Tecnológico Nacional de México
Instituto Tecnológico de Tijuana

Subdirección Académica
Departamento de Sistemas y Computación

Semestre:
Agosto – Diciembre 2021

Carrera:
Ingeniería en Tecnologías de la Información y Comunicaciones

Materia y serie:
Datos Masivos BDD-1704TI9A

Unidad a evaluar: Unidad I

Nombre de la Tarea:
Práctica Evaluatoria - Unidad 1

Nombre del Alumno:
Sifuentes Martinez Manuel Javier 17212934

Nombre del docente:
José Christian Romero Hernández

Answer the following questions with Spark DataFrames and Scala using the Netflix_2011_2016.csv "CSV" found in the spark-dataframes folder.

In the evaluation of Big Data Unit 1, different actions and operations will be carried out with a provided Dataset. Spark syntax will be used to be able to reach all the expected results.

1. Start a simple Spark session.

The SparkSession library is imported so that later you can use the methods to start a session in spark.

```
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder().getOrCreate()
```

2. Upload Netflix Stock CSV file, have Spark infer the data types.

The CSV to be used is imported, specifying through the option function that header is true, this means that the first line is identified as the "title" of each column and it is saved in the constant df.

```
val df = spark.read.option("header",
"true").option("inferSchema","true")csv("Netflix_2011_2016")
```

3. What are the names of the columns?

Using the columns function, the name of the columns that the dataset contains can be displayed.

```
scala> df.columns
res0: Array[String] = Array(Date, Open, High, Low, Close, Volume, Adj
Close)
```

4. What is the scheme like?

The schema allows you to see the type of data that each column of the CSV has.

```
scala> df.printSchema()
root
 |-- Date: timestamp (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: integer (nullable = true)
 |-- Adj Close: double (nullable = true)
```

5. Print the first 5 columns.

Using the for loop to go through and print the first 5 lines of the dataset allow your display to be cleaner.

```
scala> for(row <- df.head(5)){
     |     println(row)
     | }
[2011-10-24
00:00:00.0,119.100002,120.28000300000001,115.100004,118.839996,120460200
,16.977142]
[2011-10-25
00:00:00.0,74.899999,79.390001,74.249997,77.370002,315541800,11.05285700
0000001]
[2011-10-26
00:00:00.0,78.73,81.420001,75.399997,79.400002,148733900,11.342857]
[2011-10-27
00:00:00.0,82.179998,82.71999699999999,79.249998,80.86000200000001,71190
000,11.551428999999999]
[2011-10-28
00:00:00.0,80.280002,84.660002,79.599999,84.14000300000001,57769600,12.0
2]
```

6. Use describe () to learn about the DataFrame.
Discribe allows you to see different statistical results that can be useful to us, such as the number of data, the maximum or minimum, among others.

```
scala> df.describe().show()
+-------+------------------+------------------+------------------+------
------------+--------------------+------------------+
|summary|              Open|              High|               Low|
Close|              Volume|         Adj Close|
+-------+------------------+------------------+------------------+------
------------+--------------------+------------------+
|  count|              1259|              1259|              1259|
1259|                1259|              1259|
|   mean|230.39351086656092|233.97320872915006|226.80127876251044|
230.522453845909|2.5634836060365368E7|55.610540036536875|
| stddev|164.37456353264244| 165.9705082667129|
162.6506358235739|164.40918905512854|
2.306312683388607E7|35.186669331525486|
|    min|         53.990001|         55.480001|             52.81|
53.8|             3531300|          7.685714|
|    max|        708.900017|        716.159996|        697.569984|
707.610001|           315541800|        130.929993|
+-------+------------------+------------------+------------------+------
------------+--------------------+------------------+
```

7. Create a new dataframe with a new column called “HV Ratio” which is the relationship between the price in the “High” column versus the “Volume” column of shares traded for a day.

Dividing the High column by Volume allows you to find the relationship between these two columns.

```
val df2 = df.withColumn("HV Ratio",df("High")/df("Volume"))

+-------------------+-----------------+------------------+----------+---
-------------+---------+------------------+--------------------+
|               Date|             Open|              High|       Low|
Close|   Volume|         Adj Close|            HV Ratio|
+-------------------+-----------------+------------------+----------+---
-------------+---------+------------------+--------------------+
|2011-10-24 00:00:00|       119.100002|120.28000300000001|115.100004|
118.839996|120460200|         16.977142|9.985040951285156E-7
+-------------------+-----------------+------------------+----------+---
-------------+---------+------------------+--------------------+
```

8. What day had the highest peak in the "Open" column?
To easily get the day with the highest selling price, just sort the dataset in descending order and take the first value.

```
df.orderBy($"Open".desc).show(1)

+-------------------+----------+----------+----------+----------+-------
-+-----------+
|               Date|      Open|      High|       Low|     Close|
Volume| Adj Close|
+-------------------+----------+----------+----------+----------+-------
-+-----------+
|2015-07-14
00:00:00|708.900017|711.449982|697.569984|702.600006|19736500|100.371429
|
+-------------------+----------+----------+----------+----------+-------
-+-----------+
```

9. What is the meaning of the Close column "Close" in the context of financial information, explain it, there is nothing to code?

```
//It is the price at which the stock sales ended on the respective day.
```

10. What is the maximum and minimum in the "Volume" column?
Using the Max and Min function we can quickly find the maximum and minimum values of the Volume column.

```
df.select(max("Volume")).show()

+-----------+
|max(Volume)|
+-----------+
|  315541800|
+-----------+

df.select(min("Volume")).show()
+-----------+
|min(Volume)|
+-----------+
|    3531300|
+-----------+
```

11. With Scala/Spark $ Syntax answer the following:

A. How many days was the “Close” column less than $ 600?

The spark method is used, which allows filtering data from a column with a specific condition to see those less than 600.

```
df.filter($"Close"<600).count()

res1: Long = 1218
```

B. What percentage of the time was the “High” column greater than $ 500?

The same as the previous one, but now first the ones greater than 500 are searched, and from this it is multiplied by 1 and divided by the total of records in data frame and multiplied by 100, to obtain the required percentage.

```
(df.filter($"High">500).count()*1.0/df.count())*100

res2: Double = 4.924543288324067
```

C. What is the Pearson correlation between column "High" and column "Volume"?

We use the spark method of corr to obtain the Pearson correlation.

```
df.select(corr("High","Volume")).show()

+--------------------+
|  corr(High, Volume)|
+--------------------+
|-0.20960233287942157|
+--------------------+
```

D. What is the maximum in the “High” column per year?

A new dataframe is obtained from the date column and then the required year is obtained. Once again another dataframe is created that has the highest values of each year, ending with showing the maximum value of each year.

```
val df_year = df.withColumn("Year",year(df("Date")))
val df_max = df_year.select($"Year",$"High").groupBy("Year").max()
df_max.select($"Year",$"max(High)").show()

+----+------------------+
|Year|         max(High)|
+----+------------------+
|2015|        716.159996|
|2013|        389.159988|
|2014|        489.290024|
|2012|        133.429996|
|2016|129.28999299999998|
|2011|120.28000300000001|
+----+------------------+
```

E. What is the "Close" column average for each calendar month?
A new dataframe is obtained from the date column and then the required month is obtained. Once again another dataframe is created that has the values with which I close each month, ending with showing the average value of each month.

```
val df_month = df.withColumn("Month",month(df("Date")))
val month_avgs = df_month.select($"Month",$"Close").groupBy("Month").mean()
month_avgs.select($"Month",$"avg(Close)").show()

+-----+------------------+
|Month|        avg(Close)|
+-----+------------------+
|   12| 199.3700942358491|
|    1|212.22613874257422|
|    6| 295.1597153490566|
|    3| 249.5825228971963|
|    5|264.37037614150944|
|    9|206.09598121568627|
|    4|246.97514271428562|
|    8|195.25599892727263|
|    7|243.64747528037387|
|   10|205.93297300900903|
|   11| 194.3172275445545|
|    2| 254.1954634020619|
+-----+------------------+
```