TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE SISTEMAS COMPUTACIONALES
Semestre: AGOSTO - DICIEMBRE 2020

CARRERA

Ingeniería en Sistemas Computacionales

MATERIA
Datos Masivos

TÍTULO
Práctica Evaluatoria

UNIDAD
1

ALUMNO(S) Y NO. DE CONTROL
Encarnacion Ocampo Gustavo 16211993
Alcantar Balcon Adrian Giovanny 16210504

DOCENTE
JOSE CHRISTIAN ROMERO HERNANDEZ

Introduction.

Big data is a field that deals with ways to systematically analyze, extract information, or deal with data sets that are too large or complex to be handled by traditional data processing application software. Data with many cases (rows) offer higher statistical power, while data with greater complexity (more attributes or columns) can lead to a higher rate of false discoveries. Big Data challenges include data capture, data storage, data analysis, search, exchange, transfer, display, query, update, information privacy, and data source. In the matter of big data, the students were asked to carry out an evaluative practice, this is based on the Spark functions with the Scala language for the Dataframes section.

The exercises were the following:

1. **Start a simple Spark session.**

Solution:

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._
import spark.implicits._

val spark = SparkSession.builder().getOrCreate()
```

A SparkSession class was imported to be able to start the simple session, sql.functions to be able to use sql functions and spark.implicits this is a method to convert common Scala objects into the DataFram.

Result:

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._
import spark.implicits._
spark: org.apache.spark.sql.SparkSession =
org.apache.spark.sql.SparkSession@5b9ca528
```

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

Solution::

```
val df = spark.read.option("header",
"true").option("inferSchema","true")csv("/home/gussy/git_workspace/Big-
Data2020/Unidad1/Evaluaciones/Netflix_2011_2016.csv")
val df = spark.read.option("header",
"true").option("inferSchema","true")csv("C:/Users/adria/Documents/GitHub/Big-
Data2020/Unidad1/Evaluaciones/Netflix_2011_2016.csv")
```

The csv file was loaded via its directory to the df value for manipulation in Spark.

Result:

```
df: org.apache.spark.sql.DataFrame = [Date: timestamp, Open: double ... 5 more fields]
```

3. **What are the names of the columns?**

Solution:

```
df.columns
```

The .columns operation will display the name of the columns of the dataframe in an array.

Result:

```
res0: Array[String] = Array(Date, Open, High, Low, Close, Volume, Adj Close)
```

4. **What is the scheme like?**

Solution:

```
df.printSchema()
```

The .printSchema () operation will display the dataframe scheme, in which we can see the name of each column together with its data type and if it can be null or not.

Result:

```
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.**

Solution:

```
df.head(5)
```

The .head (5) operation will display the first 5 columns (or those that we mark in parentheses) with the first 5 lines in an array.

Result:

```
res1: Array[org.apache.spark.sql.Row] = Array([2011-10-24
00:00:00.0,119.100002,120.28000300000001,115.100004,118.839996,120460200,16.977
142], [2011-10-25
00:00:00.0,74.899999,79.390001,74.249997,77.370002,315541800,11.052857000000001]
, [2011-10-26 00:00:00.0,78.73,81.420001,75.399997,79.400002,148733900,11.342857],
[2011-10-27
00:00:00.0,82.179998,82.71999699999999,79.249998,80.86000200000001,71190000,11.
551428999999999], [2011-10-28
00:00:00.0,80.280002,84.660002,79.599999,84.14000300000001,57769600,12.02])
```

6. **Use describe () to learn about the DataFrame.**

Solution:

```
df.describe().show()
```

The .describe () operation will display essential data about the dataframe, such as the name of the columns, the amount of data in each one, the average per column, and the minimum and maximum value of each column.

Result:

```
+-------+-----------------+-----------------+-----------------+-----------------+-------------------+-----------------+
|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|55.6105400365368 75|
| 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. (Hint: It is a column operation).**

Solution:

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

We declare a new value with the name of hvRatio to which we apply a Withcolum which is used to change the name, as in this case the column will be named "HV Ratio", then we select the column "High" and "Volume" and apply it / so that it will help us to know the relationship between the price of both columns by performing the column operation.

Result:

```
|               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|
|2011-10-25 00:00:00|        74.899999|         79.390001| 74.249997|
77.370002|315541800|11.052857000000001|2.515989989281927E-7|
|2011-10-26 00:00:00|            78.73|         81.420001| 75.399997|
79.400002|148733900|         11.342857|5.474206014903126E-7|
|2011-10-27 00:00:00|        82.179998| 82.71999699999999|
79.249998|80.86000200000001| 71190000|11.551428999999999|1.161960907430818...|
|2011-10-28 00:00:00|        80.280002|         84.660002| 79.599999|84.14000300000001|
57769600|             12.02|1.465476686700271...|
|2011-10-31 00:00:00|83.63999799999999|         84.090002| 81.450002|         82.080003|
39653600|         11.725715|2.120614572195210...|
|2011-11-01 00:00:00|        80.109998|         80.999998|     78.74|         80.089997|
```

```
33016200|         11.441428|2.453341026526372E-6|
|2011-11-02 00:00:00|        80.709998|         84.400002| 80.109998|        83.389999|
41384000|         11.912857|2.0394355789677717E-6|
|2011-11-03 00:00:00|        84.130003|         92.600003| 81.800003|        92.290003|
94685500|13.184285999999998| 9.77974483949496E-7|
|2011-11-04 00:00:00|91.46999699999999| 92.89000300000001| 87.749999|
90.019998| 84483700|             12.86|1.099502069629999...|
|2011-11-07 00:00:00|             91.0|         93.839998| 89.979997|        90.830003| 47485200|
12.975715|1.976194645910725...|
|2011-11-08 00:00:00|91.22999899999999|         92.600003| 89.650002|        90.470001|
31906000|         12.924286|2.902275528113834...|
|2011-11-09 00:00:00|        89.000001|         90.440001| 87.999998|        88.049999|
28756000|         12.578571|3.145082800111281E-6|
|2011-11-10 00:00:00|        89.290001| 90.29999699999999|
84.839999|85.11999899999999| 39614400|             12.16|2.279474054889131E-6|
|2011-11-11 00:00:00|        85.899997|         87.949997|      83.7|        87.749999| 38140200|
12.535714|2.305965805108520...|
|2011-11-14 00:00:00|        87.989998|             88.1|     85.45|        85.719999| 21811300|
12.245714|4.039190694731629...|
|2011-11-15 00:00:00|           85.15|         87.050003| 84.499998|        86.279999|
21372400|         12.325714|4.073010190713256...|
|2011-11-16 00:00:00|        86.460003|         86.460003| 80.890002|        81.180002|
34560400|11.597142999999999|2.501707242971725E-6|
|2011-11-17 00:00:00|           80.77|         80.999998| 75.789999|        76.460001|
52823400|         10.922857|1.533411291208063...|
|2011-11-18 00:00:00|            76.7|         78.999999| 76.039998|        78.059998| 34729100|
11.151428|2.274749388841058...|
+-------------------+-----------------+------------------+----------+-----------------+---------+------------------
+--------------------+
```

8. **Which day had the highest peak in the “Close” column?**

Solution:

```
df.groupBy(dayofweek(df("Date")).alias("Day")).max("Close").sort(asc("Day")).show()
```

To carry out this problem it was first necessary to perform a GroupBy, once we had that we found a function called dayofweek that is used in C # but at the same time it could also be applied in scala, we made a reference to the column "Date" and gave it an Alias " Day ”and we apply the Max method so that it obtains the maximum values of each day that on the“ Close ”column, giving it a .sort so that it orders it, and we also choose to put an asc on the Day so that it orders them Ascendant.

Result:

```
+---+----------------+
|Day|      max(Close)|
+---+----------------+
|  2|      707.610001|
|  3|      702.600006|
|  4|678.6099780000001|
|  5|      670.089996|
|  6|      680.599983|
+---+----------------+
```

9. **Write in your own words in a comment of your code. What is the meaning of the Close column "Close"?**

Solution:

```
//Es el Total de los valores con los que cerraron esa fecha
```

In the stock market, "Close" is the term used for the security at which it was closed that day.

10. **What is the maximum and minimum of the “Volume” column?**

Solution:

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

In this exercise the select () operation was used and in it we only sent to call the maximum value and the minimum value of the Volume column to display it last with the .show () operation

Result:

```
+-----------+-----------+
|max(Volume)|min(Volume)|
+-----------+-----------+
|  315541800|    3531300|
+-----------+-----------+
```

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

- **Hint: Basically very similar to the dates session, you will have to create another dataframe to answer some of the items.**

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

Solution:

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

To solve problem A, we filter on the “Close” column, but only those that are <600 and we count them to obtain the result that was requested.

Result:

```
res6: Long = 1218
```

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

Solution:

```
val perTiem = df.filter($"High">500).count()
val totalR = df.count()
val percen = (perTiem*1.0)/(totalR*100)
```

In this exercise, the number of data greater than 500 in the “High” column had to be calculated, so first that calculation was assigned to a value named perTiem, the filter () operation was used in the “High” column and with count () any value greater than 500 was counted. In the TotalR value we simply assign the amount of total data in the dataframe by counting it with a count (). The rule of 3 is assigned to the value percen last in order to calculate the percentage that was needed in this exercise. For the rule of 3, the perTiem value (the amount of data filtered) is multiplied by 1 (to make it whole) and divided by the multiplication of TotalR (the total amount of data) by 100, which gives us the result of the exercise .

Result:

```
percen: Double = 4.9245432883240 66E-4
```

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

Solution:

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

To obtain the Pearson Correlation between the column "High" and "Volume" it was only necessary to first perform a select and before knowing the columns to select we use the .corr that this will make the pearson correlation on the selected columns "High" and "Volume".

Result:

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

d. ¿Cuál es el máximo de la columna “High” por año?

Solución:

```
val df2 = df.withColumn("Year", year(df("Date")))
val dfmaxyear = df2.groupBy("Year").max()
dfmaxyear.select($"Year", $"max(High)").show()
```

For this exercise we had to start by making another dataframe declaring as a value of name df2, to this we assign a new column named "Year" in which we extract the data year (only the year of the dates) from the column "Date" of the original dataframe. In the next step we define the dfmaxyear value in which we group the maximums of the new column "Year" with the operation .groupBy and .max (). Finally we make a .select in the dfmaxyear value of the maximum in the column "High" per year, using the new column "Year" and displaying with .show ()

Result:

```
+----+-----------------+
|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?

Solution:

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

A new value called monthdf is declared to which we made a WhitColum to change the name to “Month” all on the column of “Date”, extracting only the Month, then we create another new

value to which we name it monthavg referring to the fact that we want to know the average of the months, in this it was necessary to call our previously created value and make a select to "Month" and "Close" and we applied a GroupBy in "Month" and the .mean we used for the one to perform the calculation of the average. Finally we just call our value monthavgs doing a select again on month and the avg (average) of Close.

Result:

```
+-----+-----------------+
|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|
+-----+-----------------+
```

Conclusion.

Since the beginning of globalization, data went from being insufficient to being overwhelmingly large, leaving the question of what to do with it. Data Science, Data Analytics and Big Data were born to be able to translate all this data into graphs that entrepreneurs can understand to help them decide the next step they will take, although this can be applied to more fields. Dataframes are two-dimensional data structures (row and column) that can contain different types of data, this data structure is the most used when performing data analysis and using statistical libraries. For Big Data, it is necessary to know how to manipulate these dataframes with the tools that are at our disposal, since from them we can obtain or extract information, analysis and forecasts of the subject assigned to us.

In conclusion, it can be said that the evaluative practice was not complicated, but it was necessary to investigate some functions or methods to be able to solve some points that were requested, the way to make the clearest and most accurate solutions possible was sought, while investigating realizes that scala offers us too many tools for the solution or extraction of information required on a DataFrame, also with the knowledge had about other languages and the syntax that dealt with SQL, it was helpful to find if it could be done in scala, and Seeing that it is similar made it easier for us to resolve the work.

.

Video explicatorio: https://www.youtube.com/watch?v=w-Jjw3dQIoo&feature=youtu.be

Fuentes.

- https://spark.apache.org/docs/2.4.7/sql-getting-started.html
- https://spark.apache.org/docs/2.4.7/api/scala/index.html#org.apache.spark.sql.Dataset