**Instituto Tecnológico de Tijuana**

**Ingeniería en Sistemas Computacionales** 

**Examen I**

Scala Basic, Funciones y Dataframes

**Materia:** Datos Masivos

**Unidad:** Unidad I

**Facilitador:**

José Christian Romero Sánchez

**Alumnos:**

* Hernández Negrete Juan Carlos - **16212021**
* Manzano Guzman Jesua - **16212033**

**Fecha:**

Tijuana Baja California a 28 de Octubre del 2020.

**Introduction.**

The purpose of this document is to argue what was raised in the examination of the class Big Data. In this we will present our answers and the process that led us to them.

This exam was written in the Scala programming language and the Apache Spark framework.

**Development**

**Evaluation Practice Unit 1**

|  |
| --- |
| **import org.apache.spark.sql.SparkSession val spark = SparkSession.builder().getOrCreate() val df = spark.read.option("header", "true").option("inferSchema","true")csv("Netflix\_2011\_2016.csv")** |

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

The entry point into all functionality in Spark is the SparkSession class. To create a basic SparkSession, just use SparkSession.builder ():

|  |
| --- |
| **scala> import org.apache.spark.sql.SparkSession  import org.apache.spark.sql.SparkSession  scala> val spark = SparkSession.builder().getOrCreate()  spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@69e8f7a5** |

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

With a SparkSession, applications can create DataFrames from an existing RDD, from a Hive table, or from Spark data sources. In this case, we will take the CSV file "Netflix\_2011\_2016.csv"

|  |
| --- |
| **scala> val df = spark.read.option("header", "true").option("inferSchema","true")csv("Netflix\_2011\_2016.csv")  df: org.apache.spark.sql.DataFrame = [Date: timestamp, Open: double ... 5 more fields]** |

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

The column names are: Date, Open, High, Low, Close, Volume and Adj Close

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

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

In order to know the structure of the schema of our dataframe we will use printSchema that will describe the type of each of our fields in the dataframe

|  |
| --- |
| **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.**

With head we print the first 5 values of our dataframe as an array

|  |
| --- |
| scala> df.head(5)  res2: Array[org.apache.spark.sql.Row] = Array([2011-10-24 00:00:00.0,119.100002,120.28000300000001,115.100004,118.839996,120460200,16.977142], [2011-10-25 00:00:00.0,74.899999,79.390001,74.249997,77.370002,315541800,11.052857000000001], [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.**

The average, column count, minimum and maximum, etc. of our dataframe are described below

|  |
| --- |
| 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. (Hint: It is a column operation).**

The calculation of the operation between HIGH and VOLUME is carried out where high will be the highest point that was had depending on the date where we will have to divide it by the volume to be able to know the average of the sale that was with the stock that we have

|  |
| --- |
| scala> val df2 = df.withColumn("HV Ratio",df("High")/df("Volume")) df2: org.apache.spark.sql.DataFrame = [Date: timestamp, Open: double ... 6 more fields]  scala> df2.show() +-------------------+-----------------+------------------+----------+-----------------+---------+------------------+--------------------+ | 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|2.039435578967717E-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...| +-------------------+-----------------+------------------+----------+-----------------+---------+------------------+--------------------+ only showing top 20 rows |

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

All the high values are grouped within the dataframe where they will be displayed in descending order, in order to show the highest value first and only be able to show the first value of the dataframe

|  |
| --- |
| **scala> df.orderBy($"High".desc).show(1)  +-------------------+-----------------+----------+----------+----------+--------+------------------+ | Date| Open| High| Low| Close| Volume| Adj Close| +-------------------+-----------------+----------+----------+----------+--------+------------------+ |2015-07-13 00:00:00|686.6900019999999|716.159996|686.550026|707.610001|33205200|101.08714300000001| +-------------------+-----------------+----------+----------+----------+--------+------------------+ only showing top 1 row** |

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

|  |
| --- |
| In relation to the dataframe with the CSV, they are stock values or records where the date of their registration is taken into account, where the entry and exit of these can be considered with OPEN and CLOSE, that is, how many there were before and after , like HIGH and LOW which would be the highest and lowest stock. |

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

To determine the maximum or minimum of a column we must use the select function and the parameter "max" or "min" depending on the case, and to be able to observe the result the "show" function is used.

|  |
| --- |
| df.select(max("Volume")).show() df.select(min("Volume")).show() |

**Results**

For the maximum

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

For the minimum

|  |
| --- |
| +-----------+ |min(Volume)| +-----------+ | 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.

In order to carry out the following exercises it is necessary to import a library (spark.implicits)

|  |
| --- |
| import spark.implicits.\_ |

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

In order to obtain the result, we must first know which elements of the column meet the condition, for that we use "filter" with the condition and to determine the number of elements that satisfy it we use "count".

|  |
| --- |
| val result = df.filter($"Close" < 600).count() |

**Results**

|  |
| --- |
| result: Long = 1218 |

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

In order to obtain the result, we must first know which elements of the column meet the condition, for that we use the "filter" function with the "High" condition greater than 500 and to determine the number of elements that satisfy it we use "count", then we multiply by 1.0 to be able to obtain the result in double data type, we divide it by the number of data that we obtain with "df.count ()" and finally we multiply by 100.

|  |
| --- |
| val result = (df.filter($"High" > 500).count())\*1.0/df.count()\*100 |

**Results**

|  |
| --- |
| result: Double = 4.924543288324067 |

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

To obtain the Pearson correlation, we first have to use the "Select" function with the corr ("High", "Volume") parameter that corresponds to the correlation of both columns and to display the data we use the "show" function.

|  |
| --- |
| df.select(corr("High", "Volume")).show() |

**Results**

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

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

To determine this, we must first take the value of the year from the date column and put it in a new column, then we have to determine the maximum of each row and group it by year, then we have to select the columns that we are going to show in in this case the year and the maximum of the High column.

|  |
| --- |
| val df2 = df.withColumn("Year", year(df("Date"))) val df3 = df2.select($"Year", $"High").groupBy("Year").max() df3.select($"Year", $"max(High)").show() |

**Results**

|  |
| --- |
| +----+------------------+  |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?**

This exercise is very similar to the previous one, first we must create a new column with the month of the Date column, then we calculate the average of the columns and group it by the month and finally we select the month column and the column of the average of Close.

|  |
| --- |
| val df2 = df.withColumn("Month", month(df("Date"))) val df3 = df2.select($"Month", $"Close").groupBy("Month").mean() df3.select($"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| +-----+------------------+ |

**Conclusión**

In this unit the basics of scala were learned, such as the language and its functions, as well as the use of spark apache to check results, what was seen in class was applied in this exam, as well as some examples that could be used in this exam and others not, and thus, be able to solve problems.

**Repository**

<https://github.com/JesuaMG/BigData/tree/Unit_1/Unit1/Evaluation>

**Video**

<https://youtu.be/2mnYFzRepGA>