TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

NOMBRE DE LOS ALUMNOS:

GALAVIZ LONA OSCAR EDUARDO (N.CONTROL: 17212993)

MARQUEZ MILLAN SEASHELL VANESSA (N.CONTROL:)

Carrera: Ingeniería Informática

Semestre: 9no

MATERIA: Datos Masivos

PROFESOR: JOSE CHRISTIAN ROMERO HERNANDEZ

Practica evaluatoria 1

Unidad 1

Introduccion

En este documento se explicara la realizacion de la practica evaluatoria correspondiente a la unidad 1, veremos la utilizacion de una sesion de spark, la utilizacion de un archivo csv y la manipulacion de los datos dentro del mismo. pero antes de adentrarnos en la explicacion primero debemos entender que es Apache spark la herramienta que nos permitio realizar esta practica

Apache spark combina un sistema de computacion distribuida a traves de clusters de ordenadores mediante una manera sencilla y elegante de escribir programas, es considerado el primer software de codigo abierto permitiendo que la programacion distribuida sea accessible para los cientificos de datos

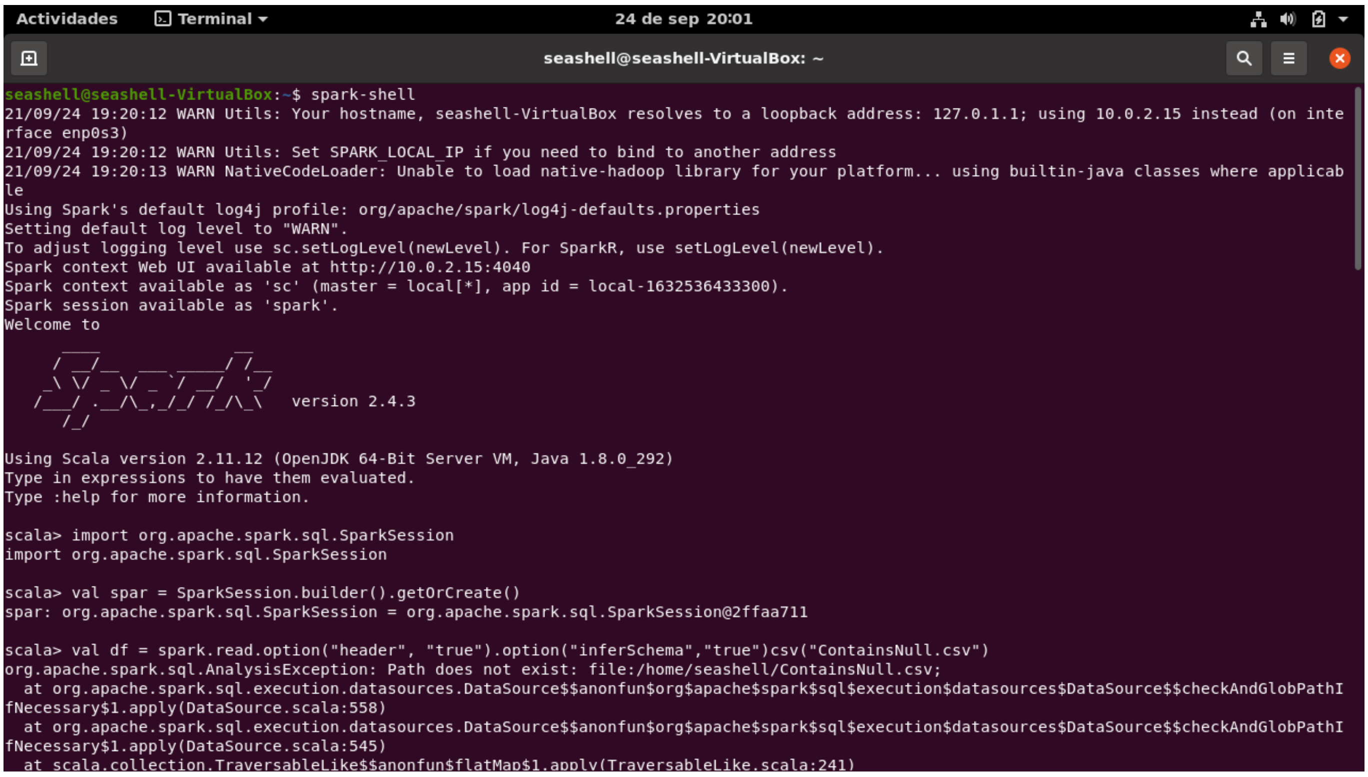
En cuanto a lo que se refiere la herramienta apache spark es util y eficiente para tareas de procesamiento masivo de datos. El entender el lengua es facil ya que tiene1 la bases de otros lenguajes para realizacion de operaciones y manipulacion de datos, ademas cuenta con la importacion de funciones que no se tengan dentro del scala ademas de la utilizacion de los dataframes que son archivos o contenedores de datos que nos permitira ver, leer, manipular los datos de un archivo csv, esto nos permitara realizar operaciones que nos den informacion para sacar conclusiones respecto a los datos que obtenemos.

Desarrollo

1-.SPARK session

Only start SPARK with the comand

```
SPARK-SHELL
```

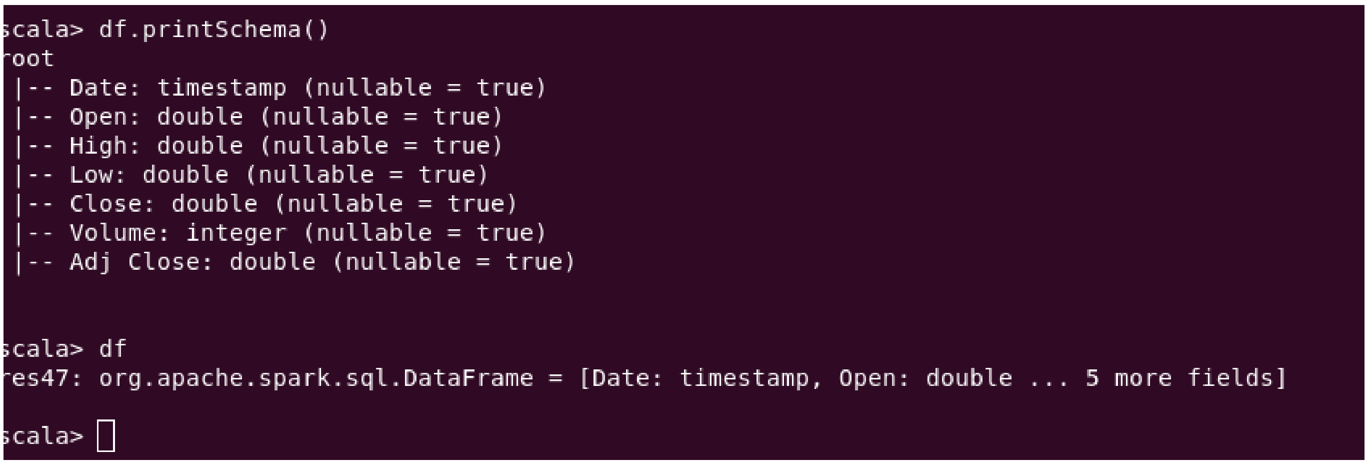


```
seashell@seashell-VirtualBox:~$ spark-shell
21/09/24 19:20:12 WARN Utils: Your hostname, seashell-VirtualBox resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on inte
rface enp0s3)
21/09/24 19:20:12 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
21/09/24 19:20:13 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicab
le
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1632536433300).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.3
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_292)
Type in expressions to have them evaluated.
Type :help for more information.

scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> val spar = SparkSession.builder().getOrCreate()
spar: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@2ffaa711

scala> val df = spark.read.option("header", "true").option("inferSchema","true")csv("ContainsNull.csv")
org.apache.spark.sql.AnalysisException: Path does not exist: file:/home/seashell/ContainsNull.csv;
  at org.apache.spark.sql.execution.datasources.DataSource$$anonfun$org$apache$spark$sql$execution$datasources$DataSource$$checkAndGlobPathI
fNecessary$1.apply(DataSource.scala:558)
  at org.apache.spark.sql.execution.datasources.DataSource$$anonfun$org$apache$spark$sql$execution$datasources$DataSource$$checkAndGlobPathI
fNecessary$1.apply(DataSource.scala:545)
  at scala.collection.TraversableLike$$anonfun$flatMap$1.apply(TraversableLike.scala:241)
```

2-. File Netflix Stock CSV

First need import the library, and is important you have the dataFrame in this address "/home/"name computer"/"name dataFrame" because the comand stearchh the archive here, then only print the data types

```
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")
df
df.printSchema()
```

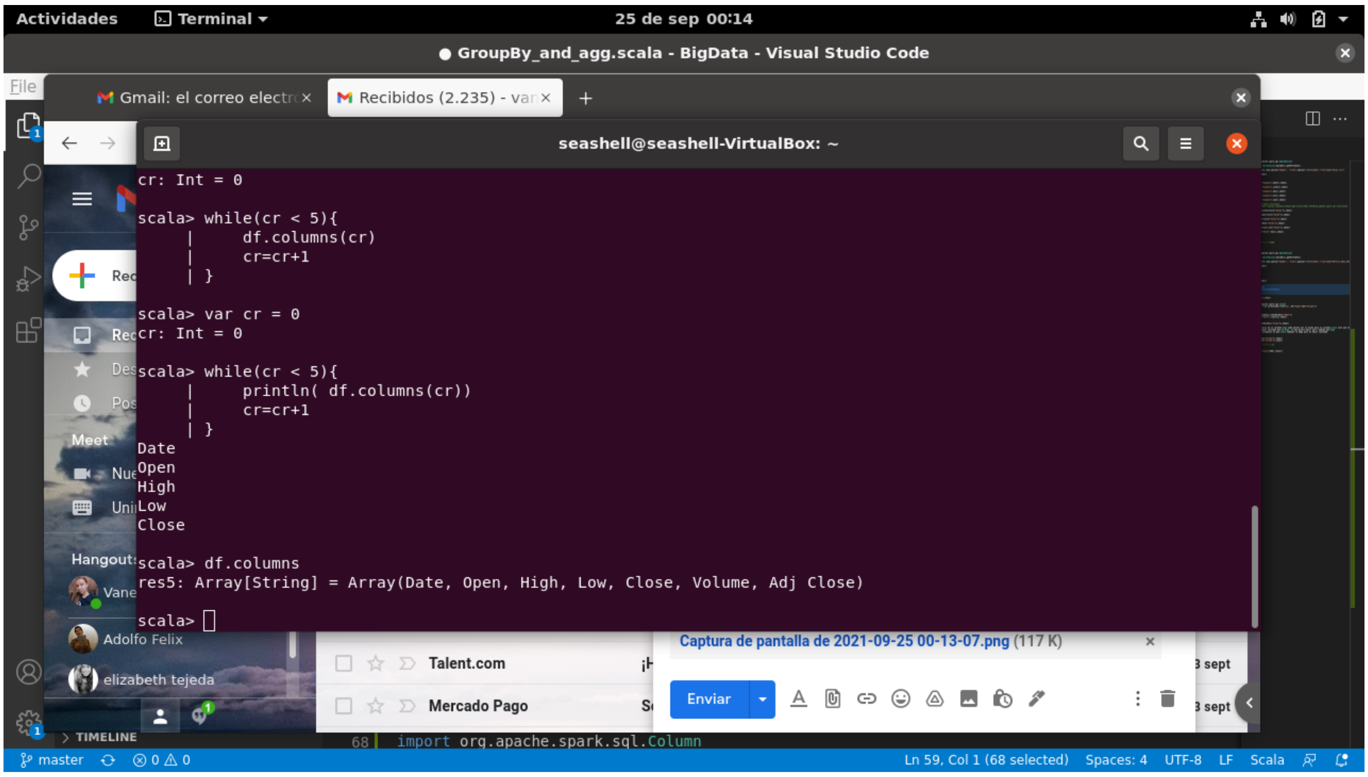


```
scala> df.printSchema()
root
 |-- Date: timestamp (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: integer (nullable = true)
 |-- Adj Close: double (nullable = true)


scala> df
res47: org.apache.spark.sql.DataFrame = [Date: timestamp, Open: double ... 5 more fields]

scala> 
```

3-.Names columns

Here only we want see the column names, but we want see all columns on the DataFrame.

```
df.columns
```

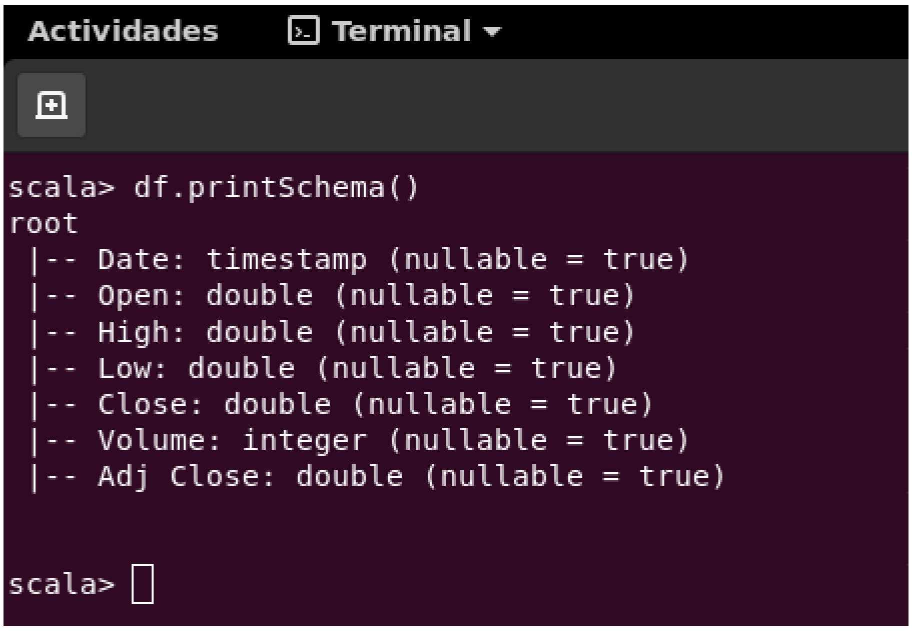


```
cr: Int = 0

scala> while(cr < 5){
     |     df.columns(cr)
     |     cr=cr+1
     | }

scala> var cr = 0
cr: Int = 0

scala> while(cr < 5){
     |     println( df.columns(cr))
     |     cr=cr+1
     | }
Date
Open
High
Low
Close

scala> df.columns
res5: Array[String] = Array(Date, Open, High, Low, Close, Volume, Adj Close)

scala> 
```

4-.Schema

For know thw schema only need the dataFrame and the next reserverd word, is for can you know the structure and the types of each column

```
df.printSchema()
```

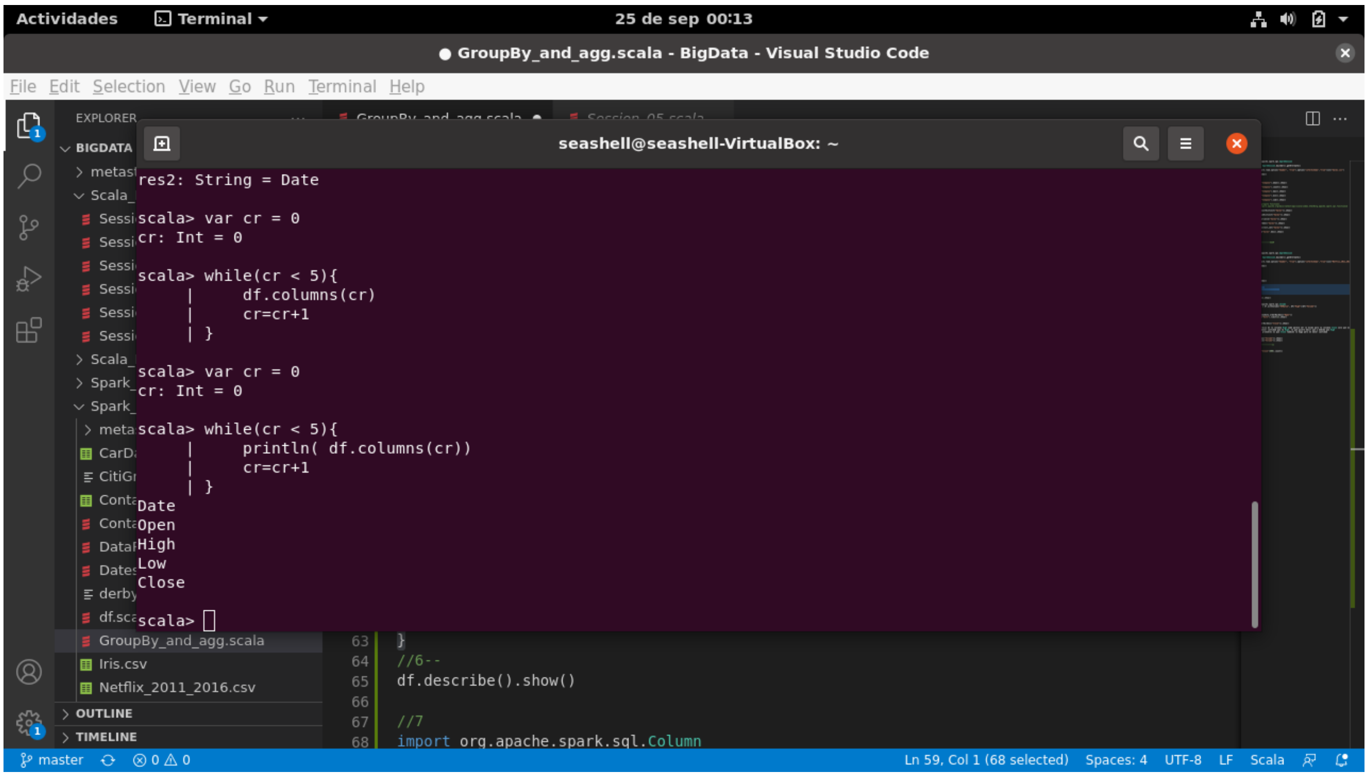


```
scala> df.printSchema()
root
 |-- Date: timestamp (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: integer (nullable = true)
 |-- Adj Close: double (nullable = true)


scala> 
```

5-Print first 5 columns

For that need create a variable inthis case cr and i say is equals to 0, then use while for create a bucle ever cr is less than 5, print the columns and increase cr if i don't do that is a infinite bucle

```
var cr = 0
while(cr < 5){
    println( df.columns(cr))
    cr=cr+1
}
```


Actividades
Terminal
25 de sep 00:13
GroupBy_and_agg.scala - BigData - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
BIGDATA
seashell@seashell-VirtualBox: ~
GroupBy_and_agg.scala
Iris.csv
Netflix_2011_2016.csv
OUTLINE
TIMELINE
master
Ln 59, Col 1 (68 selected) Spaces: 4 UTF-8 LF Scala

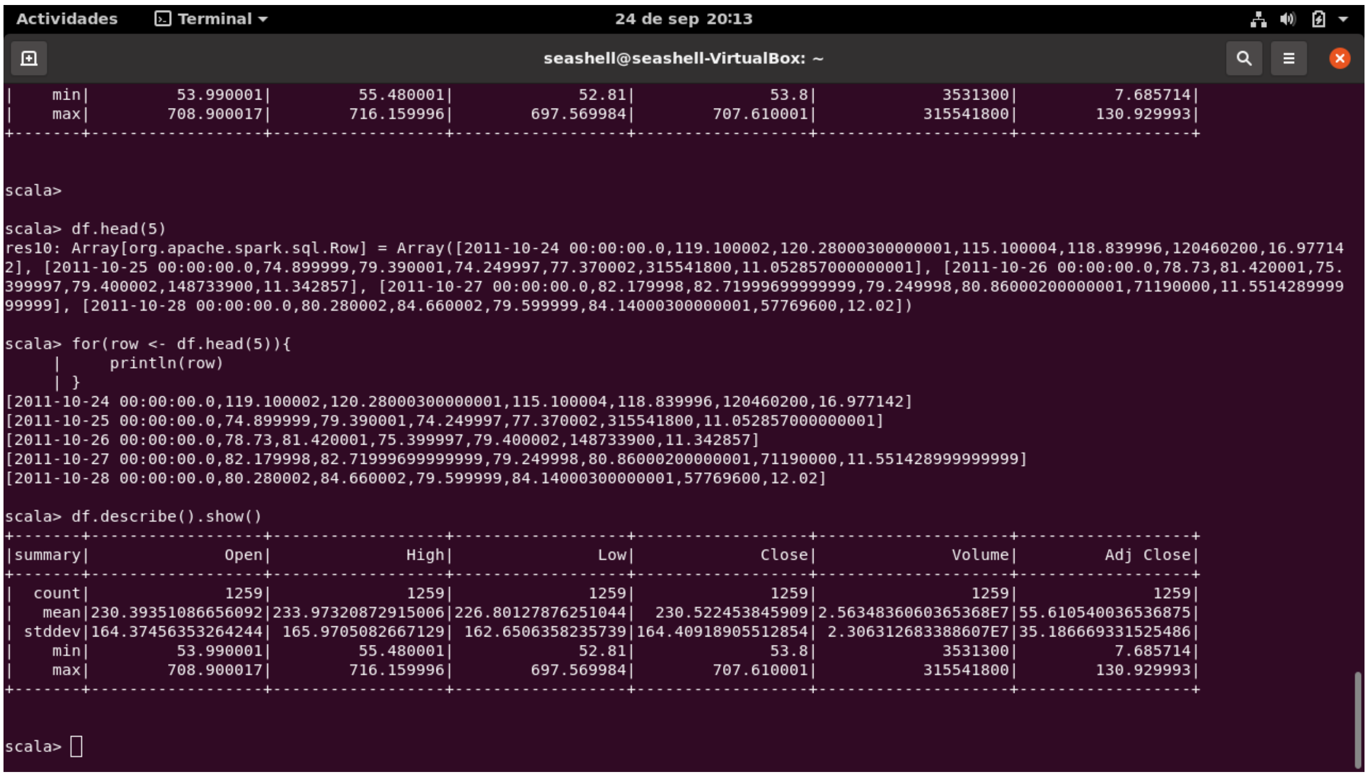
```
res2: String = Date

scala> var cr = 0
cr: Int = 0

scala> while(cr < 5){
     |     df.columns(cr)
     |     cr=cr+1
     | }

scala> var cr = 0
cr: Int = 0

scala> while(cr < 5){
     |     println( df.columns(cr))
     |     cr=cr+1
     | }
Date
Open
High
Low
Close

scala>
```

```
}
//6--
df.describe().show()

//7
import org.apache.spark.sql.Column
```

6-.Uses describe ()

That comand is for knows more informations about the dataFrame, statistical data

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

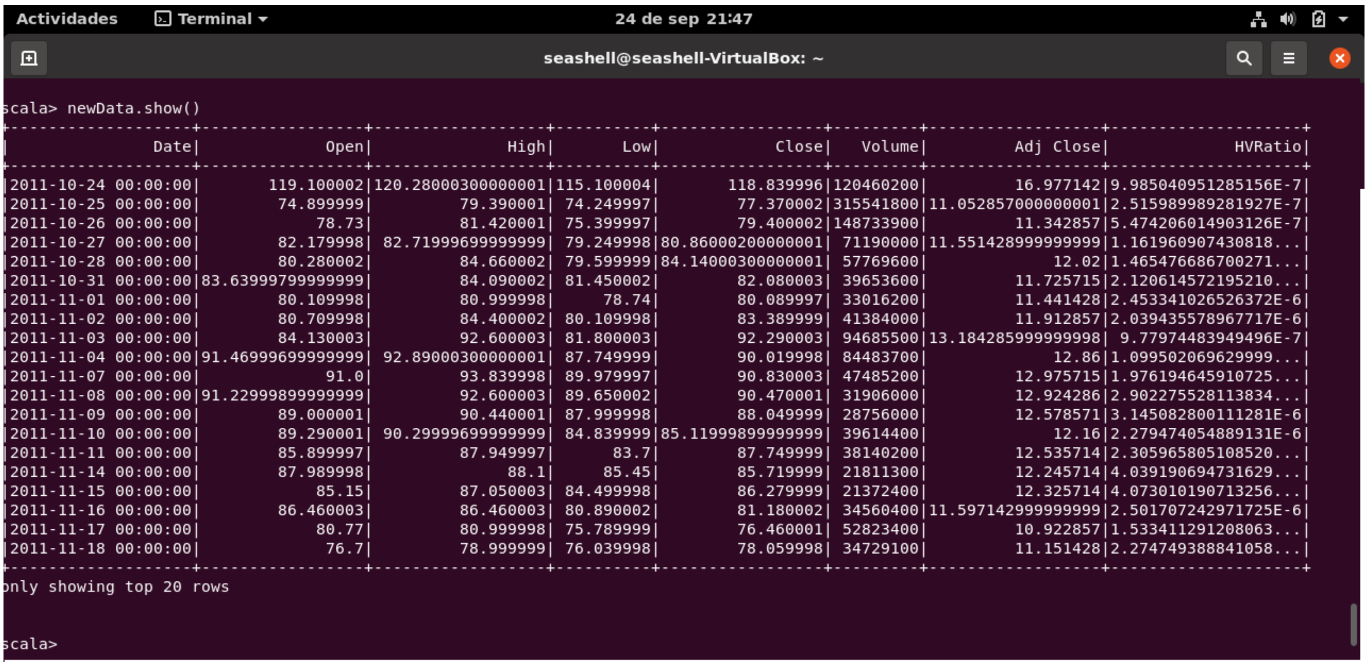


```
|    min|         53.990001|         55.480001|             52.81|              53.8|             3531300|          7.685714|
|    max|        708.900017|        716.159996|        697.569984|        707.610001|           315541800|        130.929993|
+-------+------------------+------------------+------------------+------------------+--------------------+------------------+


scala>

scala> df.head(5)
res10: Array[org.apache.spark.sql.Row] = Array([2011-10-24 00:00:00.0,119.100002,120.28000300000001,115.100004,118.839996,120460200,16.97714
2], [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 00:00:00.0,82.179998,82.71999699999999,79.249998,80.86000200000001,71190000,11.5514289999
99999], [2011-10-28 00:00:00.0,80.280002,84.660002,79.599999,84.14000300000001,57769600,12.02])

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 00:00:00.0,74.899999,79.390001,74.249997,77.370002,315541800,11.052857000000001]
[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]

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|
+-------+------------------+------------------+------------------+------------------+--------------------+------------------+


scala> 
```

7-.Create new DataFrame with new column

We need create a new dataFrame for can to do some modification so here to make a new column with the relationship of column High and Volume

```
val newData = df.withColumn("HVRatio", df("High")/df("Volume"))
newData.show()
```

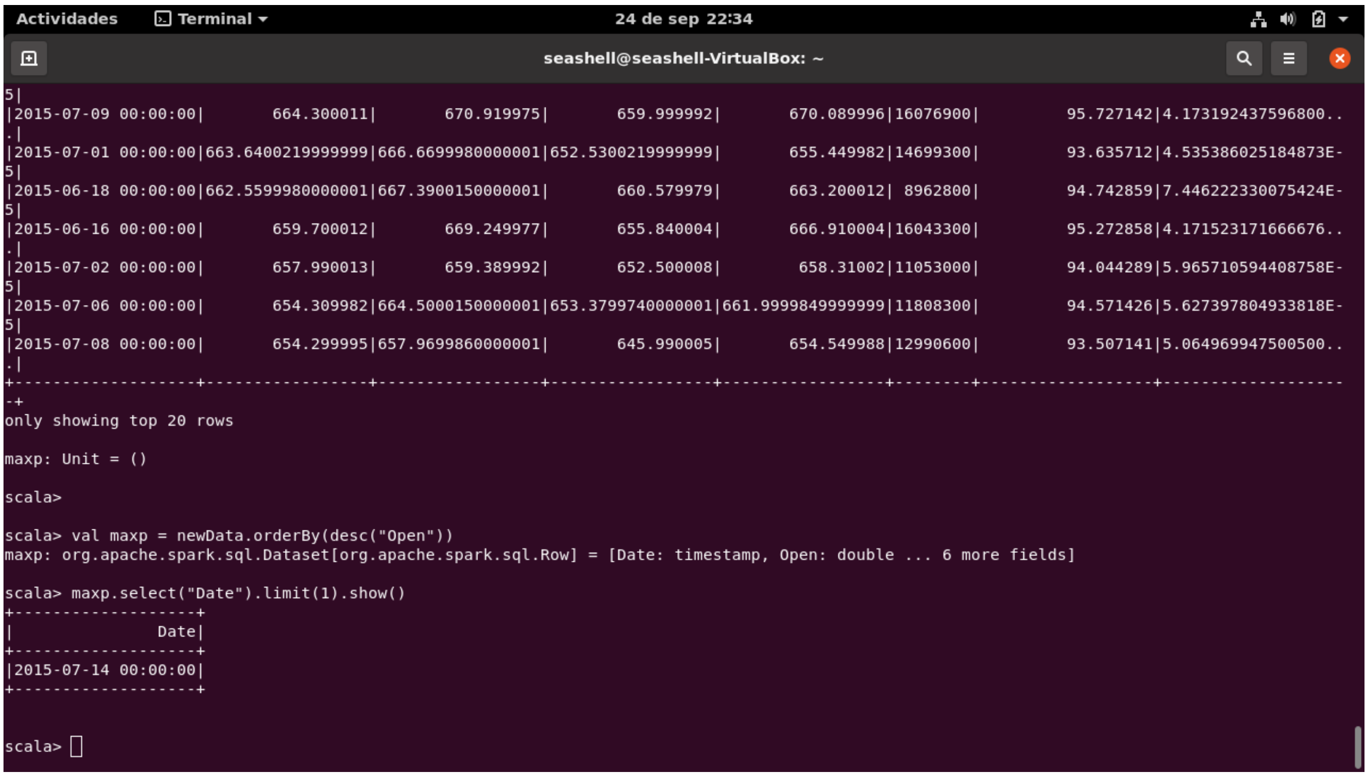


```
scala> newData.show()
+-------------------+-----------------+------------------+----------+-----------------+---------+------------------+--------------------+
|               Date|             Open|              High|       Low|            Close|   Volume|         Adj Close|             HVRatio|
+-------------------+-----------------+------------------+----------+-----------------+---------+------------------+--------------------+
|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


scala>
```

8-.Max Open

We need know the date of the maxium data, so first we order the column Open and save in maxp then select Date of de maxp but only the first row

```
val maxp = newData.orderBy(desc("Open"))
maxp.select("Date").limit(1).show()
```



```
5|
|2015-07-09 00:00:00|       664.300011|       670.919975|       659.999992|       670.089996|16076900|         95.727142|4.173192437596800..
.|
|2015-07-01 00:00:00|663.6400219999999|666.6699980000001|652.5300219999999|       655.449982|14699300|         93.635712|4.535386025184873E-
5|
|2015-06-18 00:00:00|662.5599980000001|667.3900150000001|       660.579979|       663.200012| 8962800|         94.742859|7.446222330075424E-
5|
|2015-06-16 00:00:00|       659.700012|       669.249977|       655.840004|       666.910004|16043300|         95.272858|4.171523171666676..
.|
|2015-07-02 00:00:00|       657.990013|       659.389992|       652.500008|        658.31002|11053000|         94.044289|5.965710594408758E-
5|
|2015-07-06 00:00:00|       654.309982|664.5000150000001|653.3799740000001|661.9999849999999|11808300|         94.571426|5.627397804933818E-
5|
|2015-07-08 00:00:00|       654.299995|657.9699860000001|       645.990005|       654.549988|12990600|         93.507141|5.064969947500500..
.|
+-------------------+-----------------+-----------------+-----------------+-----------------+--------+------------------+-------------------
-+
only showing top 20 rows

maxp: Unit = ()

scala>

scala> val maxp = newData.orderBy(desc("Open"))
maxp: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Date: timestamp, Open: double ... 6 more fields]

scala> maxp.select("Date").limit(1).show()
+-------------------+
|               Date|
+-------------------+
|2015-07-14 00:00:00|
+-------------------+


scala> 
```

9-.Meaning Close in DataFrame

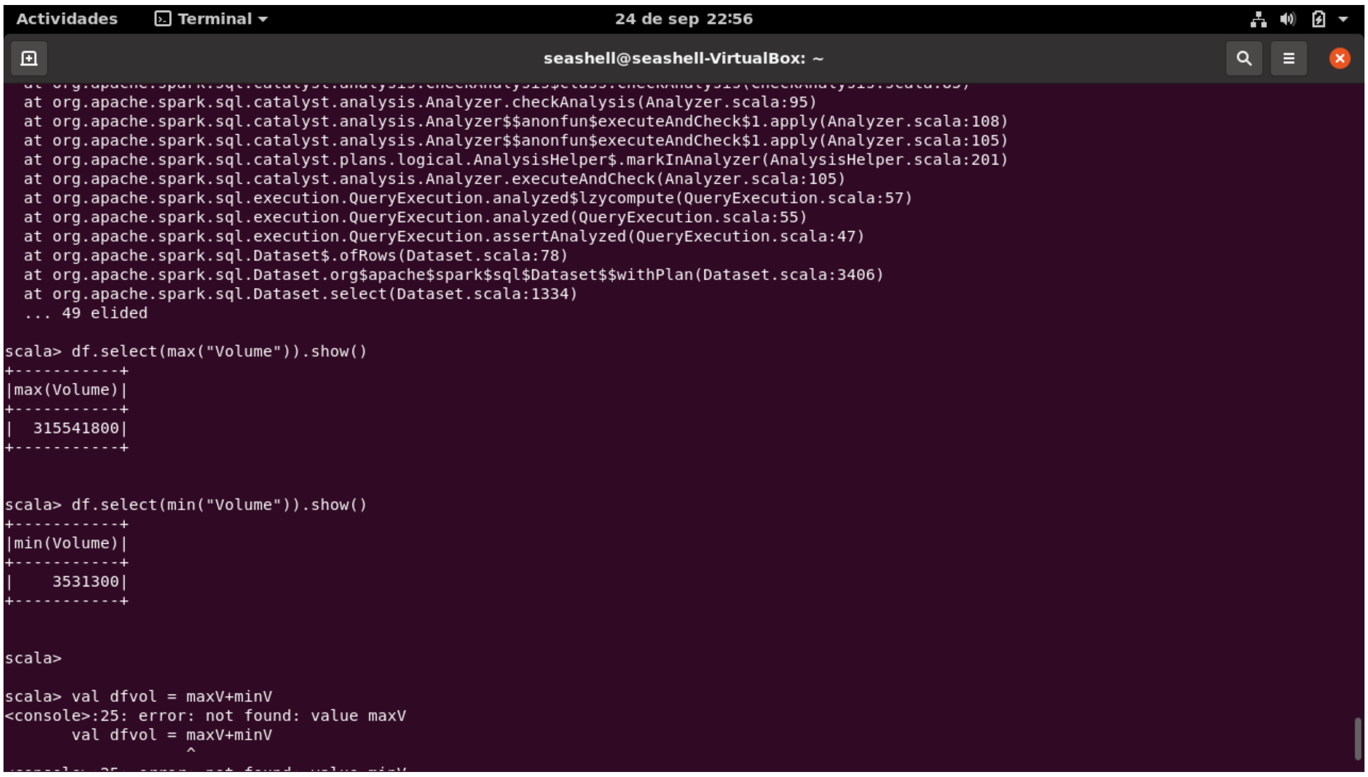
When the price of the High column goes up it seems to be the same for the Close column only that it is always less than High, which means that as High it goes up, the most probable thing is that Close will also do it but in less quantity

```
newData.orderBy(desc("Close")).show()
```

10-.Maximum and minimum of Volume

This is only to know thw first row the most big and the most lowest, and oly select the volume and your minium or maximun

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



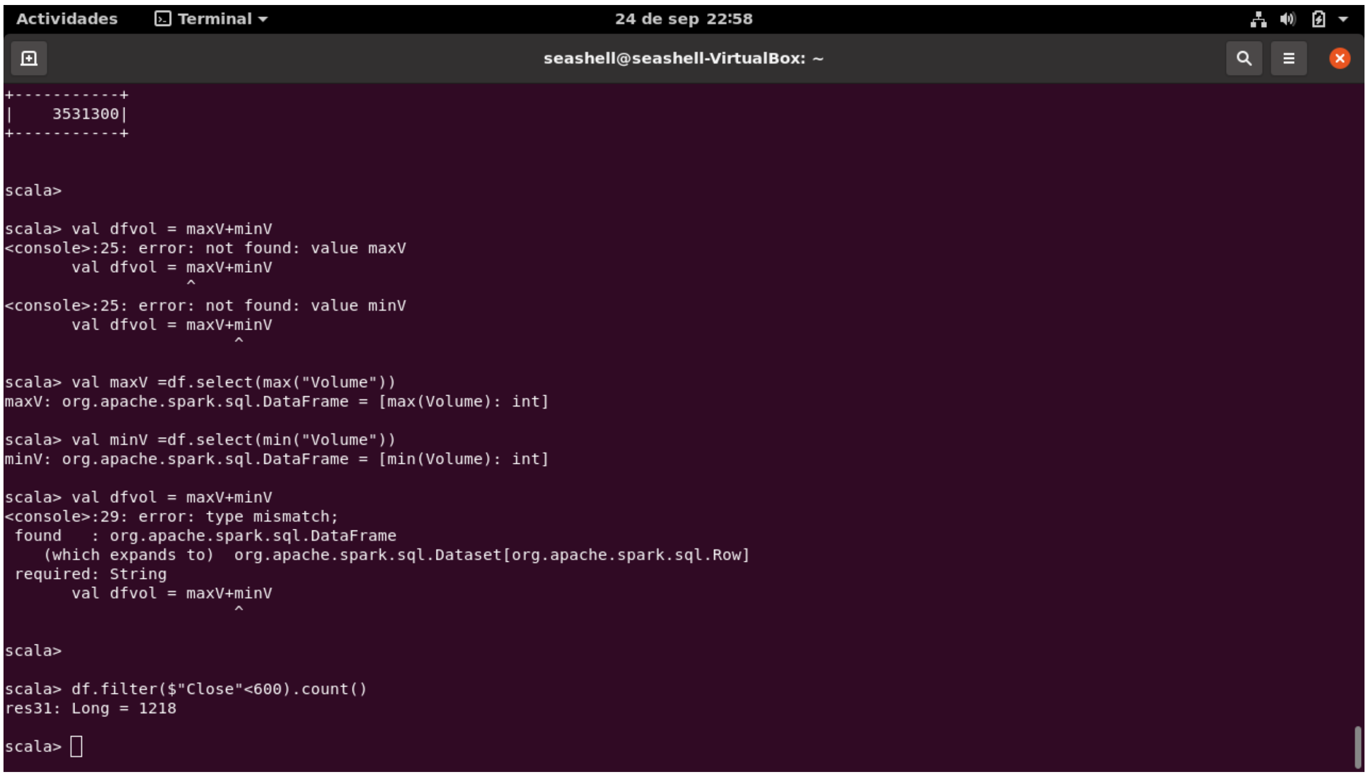
```
  at org.apache.spark.sql.catalyst.analysis.Analyzer.checkAnalysis(Analyzer.scala:95)
  at org.apache.spark.sql.catalyst.analysis.Analyzer$$anonfun$executeAndCheck$1.apply(Analyzer.scala:108)
  at org.apache.spark.sql.catalyst.analysis.Analyzer$$anonfun$executeAndCheck$1.apply(Analyzer.scala:105)
  at org.apache.spark.sql.catalyst.plans.logical.AnalysisHelper$.markInAnalyzer(AnalysisHelper.scala:201)
  at org.apache.spark.sql.catalyst.analysis.Analyzer.executeAndCheck(Analyzer.scala:105)
  at org.apache.spark.sql.execution.QueryExecution.analyzed$lzycompute(QueryExecution.scala:57)
  at org.apache.spark.sql.execution.QueryExecution.analyzed(QueryExecution.scala:55)
  at org.apache.spark.sql.execution.QueryExecution.assertAnalyzed(QueryExecution.scala:47)
  at org.apache.spark.sql.Dataset$.ofRows(Dataset.scala:78)
  at org.apache.spark.sql.Dataset.org$apache$spark$sql$Dataset$$withPlan(Dataset.scala:3406)
  at org.apache.spark.sql.Dataset.select(Dataset.scala:1334)
  ... 49 elided

scala> df.select(max("Volume")).show()
+-----------+
|max(Volume)|
+-----------+
|  315541800|
+-----------+


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


scala>

scala> val dfvol = maxV+minV
<console>:25: error: not found: value maxV
       val dfvol = maxV+minV
                   ^
```

11-.With Scala/Spark $ resolve the next

A-.With Scala/Spark $ resolve the next

Need to know the data less than numbesr 600 and cout that

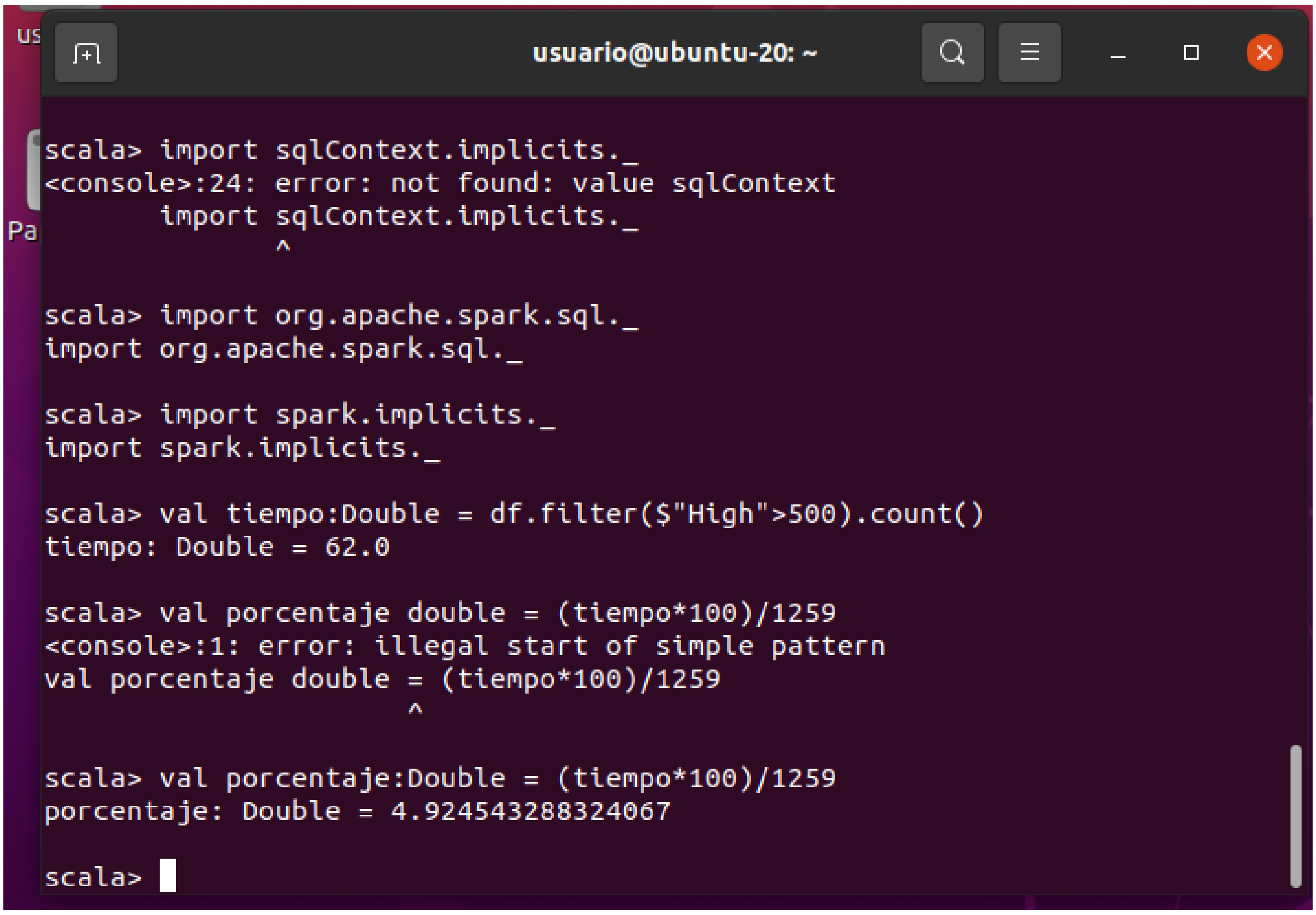
```
df.filter($"Close"<600).count()
```



```
+-----------+
|    3531300|
+-----------+


scala>

scala> val dfvol = maxV+minV
<console>:25: error: not found: value maxV
       val dfvol = maxV+minV
                   ^
<console>:25: error: not found: value minV
       val dfvol = maxV+minV
                        ^

scala> val maxV =df.select(max("Volume"))
maxV: org.apache.spark.sql.DataFrame = [max(Volume): int]

scala> val minV =df.select(min("Volume"))
minV: org.apache.spark.sql.DataFrame = [min(Volume): int]

scala> val dfvol = maxV+minV
<console>:29: error: type mismatch;
 found   : org.apache.spark.sql.DataFrame
    (which expands to)  org.apache.spark.sql.Dataset[org.apache.spark.sql.Row]
 required: String
       val dfvol = maxV+minV
                        ^

scala>

scala> df.filter($"Close"<600).count()
res31: Long = 1218

scala> 
```

B-.We need to know what is the percentage of time in this question

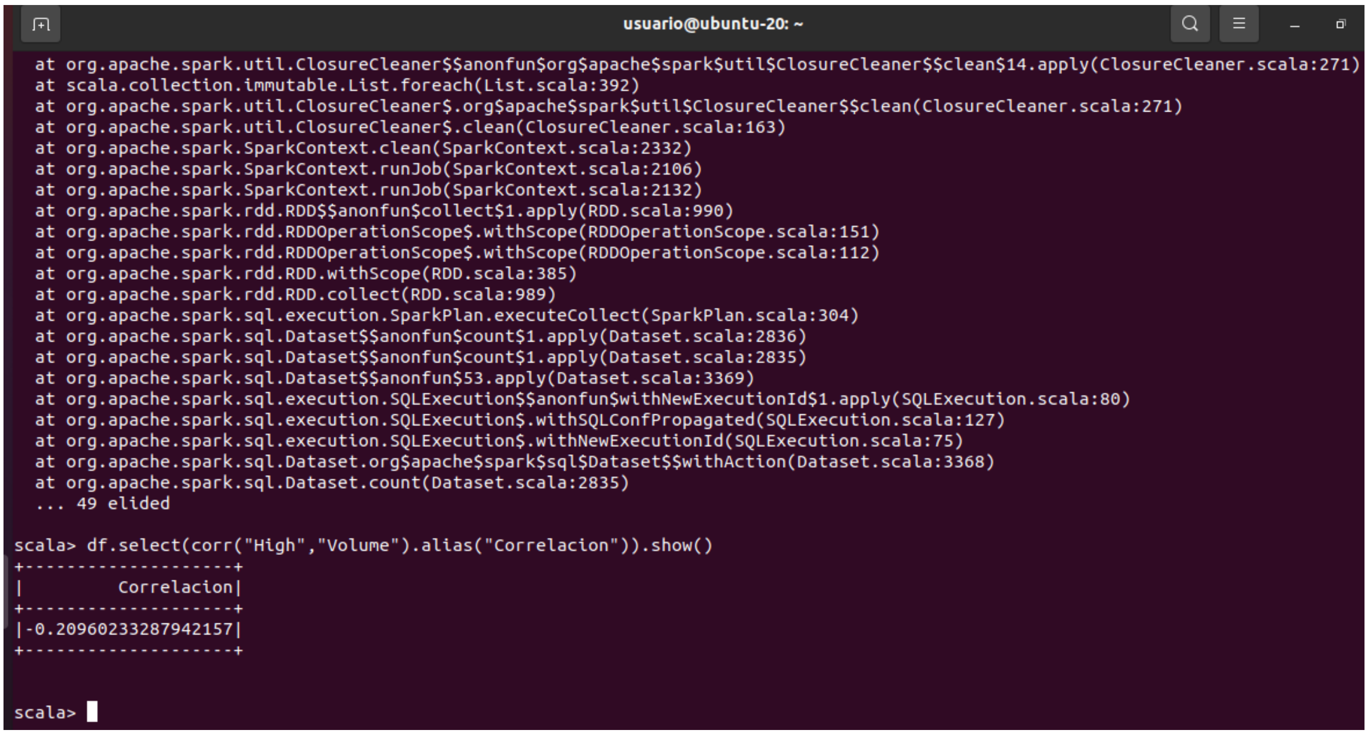
```scala
val tiempo:Double = df.filter($"High">500).count()
val porcentaje:Double = (tiempo*100)/1259
```



```
scala> import sqlContext.implicits._
<console>:24: error: not found: value sqlContext
       import sqlContext.implicits._
              ^

scala> import org.apache.spark.sql._
import org.apache.spark.sql._

scala> import spark.implicits._
import spark.implicits._

scala> val tiempo:Double = df.filter($"High">500).count()
tiempo: Double = 62.0

scala> val porcentaje double = (tiempo*100)/1259
<console>:1: error: illegal start of simple pattern
val porcentaje double = (tiempo*100)/1259
                ^

scala> val porcentaje:Double = (tiempo*100)/1259
porcentaje: Double = 4.924543288324067

scala> 
```

C-.We need to know what is the correlation of high and volumen

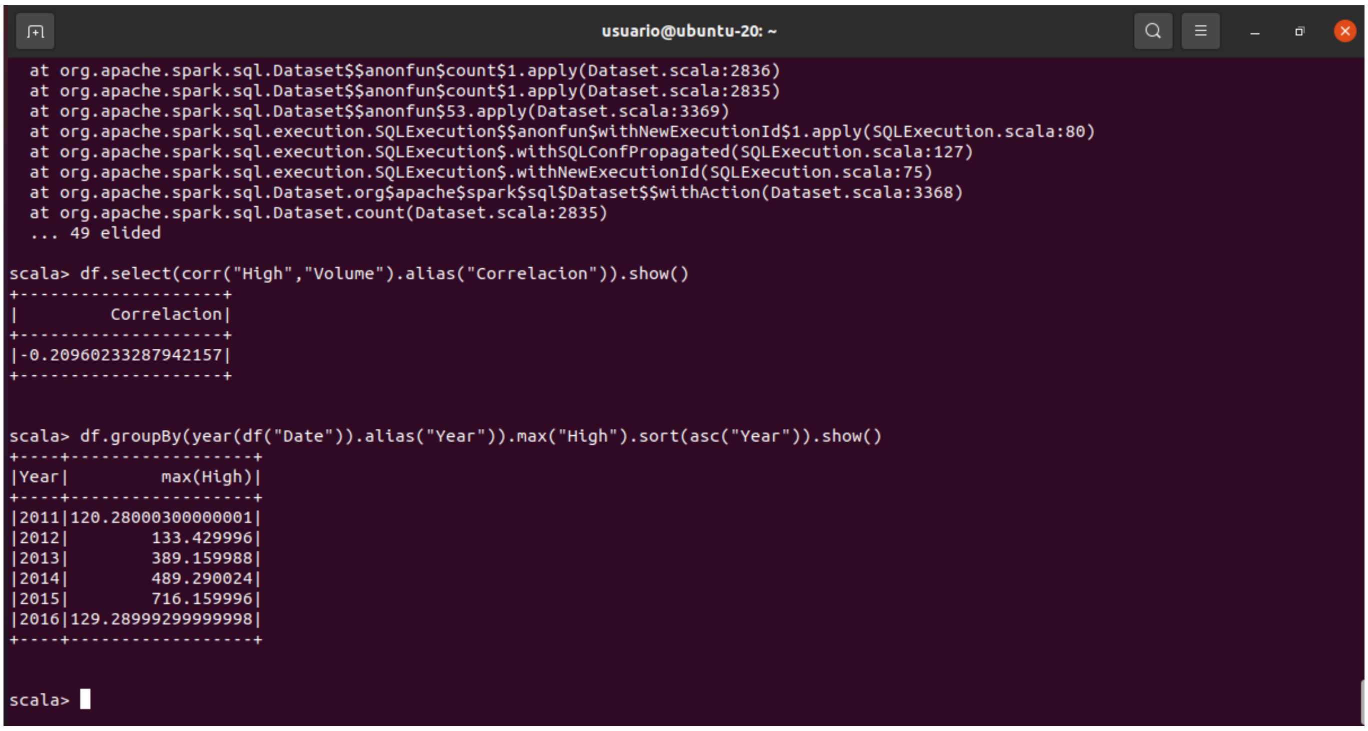
```
df.select(corr("High","Volume").alias("Correlacion")).show()
```



```
  at org.apache.spark.util.ClosureCleaner$$anonfun$org$apache$spark$util$ClosureCleaner$$clean$14.apply(ClosureCleaner.scala:271)
  at scala.collection.immutable.List.foreach(List.scala:392)
  at org.apache.spark.util.ClosureCleaner$.org$apache$spark$util$ClosureCleaner$$clean(ClosureCleaner.scala:271)
  at org.apache.spark.util.ClosureCleaner$.clean(ClosureCleaner.scala:163)
  at org.apache.spark.SparkContext.clean(SparkContext.scala:2332)
  at org.apache.spark.SparkContext.runJob(SparkContext.scala:2106)
  at org.apache.spark.SparkContext.runJob(SparkContext.scala:2132)
  at org.apache.spark.rdd.RDD$$anonfun$collect$1.apply(RDD.scala:990)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112)
  at org.apache.spark.rdd.RDD.withScope(RDD.scala:385)
  at org.apache.spark.rdd.RDD.collect(RDD.scala:989)
  at org.apache.spark.sql.execution.SparkPlan.executeCollect(SparkPlan.scala:304)
  at org.apache.spark.sql.Dataset$$anonfun$count$1.apply(Dataset.scala:2836)
  at org.apache.spark.sql.Dataset$$anonfun$count$1.apply(Dataset.scala:2835)
  at org.apache.spark.sql.Dataset$$anonfun$53.apply(Dataset.scala:3369)
  at org.apache.spark.sql.execution.SQLExecution$$anonfun$withNewExecutionId$1.apply(SQLExecution.scala:80)
  at org.apache.spark.sql.execution.SQLExecution$.withSQLConfPropagated(SQLExecution.scala:127)
  at org.apache.spark.sql.execution.SQLExecution$.withNewExecutionId(SQLExecution.scala:75)
  at org.apache.spark.sql.Dataset.org$apache$spark$sql$Dataset$$withAction(Dataset.scala:3368)
  at org.apache.spark.sql.Dataset.count(Dataset.scala:2835)
  ... 49 elided

scala> df.select(corr("High","Volume").alias("Correlacion")).show()
+-------------------+
|        Correlacion|
+-------------------+
|-0.20960233287942157|
+-------------------+


scala> 
```

D-.We need to know which are the maximun for each year

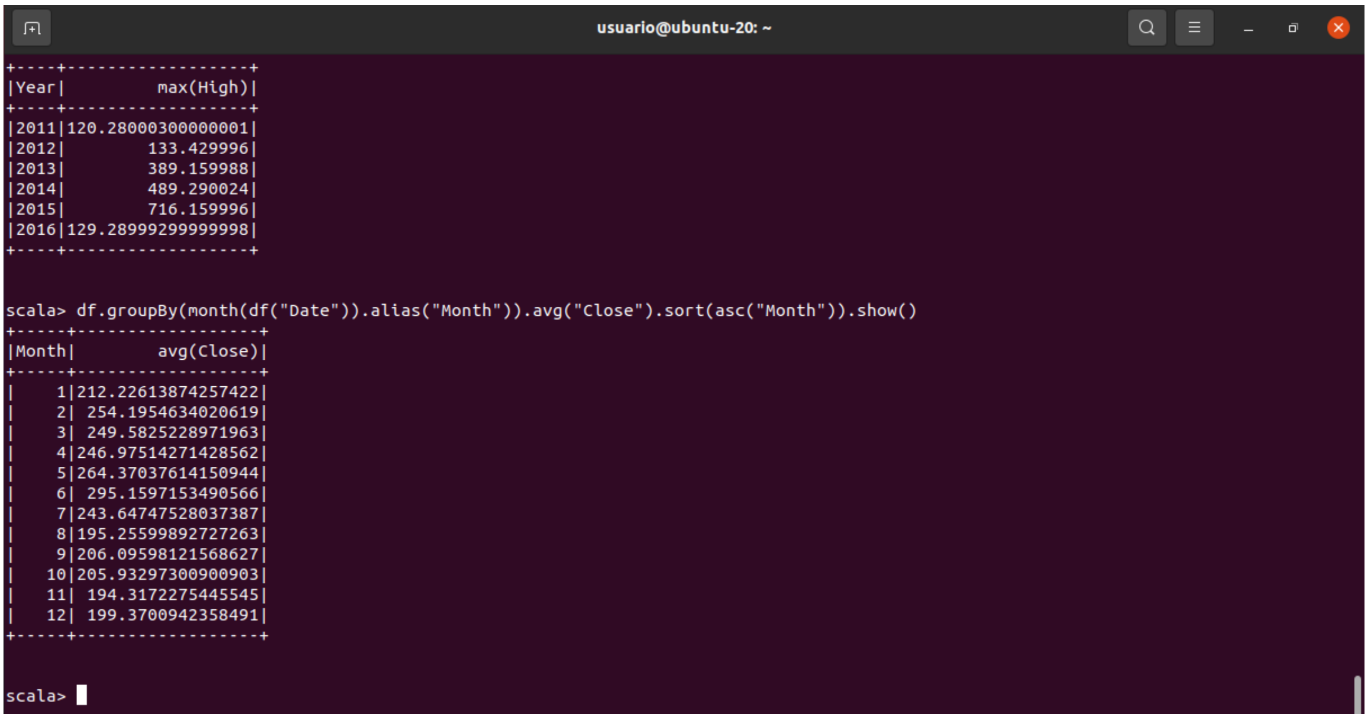
```
df.groupBy(year(df("Date")).alias("Year")).max("High").sort(asc("Year")).show()
```



```
  at org.apache.spark.sql.Dataset$$anonfun$count$1.apply(Dataset.scala:2836)
  at org.apache.spark.sql.Dataset$$anonfun$count$1.apply(Dataset.scala:2835)
  at org.apache.spark.sql.Dataset$$anonfun$53.apply(Dataset.scala:3369)
  at org.apache.spark.sql.execution.SQLExecution$$anonfun$withNewExecutionId$1.apply(SQLExecution.scala:80)
  at org.apache.spark.sql.execution.SQLExecution$.withSQLConfPropagated(SQLExecution.scala:127)
  at org.apache.spark.sql.execution.SQLExecution$.withNewExecutionId(SQLExecution.scala:75)
  at org.apache.spark.sql.Dataset.org$apache$spark$sql$Dataset$$withAction(Dataset.scala:3368)
  at org.apache.spark.sql.Dataset.count(Dataset.scala:2835)
  ... 49 elided

scala> df.select(corr("High","Volume").alias("Correlacion")).show()
+--------------------+
|         Correlacion|
+--------------------+
|-0.20960233287942157|
+--------------------+


scala> df.groupBy(year(df("Date")).alias("Year")).max("High").sort(asc("Year")).show()
+----+------------------+
|Year|         max(High)|
+----+------------------+
|2011|120.28000300000001|
|2012|        133.429996|
|2013|        389.159988|
|2014|        489.290024|
|2015|        716.159996|
|2016|129.28999299999998|
+----+------------------+


scala> 
```

E-.This question deals with knowing the average close for each month

```
df.groupBy(month(df("Date")).alias("Month")).avg("Close").sort(asc("Month")).show()
```



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


scala> df.groupBy(month(df("Date")).alias("Month")).avg("Close").sort(asc("Month")).show()
+-----+------------------+
|Month|        avg(Close)|
+-----+------------------+
|    1|212.22613874257422|
|    2| 254.1954634020619|
|    3| 249.5825228971963|
|    4|246.97514271428562|
|    5|264.37037614150944|
|    6| 295.1597153490566|
|    7|243.64747528037387|
|    8|195.25599892727263|
|    9|206.09598121568627|
|   10|205.93297300900903|
|   11| 194.3172275445545|
|   12| 199.3700942358491|
+-----+------------------+


scala> 
```