53.54%
Stock: ITC, First Price: 198.65, Last Price: 286.80, Change: 88.15, Total
Profit: 176.30
Stock: GVKPIL, First Price: 12.40, Last Price: 13.55, Change: 1.15, Total
Profit: 2.30
Stock: IDFC, First Price: 91.95, Last Price: 171.30, Change: 79.35, Total
Profit: 158.70
Stock: RENUKA, First Price: 24.75, Last Price: 31.75, Change: 7.00, Total
Profit: 7.00
Stock: RCOM, First Price: 72.10, Last Price: 73.90, Change: 1.80, Total Profit:
1.80
Stock: TATAMOTORS, First Price: 183.95, Last Price: 312.65, Change: 128.70,
Total Profit: 128.70
```

[55]:
```
from pyspark.sql import functions as F
from pyspark.sql.types import StructType, StructField, StringType, FloatType
from pyspark.sql import Row

# Step 1: Filter for 2011 data to train the model
df_2011 = df.filter("SERIES == 'EQ'").filter("TIMESTAMP like '%2011'")
df_2011_avgqty = df_2011.groupBy("SYMBOL").avg("TOTTRDQTY")\
                        .filter(F.col("avg(TOTTRDQTY)") < 10000000)\
                        .filter(F.col("avg(TOTTRDQTY)") > 500000)\
                        .orderBy("avg(TOTTRDQTY)", ascending=False)
df_2011_avgqty.show(10)

# Step 2: Select the top 10 stocks for training
top10_2011 = df_2011_avgqty.limit(10)
t1 = top10_2011.select("SYMBOL").rdd.flatMap(lambda x: x).collect()

# Step 3: Filter the data for 2012 testing based on selected stocks
df_2012 = df.filter("SERIES == 'EQ'").filter("TIMESTAMP like '%2012'")
t2 = df_2012.filter(F.col("SYMBOL").isin(t1))
```

```python
# Step 4: Prepare for correlation calculation (join data for pairwise correlations)
t3 = t2.select(F.col("SYMBOL").alias("S1"), F.col("CLOSE").alias("Close1"), "TIMESTAMP")
t4 = t2.select(F.col("SYMBOL").alias("S2"), F.col("CLOSE").alias("Close2"), "TIMESTAMP")
df_for_corr = t3.join(t4, "TIMESTAMP")
df_for_corr.show(5)

# Step 5: Get distinct pairs for correlation calculation
wrklist = df_for_corr.select("S1", "S2").filter("S1 != S2").distinct().collect()

# Step 6: Calculate correlation for each pair (only positive correlation)
tcorr = []
tlen = len(wrklist)
for i in range(tlen):
    s1 = wrklist[i][0]
    s2 = wrklist[i][1]
    corr = df_for_corr.filter((F.col('S1') == s1) & (F.col('S2') == s2)).corr("Close1", "Close2")
    if corr > 0:  # Only include pairs with positive correlation
        tcorr.append([s1, s2, corr])
    # if ((i + 1) % 10 == 0):
    #     print(f"Processed: {i + 1} of {tlen}", end='')

# Step 7: Create a DataFrame for positive correlations
schema = StructType([
    StructField("Symbol1", StringType(), True),
    StructField("Symbol2", StringType(), True),
    StructField("Corr", FloatType(), True)
])

rdd = sc.parallelize(tcorr)
df_corr = ss.createDataFrame(rdd.map(lambda x: Row(Symbol1=x[0], Symbol2=x[1], Corr=float(x[2]))), schema)

df_corr.show(5)

# Step 8: Filter positive correlations and analyze the portfolio
df_corr_pos = df_corr.filter(F.col("Corr") > 0.0).dropDuplicates(["Corr"]).orderBy(F.col("Corr").desc())
df_corr_pos.count()

df_corr_pos.show()
```

```
# Step 9: Get first and last days of 2012 for selected stocks (testing period)
df_2012_first_day = df_2012.select("TIMESTAMP").filter("TIMESTAMP like␣
↪'%JAN-2012'").distinct().orderBy("TIMESTAMP").first()[0]
df_2012_last_day = df_2012.select("TIMESTAMP").filter("TIMESTAMP like␣
↪'%DEC-2012'").distinct().orderBy("TIMESTAMP", ascending=False).first()[0]

print(f"First day of 2012: {df_2012_first_day}, Last day of 2012:␣
↪{df_2012_last_day}")

# Step 10: Function to get price on a specific day
def get_price_on_day(loc_stock, loc_date):
    loc_price = df_2012.where(F.col("TIMESTAMP") == loc_date).where(F.
↪col("SYMBOL") == loc_stock).select("CLOSE").collect()[0]
    return (loc_price)[0]
```

```
+----------+-----------------+
|    SYMBOL|   avg(TOTTRDQTY)|
+----------+-----------------+
|  ALOKTEXT|8677144.275303643|
|    GVKPIL|8149384.765182186|
|  HINDALCO|7992351.906882592|
|      RCOM|7713584.137651822|
|    RENUKA|7459392.910931174|
|SHREEASHTA|7339390.076923077|
|       ITC|7325373.246963562|
|      IDFC|7102852.137651822|
|      HDIL|6585712.372469636|
|TATAMOTORS|6425309.267206478|
+----------+-----------------+
only showing top 10 rows
```

```
+-----------+--------+------+----------+------+
|  TIMESTAMP|      S1|Close1|        S2|Close2|
+-----------+--------+------+----------+------+
|02-FEB-2012|ALOKTEXT|  20.2|TATAMOTORS|246.45|
|02-FEB-2012|ALOKTEXT|  20.2|SHREEASHTA|   4.0|
|02-FEB-2012|ALOKTEXT|  20.2|    RENUKA| 38.85|
|02-FEB-2012|ALOKTEXT|  20.2|      RCOM|  96.7|
|02-FEB-2012|ALOKTEXT|  20.2|       ITC|199.05|
+-----------+--------+------+----------+------+
only showing top 5 rows
```

```
+----------+---------+-----------+
|   Symbol1|  Symbol2|       Corr|
+----------+---------+-----------+
|    GVKPIL|     IDFC|0.025669474|
|    RENUKA|     IDFC|  0.2969912|
|      RCOM|   GVKPIL|  0.8123094|
|TATAMOTORS|      ITC| 0.25560105|
|      IDFC|TATAMOTORS|  0.4580152|
+----------+---------+-----------+
only showing top 5 rows
```

```
+----------+---------+----------+
|   Symbol1|  Symbol2|      Corr|
+----------+---------+----------+
|  HINDALCO|     RCOM|0.87091833|
|      HDIL|     IDFC| 0.8498548|
|      RCOM|   GVKPIL| 0.8123094|
|    GVKPIL| HINDALCO| 0.7652074|
|       ITC|     IDFC|0.70560426|
|      RCOM| ALOKTEXT|0.69144714|
|    RENUKA| HINDALCO|0.68089527|
|    GVKPIL| ALOKTEXT|0.63710546|
|  ALOKTEXT| HINDALCO| 0.6273346|
|    RENUKA|   GVKPIL| 0.6138341|
|SHREEASHTA|     RCOM| 0.5673514|
|SHREEASHTA| ALOKTEXT|  0.531341|
|    RENUKA|     HDIL|0.52904177|
|      HDIL|TATAMOTORS|0.50435644|
|      RCOM|   RENUKA|0.49194598|
|      IDFC|TATAMOTORS| 0.4580152|
|  HINDALCO|SHREEASHTA|0.45538712|
|    GVKPIL|     HDIL|0.40701145|
|       ITC|     HDIL|0.39094618|
|SHREEASHTA|   GVKPIL|0.35677636|
+----------+---------+----------+
only showing top 20 rows

[Stage 4450:==================================>                     (12 + 8) / 20]

First day of 2012: 02-JAN-2012, Last day of 2012: 31-DEC-2012
```

[56]:

```python
# Step 11: Selected stocks based on the positive correlation analysis
# These are the stocks selected based on their positive correlations
stock_list = ["ITC", "GVKPIL", "IDFC", "RENUKA", "RCOM", "TATAMOTORS"]
```

```python
multiplier = [1, 2, 3, 1, 1, 2]

# Step 12: Simulate the investment strategy based on 2012 data (testing period)
prices = []
total_profit = 0
total_investment = 0
for the_stock, the_multiplier in zip(stock_list, multiplier):
    first_day_price = get_price_on_day(the_stock, df_2012_first_day)
    last_day_price = get_price_on_day(the_stock, df_2012_last_day)
    diff = (last_day_price - first_day_price)
    total_diff = diff * the_multiplier
    total_profit += total_diff
    total_investment += (first_day_price * the_multiplier)
    prices.append([the_stock, first_day_price, last_day_price, diff,␣
 ↪total_diff])

# Step 13: Display results
print("Investment Summary:")
print(f"Total Investment: {total_investment:.2f}")
print(f"Total Profit: {total_profit:.2f}")
print(f"Profit Percentage: {total_profit / total_investment * 100:.2f}%")

# Display prices for each stock
for result in prices:
    print(f"Stock: {result[0]}, First Price: {result[1]:.2f}, Last Price:␣
 ↪{result[2]:.2f}, Change: {result[3]:.2f}, Total Profit: {result[4]:.2f}")
```

```
Investment Summary:
Total Investment: 964.05
Total Profit: 594.70
Profit Percentage: 61.69%
Stock: ITC, First Price: 198.65, Last Price: 286.80, Change: 88.15, Total
Profit: 88.15
Stock: GVKPIL, First Price: 12.40, Last Price: 13.55, Change: 1.15, Total
Profit: 2.30
Stock: IDFC, First Price: 91.95, Last Price: 171.30, Change: 79.35, Total
Profit: 238.05
Stock: RENUKA, First Price: 24.75, Last Price: 31.75, Change: 7.00, Total
Profit: 7.00
Stock: RCOM, First Price: 72.10, Last Price: 73.90, Change: 1.80, Total Profit:
1.80
Stock: TATAMOTORS, First Price: 183.95, Last Price: 312.65, Change: 128.70,
Total Profit: 257.40
```

0.4.1 We find that the highest profit percentage is found when we take the stocks which are positively correlated, and invest in them. This is because when the stocks are positively correlated, they tend to move in the same direction, and hence the risk is higher, but the reward is also higher. This is reflected in the higher profit percentage.

[ ]:

```

```