# 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 [ ]:

In [4]:
```
import findspark
findspark.init()
# Here we have imported the library findspark and initialised it.
```

In [5]:
```
import pyspark
from pyspark.sql.types import *
# Here we have imported the PySpark library and then imported all the classes and functions from the types module within sql package.
```

In [6]:
```
sc = pyspark.SparkContext(appName="E10")
# Here we have created a new variable that holds the reference to SparkContext that can be used to perform various tasks.
# Then we have named the application name to "E10".
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/10/31 17:35:43 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

In [7]:
```
rdd1 = sc.textFile("/home/hduser/spark/nsedata.csv")
# Here a RDD is created with the name rdd1 by reading the csv file nsedata
```

In [8]:
```
rdd1 = rdd1.filter(lambda x: "SYMBOL" not in x)
# This line is used to filter out any line that contains the word "SYMBOL".
```

In [9]:
```
rdd2 = rdd1.map(lambda x : x.split(","))
# Here we create a new RDD file using rdd1 where we split every line of rdd1 by a comma.
```

In [10]:
```
# Helper comment!: The goal is to find out the mean of the OPEN prices and the mean of the CLOSE price in one batch of tasks ...
```

In [11]:
```
rdd_open = rdd2.map(lambda x : (x[0]+"_open",float(x[2])))
# This creates a key value pair
#In this case it is done by appending _open to the first element and the value is the third element
rdd_close = rdd2.map(lambda x : (x[0]+"_close",float(x[5])))
#Here we append _close to the first element and the value is the fifth element.
```

In [12]:
```
rdd_united = rdd_open.union(rdd_close)
# It unites the two rdd rdd_open and rdd_close
```

In [13]:
```
reducedByKey = rdd_united.reduceByKey(lambda x,y: x+y)
# Here we have applied the reduceByKey transformation that applies a reduction function to values with the same key.
# Here it is used to sum up the values for each key.
```

In [14]:
```
temp1 = rdd_united.map(lambda x: (x[0],1)).countByKey()
countOfEachSymbol = sc.parallelize(temp1.items())
# Here we map each element x in rdd to a key value pair where the key is the stock symbol and the value is always 1.
# This assigns a count of '1' to each occurence of a stock symbol.
# Then we count the occurences of each key by using countByKey
# This returns the items of the temp1 dictionary as a list of tuples
# sc.parallelise parallises the list of tuples creating an RDD named countOfEachSymbol.
# This RDD contains key value pairs where the key is the stock symbol and the value is the count of occurences of that symbol in the original data.
```

In [15]:
```
symbol_sum_count = reducedByKey.join(countOfEachSymbol)
# This combines two RDDs reducedByKey and countOfEachSymbol based on a common key.
```

In [16]:
```
averages = symbol_sum_count.map(lambda x : (x[0], x[1][0]/x[1][1]))
# This performs a mapping operation to calculate the average value for each stock symbol based on the sum of prices and the count of occrences.
```

In [17]:
```
averagesSorted = averages.sortByKey()
# This sorts the RDD bases on its keys in ascending order i.e, stock symbol in alphabetical order.
```

In [18]:
```
averagesSorted.saveAsTextFile("/home/hduser/spark/averages")
# The content of averagesSorted is saved as text files with each partition of the RDD being saved as a seperate text file in specified directory
```

In [17]:
```
sc.stop()
# This stops the Spark Session.
```

## 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

In [135…
```
# ('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)
# Here we can see that the generated text file contains average closing and opening data where companies are sorted in ascending order.
```

## 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 [19]:
```
def filter_data(record, start_date, end_date):
    date_str = record[10]

    formatted_date = date_str.replace("-", "-").replace(" ", "-")
    return start_date <= formatted_date <= end_date
```

In [20]:
```
start_date = "01-FEB-2023"
end_date = "30-APR-2023"
```

In [21]:
```
filtered_data = rdd2.filter(lambda record: filter_data(record, start_date, end_date))
```

In [22]:
```
company_high_pairs = filtered_data.map(lambda record: (record[0], float(record[2])))

high_sum_count = company_high_pairs.combineByKey(
    lambda x: (x, 1),
    lambda acc, x: (acc[0] + x, acc[1] + 1),
    lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1])
)
```

In [23]:
```
company_averages = high_sum_count.mapValues(lambda acc: acc[0] / acc[1])

sorted_results = company_averages.sortByKey(ascending=False)
```

In [24]:
```
sorted_results.saveAsTextFile("/home/hduser/spark/high_averages")
```

In [26]:
```
sorted_results.take(15)
```

Out[26]:
```
[('ZYLOG', 208.5269131556321),
 ('ZYDUSWELL', 582.6324161650904),
 ('ZUARIGLOB', 85.71753343239226),
 ('ZUARIAGRO', 518.3198979591837),
 ('ZUARI', 179.2710110584518),
 ('ZODJRDMKJ', 29.984196035242284),
 ('ZODIACLOTH', 278.0639732142858),
 ('ZICOM', 78.8250214961307),
 ('ZENTEC', 76.46021505376345),
 ('ZENSARTECH', 322.18981083404987),
 ('ZENITHINFO', 73.20907103825135),
 ('ZENITHEXPO', 51.70833333333333),
 ('ZENITHCOMP', 11.54088176352705),
 ('ZENITHBIR', 3.774075666380053),
 ('ZEENEWS', 13.065362318840577)]
```

In [27]:
```
sc.stop()
```

In [147…

In [ ]: