[1]:
```python
import findspark
findspark.init()
#inporting module to locate spark in VM
```

[2]:
```python
import pyspark
from pyspark.sql.types import *
#importing functions to run spark using python
```

[3]:
```python
sc = pyspark.SparkContext(appName="E10")
#creates a spark context, using which we assigns the cluster of computer on which our code will run
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/10/31 18:10:08 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

[4]:
```python
rdd1 = sc.textFile("/home/hduser/spark/nsedata.csv")
#importing csv file as rdd
```

[5]:
```python
rdd1 = rdd1.filter(lambda x: "SYMBOL" not in x)
#this is used to remove header row
```

[6]:
```python
rdd2 = rdd1.map(lambda x : x.split(","))
#this is used to create a new rdd from the rdd1 by splitting each record
```

[7]:
```python
# 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 ...
```

[8]:
```python
rdd_open = rdd2.map(lambda x : (x[0]+"_open",float(x[2])))
rdd_close = rdd2.map(lambda x : (x[0]+"_close",float(x[5])))
#this creates two new rdds which contain the opening and closing prices
#x[2] and x[5] are basically the positions containing the opening and closing prices
```

[3]:
```python
sc = pyspark.SparkContext(appName="E10")
#creates a spark context, using which we assigns the cluster of computer on which our code will run
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/10/31 18:10:08 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

[4]:
```python
rdd1 = sc.textFile("/home/hduser/spark/nsedata.csv")
#importing csv file as rdd
```

[5]:
```python
rdd1 = rdd1.filter(lambda x: "SYMBOL" not in x)
#this is used to remove header row
```

[6]:
```python
rdd2 = rdd1.map(lambda x : x.split(","))
#this is used to create a new rdd from the rdd1 by splitting each record
```

[7]:
```python
# 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 ...
```

[8]:
```python
rdd_open = rdd2.map(lambda x : (x[0]+"_open",float(x[2])))
rdd_close = rdd2.map(lambda x : (x[0]+"_close",float(x[5])))
#this creates two new rdds which contain the opening and closing prices
#x[2] and x[5] are basically the positions containing the opening and closing prices
```

[9]:
```python
rdd_united = rdd_open.union(rdd_close)
#here we are combining both the rdds into a single rdd to do their analysis
```

[10]:
```python
reducedByKey = rdd_united.reduceByKey(lambda x,y: x+y)
#to find the average we first need to find the sum of all
#hence, here we are basically summing up the opening and closing prices for each entry
```

[11]:
```python
temp1 = rdd_united.map(lambda x: (x[0],1)).countByKey()
countOfEachSymbol = sc.parallelize(temp1.items())
#finding out the number of times a given entry occured
#we creates a new key value pair rdd
#also we are counting the number of occurence of each stock using the "countbykey" function
#Parallelize is a method to create an RDD from an existing collection (For e.g Array) present in the driver.
```

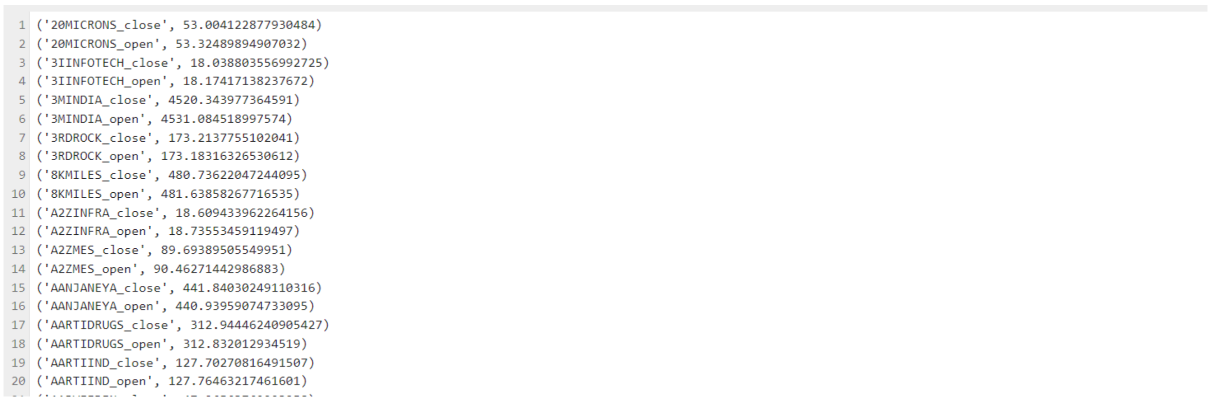
[12]:
```python
symbol_sum_count = reducedByKey.join(countOfEachSymbol)
#here we are combining the two rdds into a single one
```

[13]:
```python
averages = symbol_sum_count.map(lambda x : (x[0], x[1][0]/x[1][1]))
# here we are finally finding the average of each of the symbols
# so x[1]x[0]/x[1]x[1] is basically calculating sum of opening prices and closing prices of that particular symbol
```

[14]:
```python
averagesSorted = averages.sortByKey()
#we are sorting the records present in the rdd based on symbol name
```

[15]:
```python
averagesSorted.saveAsTextFile("/home/hduser/spark/averages")
# saving the results in folder named averages
```

[16]:
```python
sc.stop()
```



```
('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)
('AANJANEYA_open', 440.93959074733095)
('AARTIDRUGS_close', 312.94446240905427)
('AARTIDRUGS_open', 312.832012934519)
('AARTIIND_close', 127.70270816491507)
('AARTIIND_open', 127.76463217461601)
```

In [2]:
```python
import pyspark
from pyspark.sql.types import *
```

In [3]:
```python
sc = pyspark.SparkContext(appName="E10-01")
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogL
evel(newLevel).
23/10/28 17:12:24 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
```

In [4]:
```python
rdd7 = sc.textFile("/home/hduser/spark/nsedata.csv")
```

In [5]:
```python
rdd8 = rdd7.filter(lambda x: "SYMBOL" not in x)
```

In [6]:
```python
rdd9 = rdd8.map(lambda x : x.split(","))
```

In [17]:
```python
rdd_10 = rdd9.filter(lambda x: x[10] in "Apr" or  "May" or  "Jul" )
```

In [18]:
```python
rdd_high_1 = rdd_10.map(lambda x : (x[0]+"_high",float(x[3])))
```

In [19]:
```python
reducedByKey_1 = rdd_high_1.reduceByKey(lambda x,y: x+y)
```

In [20]:
```python
temp_1 = rdd_high_1.map(lambda x: (x[0],1)).countByKey()
countOfEachSymbol_1 = sc.parallelize(temp_1.items())
```

In [21]:
```python
symbol_sum_count_1 = reducedByKey_1.join(countOfEachSymbol_1)
```

In [22]:
```python
averages_high = symbol_sum_count_1.map(lambda x : (x[0], x[1][0]/x[1][1]))
```

In [23]:
```python
averagesSorted = averages_high.sortByKey(ascending=False)
```

In [24]:
```python
averagesSorted.saveAsTextFile("/home/hduser/spark/averages_high_4")
```