E10-1

This Notebook illustrates the use of "MAP-REDUCE" to calculate averages from the data contained in nsedata.csv.

Task 1

You are required to review the code (refer to the SPARK document where necessary), and **add comments / markup explaining the code in each cell**. Also explain the role of each cell in the overall context of the solution to the problem (ie. what is the cell trying to achieve in the overall scheme of things). You may create additional code in each cell to generate any debug output that you may need to complete this exercise.

Task 2

You are required to write code to solve the problem stated at the end this Notebook

Submission

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

In [1]:
```
# Where is SPARK installed on the VM? Find and import the SPARK library
import findspark
findspark.init()
```

In [2]:
```
# Import pyspark library to create the SPARK context
import pyspark
from pyspark.sql.types import *
```

In [3]:
```
# First step: create the SPARK context. Without this, SPARK functionality c
sc = pyspark.SparkContext(appName="E10")
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setL
ogLevel(newLevel).
23/10/31 12:20:30 WARN NativeCodeLoader: Unable to load native-hadoop libr
ary for your platform... using builtin-java classes where applicable
```

In [4]:
```
# Using the SparkContext, and its function 'textFile', create an RDD 'rdd1'
rdd1 = sc.textFile("/home/hduser/spark/nsedata.csv")
```

In [7]:
```
# Filter out any lines containing the string "SYMBOL" - the header line
rdd1 = rdd1.filter(lambda x: "SYMBOL" not in x)
```

In [6]:
```
# Transform the RDD elements by splitting the lines into lists of values
rdd2 = rdd1.map(lambda x : x.split(","))
print(rdd2.take(5))
```

```
[Stage 0:>                                                          (0 +
1) / 1]

[['20MICRONS', 'EQ', '37.75', '37.75', '36.35', '37.45', '37.3', '37.15',
'38638', '1420968.1', '01-APR-2011', '0', '0', ''], ['3IINFOTECH', 'EQ',
'43.75', '45.3', '43.75', '44.9', '44.8', '43.85', '1239690', '55311204.3
5', '01-APR-2011', '0', '0', ''], ['3MINDIA', 'EQ', '3374', '3439.95', '33
38', '3397.5', '3400', '3364.7', '871', '2941547.35', '01-APR-2011', '0',
'0', ''], ['A2ZMES', 'EQ', '281.8', '294.45', '279.8', '289.2', '287.2',
'281.3', '140643', '40264075.5', '01-APR-2011', '0', '0', ''], ['AARTIDRUG
S', 'EQ', '127', '132', '126.55', '131.3', '130.6', '127.6', '2972', '3844
68.2', '01-APR-2011', '0', '0', '']]
```

In [7]:
```
# Helper comment!: The goal is to find out the mean of the OPEN prices and
```

In [8]:
```
# Extract the necessary data for open prices and create key-value pairs
rdd_open = rdd2.map(lambda x : (x[0]+"_open",float(x[2])))
print(rdd_open.take(5))
# Extract the necessary data for close prices and create key-value pairs
rdd_close = rdd2.map(lambda x : (x[0]+"_close",float(x[5])))
print(rdd_close.take(5))
```

```
[('20MICRONS_open', 37.75), ('3IINFOTECH_open', 43.75), ('3MINDIA_open', 3
374.0), ('A2ZMES_open', 281.8), ('AARTIDRUGS_open', 127.0)]
[('20MICRONS_close', 37.45), ('3IINFOTECH_close', 44.9), ('3MINDIA_close',
3397.5), ('A2ZMES_close', 289.2), ('AARTIDRUGS_close', 131.3)]
```

In [9]:
```
# Merge the two RDDs into a single RDD
rdd_united = rdd_open.union(rdd_close)
print(rdd_united.take(5))
```

```
[Stage 3:>                                                          (0 +
1) / 1]

[('20MICRONS_open', 37.75), ('3IINFOTECH_open', 43.75), ('3MINDIA_open', 3
374.0), ('A2ZMES_open', 281.8), ('AARTIDRUGS_open', 127.0)]
```

In [10]:
```
# Reduce by key - calculate the sum of open and close prices for each key
reducedByKey = rdd_united.reduceByKey(lambda x,y: x+y)
print(reducedByKey.take(5))
```

```
[Stage 4:=====================================================>     (9 + 1)
/ 10]

[('AARTIIND_open', 158044.85), ('ABGSHIP_open', 387447.4), ('ACKRUTI_ope
n', 85931.59999999999), ('AIAENG_open', 650429.5499999999), ('ALCHEM_ope
n', 122052.3)]
```

In [11]:
```
# Count the occurrences of each symbol in the united RDD
temp1 = rdd_united.map(lambda x: (x[0],1)).countByKey()
countOfEachSymbol = sc.parallelize(temp1.items())
```

In [12]:
```
# Join the reduced data with the count of each symbol
symbol_sum_count = reducedByKey.join(countOfEachSymbol)
```

In [13]:
```
# Calculate the averages for open and close prices for each symbol
averages = symbol_sum_count.map(lambda x : (x[0], x[1][0]/x[1][1]))
```

In [14]:
```
# Sort the averages by key
averagesSorted = averages.sortByKey()
```

In [16]:
```
# Save the sorted averages as a text file
averagesSorted.saveAsTextFile("/home/hduser/spark/average")
```

In [17]:
```
# Stop the SparkSession
sc.stop()
```

Review the output files generated in the above step and copy the first 15 lines of any one of the output files into the cell below for reference. Write your comments on the generated output

```
('20MICRONS_close', 53.004122877930484)
('20MICRONS_open', 53.32489894907032)
('3IINFOTECH_close', 18.038803556992725)
('3IINFOTECH_open', 18.17417138237672)
('3MINDIA_close', 4520.343977364591)
('3MINDIA_open', 4531.084518997574)
('3RDROCK_close', 173.2137755102041)
('3RDROCK_open', 173.18316326530612)
('8KMILES_close', 480.73622047244095)
('8KMILES_open', 481.63858267716535)
('A2ZINFRA_close', 18.609433962264156)
('A2ZINFRA_open', 18.73553459119497)
('A2ZMES_close', 89.69389505549951)
```

```
('A2ZMES_open', 90.46271442986883)
('AANJANEYA_close', 441.84030249110316)
```

Task 2 - Problem Statement

Using the MAP-REDUCE strategy, write SPARK code that will create the average of HIGH prices for all the traded companies, but only for any 3 months of your choice. Create the appropriate (K,V) pairs so that the averages are simultaneously calculated, as in the above example. Create the output files such that the final data is sorted in descending order of the company names.

In [3]:
```
!pip install pyspark
```

```
Defaulting to user installation because normal site-packages is not writea
ble
Collecting pyspark
  Downloading pyspark-3.5.0.tar.gz (316.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 316.9/316.9 MB 2.3 MB/s eta
0:00:0000:0100:01
  Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9.7
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 200.5/200.5 KB 27.3 MB/s eta
0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.0-py2.py3-none-any.whl s
ize=317425364 sha256=8f598b0c50028f09e6c07fb2c095ef4ae37b766abfa4c1578dfe6
a8ad73962e8
  Stored in directory: /home/hduser/.cache/pip/wheels/41/4e/10/c2cf2467f71
c678cfc8a6b9ac9241e5e44a01940da8fbb17fc
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.7 pyspark-3.5.0
```

In [12]:
```
from datetime import datetime

# Define a function to parse the date string
def parse_date(date_str):
    # Define a dictionary to map month abbreviations to their numeric repre
    months = {
        'JAN': '01',
        'FEB': '02',
        'MAR': '03',
        'APR': '04',
        'MAY': '05',
        'JUN': '06',
        'JUL': '07',
        'AUG': '08',
        'SEP': '09',
        'OCT': '10',
        'NOV': '11',
        'DEC': '12',
    }

    # Split the date string into day, month, and year
    day, month, year = date_str.split('-')

    # Convert the month abbreviation to the numeric representation
    month = months[month]

    # Create a date object
    date = datetime(int(year), int(month), int(day))
    return date

# Transform the RDD elements by splitting the lines into lists of values
rdd2 = rdd1.map(lambda x: x.split(","))

# Extract the necessary data for timestamp, high, and symbol and create key
rdd_data = rdd2.map(lambda x: (x[0], (parse_date(x[10]), float(x[3]))))

# Filter the data for the desired 3-month period (adjust the date range as
start_date = datetime(2011, 4, 1)
end_date = datetime(2011, 6, 30)
rdd_filtered_data = rdd_data.filter(lambda x: start_date <= x[1][0] <= end_

# Reduce and calculate averages
rdd_reduced_data = rdd_filtered_data.combineByKey(
    lambda x: (x[1], 1),
    lambda acc, x: (acc[0] + x[1], acc[1] + 1),
    lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1])
)

# Calculate the average high price
rdd_average_data = rdd_reduced_data.map(lambda x: (x[0], x[1][0] / x[1][1])

# Sort the data in descending order of company names
rdd_sorted_data = rdd_average_data.sortByKey(ascending=False)

# Save the results to an output file
rdd_sorted_data.saveAsTextFile("output_directory_1")

# Print the results
results = rdd_sorted_data.collect()
for result in results:
    print(result)
```

```
('ZYLOG', 416.2943548387097)
('ZYDUSWELL', 595.3201612903225)
('ZUARIAGRO', 666.4225806451612)
('ZODJRDMKJ', 23.062962962962963)
('ZODIACLOTH', 380.6025423728813)
('ZICOM', 41.50967741935484)
('ZENSARTECH', 175.5217741935484)
('ZENITHINFO', 212.9049180327869)
('ZENITHEXPO', 51.82419354838709)
('ZENITHCOMP', 20.669354838709676)
('ZENITHBIR', 8.816129032258065)
('ZEENEWS', 11.748387096774193)
('ZEELEARN', 22.67983870967742)
('ZEEL', 136.76532258064518)
('ZANDUREALT', 2145.2443548387096)
('YESBANK', 306.2596774193549)
('XPROINDIA', 57.38790322580645)
('XLENERGY', 12.312903225806451)
```

In [ ]: