# TECNOLÓGICO NACIONAL DE MÉXICO INSTITUTO TECNOLÓGICO DE TIJUANA

**SUBDIRECCIÓN ACADÉMICA**

**DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN**

**SEMESTRE FEBRERO-JUNIO 2022**

**CARRERA**
**Ingeniería en informática**

**MATERIA**
**Datos masivos**

**TÍTULO**
**Práctica evaluatoria, unidad #1**

**Integrantes:**
**Munguía silva Edgar Geovanny #17212344**
**Alicia Guadalupe Pérez López#18210514**

**NOMBRE DEL MAESTRO**
**Jose Christian Romero Hernadez**

**Tijuana Baja California 22 de marzo del 2022**

**Introduction.**

The goal of this practice it's to show how scala/spark works in an introductory level, starting by initializing a simple spark session to make more complex things, we will learn how to load CSV files and make the programming language infer on them, also we are learning how to identify every element that is part of the environment, for example, the schema, the total of data, the type of data and other interesting information that might be useful in the future for another kind of representation(graphical for example). We will use the basic functions of scala, like the methods describe(),columns(), printhschema(), head() and others. It's an interesting practice, because we will implement these functions to find new data, patterns and other important information that might be useful, for example,find maximum and minimums of certains columns , predict the future taking the previous data as base, and a lot of interesting things. We will show the code and the steps we followed to make this evaluatory practice.

**1. Start a simple spark session:**

```
/*1.-Comienza una simple sesión Spark.
  */
spark-shell
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.8
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_321)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

**2. Load the file Netflix Stock CSV, and make that spark infer on the data type.**

```
/*2.-Cargue el archivo Netflix Stock CSV, haga que Spark infiera los tipos de datos.
  */
val dataframe=spark.read.option("header","true").option("inferSchema","true")csv("Netflix_2011_2016.csv")
```

```
scala> val dataframe=spark.read.option("header","true").option("inferSchema","true")csv("Netflix_2011_2016.csv")
dataframe: org.apache.spark.sql.DataFrame = [Date: timestamp, Open: double ... 5 more fields]

scala>
```

### 3. What are the columns name?

```
/*3.-¿Cuáles son los nombres de las columnas?
*/
dataframe.columns
```

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

scala>
```

### 4. How is the schema?

```
/*4.-¿Cómo es el esquema?
   */
  dataframe.printSchema()
```

```
scala>  dataframe.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)


scala>
```

### 5. Print the first 5 columns.

```
/*5.-Imprime las primeras 5 columnas.
   */
   dataframe.head(5)
```

```
scala> dataframe.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,
49997,77.370002,315541800,11.052857000000001], [2011-10-26 00:00:00.0,78.73,81.420001,75.399997,79.400002,148733900,11.34
0.86000200000001,71190000,11.551428999999999], [2011-10-28 00:00:00.0,80.280002,84.660002,79.599999,84.14000300000001,577

scala>
```

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

```
/*6.-Usa describe () para aprender sobre el DataFrame.
    */
    dataframe.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 "HV Ratio" which is the ratio that calls exists between the price of the “High” column versus the “Volume” column of shares traded for one day.**

```
/*7. Crea un nuevo dataframe con una columna nueva llamada “HV Ratio” que es la relación que
existe entre el precio de la columna “High” frente a la columna “Volumen” de acciones
negociadas por un día. Hint - es una operación*/

val dataframenew = dataframe.withColumn("HV Ratio", dataframe("High")- dataframe("Volume"))
/* mostrar columna nueva*/
dataframenew.show()
```

```
scala> dataframenew.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|-1.20460079719997E8|
|2011-10-25 00:00:00|        74.899999|        79.390001| 74.249997|        77.370002|315541800|11.052857000000001|-3.15541720609999E8|
|2011-10-26 00:00:00|            78.73|        81.420001| 75.399997|        79.400002|148733900|        11.342857|-1.48733818579999E8|
|2011-10-27 00:00:00|        82.179998| 82.71999699999999| 79.249998|80.86000200000001| 71190000|11.551428999999999| -7.1189917280003E7|
|2011-10-28 00:00:00|        80.280002|        84.660002| 79.599999|84.14000300000001| 57769600|            12.02| -5.7769515339998E7|
|2011-10-31 00:00:00|83.63999799999999|        84.090002| 81.450002|        82.080003| 39653600|        11.725715| -3.9653515909998E7|
|2011-11-01 00:00:00|        80.109998|        80.999998|     78.74|        80.089997| 33016200|        11.441428| -3.3016119000002E7|
+-------------------+-----------------+-----------------+----------+-----------------+---------+-----------------+-------------------+
```

Ln 32, Col 1 (19 sele

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

```
/*8. ¿Qué día tuvo el pico mas alto en la columna “Open”?
*/
dataframe.select(max("Open")).show()
dataframe.orderBy($"Open".desc).show(1)
```

```
scala> dataframe.select(max("Open")).show()
+----------+
| max(Open)|
+----------+
|708.900017|
+----------+


scala>
```

```
scala> dataframe.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|
+-------------------+----------+----------+----------+----------+--------+----------+
only showing top 1 row


scala>
```

**9.What is the meaning of the “Close” column in the context of financial information, explain it, you don't have to code anything.**

**Answer=**We could find a relationship between the column Close and High,when the column High raises, the other column (Close) also raises, that's the main pattern we could find in this dataframe.

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

```
/* 10. ¿Cuál es el máximo y mínimo de la columna “Volumen”? */

dataframe.select(max("volume")).show()
dataframe.select(min("volume")).show()
```

```
scala> dataframe.select(max("volume")).show()
+-----------+
|max(volume)|
+-----------+
|  315541800|
+-----------+


scala> dataframe.select(min("volume")).show()
+-----------+
|min(volume)|
+-----------+
|    3531300|
+-----------+
```

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

**A. How many days was the “Close” column under $600?**

```
/*a. ¿Cuántos días fue la columna “Close” inferior a $ 600?*/
dataframe.filter($"Close"<600).count()
```

```
scala> dataframe.filter($"Close"<600).count()
res10: Long = 1218

scala>
```

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

```
/* b. ¿Qué porcentaje del tiempo fue la columna “High” mayor que $ 500?

val P1 = dataframe.filter($"High"> 500). count()
val P2 = dataframe.filter($"High">0).count()
val P3 : Double = P1*100
P3/P2
```

```
scala> val P1 = dataframe.filter($"High"> 500). count()
P1: Long = 62

scala> val P2 = dataframe.filter($"High">0).count()
P2: Long = 1259

scala> val P3 : Double = P1*100
P3: Double = 6200.0

scala> P3/P2
res12: Double = 4.924543288324067

scala>
```

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

```
/*c. ¿Cuál es la correlación de Pearson entre columna “High” y la columna “Volumen”?*/
dataframe.select(corr("High","Volume")).show
```

```
scala> dataframe.select(corr("High","Volume")).show
+--------------------+
|  corr(High, Volume)|
+--------------------+
|-0.20960233287942157|
+--------------------+


scala>
```

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

```
/* d. ¿Cuál es el máximo de la columna “High” por año?*/
val year1 = dataframe.withColumn("Year", year(dataframe("Date")))
val Year2 = year1.select($"Year", $"High").groupBy("Year").max()
Year2.select($"Year", $"max(High)").show()
```

```
scala> val year1 = dataframe.withColumn("Year", year(dataframe("Date")))
year1: org.apache.spark.sql.DataFrame = [Date: timestamp, Open: double ... 6 more fields]

scala> val Year2 = year1.select($"Year", $"High").groupBy("Year").max()
Year2: org.apache.spark.sql.DataFrame = [Year: int, max(Year): int ... 1 more field]

scala> Year2.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 average of the “Close” column for each calendar month?**

```
/* e. ¿Cuál es el promedio de columna “Close” para cada mes del calendario?*/
val Mes = dataframe.withColumn("Month", month (dataframe("Date")))

val mespr= Mes.select($"Month",$"Close").groupBy("Month").mean()

mespr.select($"Month",$"avg(Close)").show()
```

```
scala> val Mes = dataframe.withColumn("Month", month (dataframe("Date")))
Mes: org.apache.spark.sql.DataFrame = [Date: timestamp, Open: double ... 6 more fields]

scala> val mespr= Mes.select($"Month",$"Close").groupBy("Month").mean()
mespr: org.apache.spark.sql.DataFrame = [Month: int, avg(Month): double ... 1 more field]

scala> mespr.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|
+-----+-----------------+
```

**Conclusions.**

**Edgar Munguia:** This evaluation practice was an interesting one, because i learned the syntax of another programming language that i have not learned before. I learned a lot of useful things, for example: to find peaks of certain data, the maximum and minimum, the mean,i also learned to identify the data type of a data frame, i learned a bit about the methods that scala/ spark offers us, and i realized we can do a lot of operations and things using this programming language, this time, was a kind of introductory practice to start knowing the environment and the things we can do. The possibilities are infinite, in this time, we used just a few of the functions that it offers to us and we are more aware of the power of scala/spark in big data context and other important data science areas.

**Alicia Pérez:**
By working with these tools (spark/scala), I have learned a little more, despite the fact that I had a little difficulty doing it, the topic seems very efficient to me, in addition to expanding my knowledge.
In the same way, once we have worked on the functions and data, we are gaining more practice in managing the databases.

**Github repository link:** ***https://github.com/Aliciap26/DATOS-MASIVOS***

**Youtube evidence video:** ***https://www.youtube.com/watch?v=ig76Gs_5EMM***