



**EDUCACIÓN**  
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO  
NACIONAL DE MÉXICO®



# **Tecnológico Nacional De México**

**Instituto Tecnológico De Tijuana**

**Subdirección Académica**

**Departamento de Sistemas y Computación**

**Semestre Enero - Junio 2022**

**Ingeniería Informática**

**Datos Masivos**

**Práctica Evaluatoria**

**Unidad 1**

**Israel López Pablo**

**No.17210585**

**Perez Ortega Victoria Valeria**

**No.18210718**

**JOSE CHRISTIAN ROMERO HERNANDEZ**

**Tijuana, B.C. a 22 de Marzo de 2022.**



## TECNOLÓGICO NACIONAL DE MÉXICO

### INSTITUTO TECNOLÓGICO DE TIJUANA

#### SUBDIRECCIÓN ACADÉMICA

Departamento de Sistemas y Computación

#### EXAMEN

Carrera: Ingeniería En Sistemas Computacionales/ Tecnologías de la información/ Informática  
Período: **Febrero-Junio 2022** Materia: Datos Masivos Grupo: BDD-1704SC9C Salón: Unidad  
(es) a evaluar: Unidad 1 Tipo de examen: Practico Fecha: Catedrático: José Christian Romero  
Hernandez Firma del maestro: Calificación:

Alumnos: Israel Lopez Palo y Perez Ortega Victoria Valeria No. Control: 17210585, 18210718

### Instrucciones

**Responder las siguientes preguntas con Spark DataFrames y Scala utilizando el "CSV" Netflix\_2011\_2016.csv que se encuentra en la carpeta de spark-dataframes.**

1. Comienza una simple sesión Spark.

Para iniciar sesión en Spark, primero abrimos el cmd como administrador, ejecutamos Spark con spark-shell e inmediatamente importamos una biblioteca para poder iniciar sesión

```
import org.apache.spark.sql.SparkSession
```

Declaramos la variable session y usamos la librería para crear una nueva sesión

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

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

scala> val SparkSession = SparkSession.builder().getOrCreate()
<console>:24: error: recursive value SparkSession needs type
    val SparkSession = SparkSession.builder().getOrCreate()
                        ^

scala> val session = SparkSession.builder().getOrCreate()
session: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@387ef6a7
```



2. Cargue el archivo Netflix Stock CSV, haga que Spark infiera los tipos de datos.

**val netflix =**

**spark.read.option("header","true").option("inferSchema","true").csv("Netflix\_2011\_2016.csv")**

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

3. ¿Cuáles son los nombres de las columnas?

Mandamos a llamar a nuestra variable netflix que con ese nombre cargamos el archivo CSV de Netflix Stock, y lo que queremos ver son las columnas

**netflix.columns**

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

4. ¿Cómo es el esquema?

Mandamos a llamar a nuestra variable netflix que con ese nombre cargamos el archivo Netflix Stock CSV, mandamos a imprimir el esquema que es el que queremos ver

**netflix.printSchema()**

```
scala> netflix.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. Imprime las primeras 5 columnas.

Mandamos a llamar a nuestra variable de netflix que con ese nombre cargamos el archivo CSV de Netflix Stock y seleccionamos las primeras 5 columnas

```
netflix.select($"Date", $"Open", $"High", $"Low", $"Close").show()
```

```
scala> netflix.select($"Date", $"Open", $"High", $"Low", $"Close").show()
```

Date	Open	High	Low	Close
2011-10-24 00:00:00	119.100002	120.28000300000001	115.100004	118.839996
2011-10-25 00:00:00	74.899999	79.390001	74.249997	77.370002
2011-10-26 00:00:00	78.73	81.420001	75.399997	79.400002
2011-10-27 00:00:00	82.179998	82.71999699999999	79.249998	80.86000200000001
2011-10-28 00:00:00	80.280002	84.660002	79.599999	84.14000300000001
2011-10-31 00:00:00	83.63999799999999	84.090002	81.450002	82.080003
2011-11-01 00:00:00	80.109998	80.999998	78.74	80.089997
2011-11-02 00:00:00	80.709998	84.400002	80.109998	83.389999
2011-11-03 00:00:00	84.130003	92.600003	81.800003	92.290003
2011-11-04 00:00:00	91.46999699999999	92.89000300000001	87.749999	90.019998
2011-11-07 00:00:00	91.0	93.839998	89.979997	90.830003
2011-11-08 00:00:00	91.22999899999999	92.600003	89.650002	90.470001
2011-11-09 00:00:00	89.000001	90.440001	87.999998	88.049999
2011-11-10 00:00:00	89.290001	90.29999699999999	84.839999	85.11999899999999
2011-11-11 00:00:00	85.899997	87.949997	83.7	87.749999
2011-11-14 00:00:00	87.989998	88.1	85.45	85.719999
2011-11-15 00:00:00	85.15	87.050003	84.499998	86.279999
2011-11-16 00:00:00	86.460003	86.460003	80.890002	81.180002
2011-11-17 00:00:00	80.77	80.999998	75.789999	76.460001
2011-11-18 00:00:00	76.7	78.999999	76.039998	78.059998

only showing top 20 rows

6. Usa describe () para aprender sobre el DataFrame.

```
netflix.describe().show()
```

```
scala> netflix.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. 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

Declaramos una nueva variable llamada `netflixnew`

```
var netflixnew = netflix.withColumn("HV Ratio",  
    netflix("High")/netflix("Volume"))
```

```
scala> var netflixnew = netflix.withColumn("HV Ratio", netflix("High")/netflix("Volume"))  
netflixnew: org.apache.spark.sql.DataFrame = [Date: timestamp, Open: double ... 6 more fields]
```

8. ¿Qué día tuvo el pico más alto en la columna “Open”?

Declaramos una variable llamada `columnnday` y del archivo de `netflix` tomamos los datos de la columna `day`

```
val columnnday = netflix.withColumn("Day",dayofmonth(netflix("Date")))
```

Declaramos una variable llamada `netflixdaycolumn` y vamos a agrupar todos los valores máximos del día

```
val netflixdaycolumn = columnnday.groupBy("Day").max()
```

llamamos a la variable que declaramos para mostrar la tabla

```
netflixdaycolumn.show()
```

mandamos a llamar a la variable que declaramos y seleccionamos el día máximo de la columna `maxopen`

```
netflixdaycolumn.select($"Day",$"max(Open)").show()
```

```
scala> val columnnday = netflix.withColumn("Day",dayofmonth(netflix("Date")))  
columnnday: org.apache.spark.sql.DataFrame = [Date: timestamp, Open: double ... 6 more fields]  
  
scala> val netflixdaycolumn = columnnday.groupBy("Day").max()  
netflixdaycolumn: org.apache.spark.sql.DataFrame = [Day: int, max(Open): double ... 6 more fields]
```



```
scala> netflixdaycolumn.select($"Day",$"max(Open)").show()
+---+-----+
|Day|      max(Open)|
+---+-----+
| 31|      430.259995|
| 28|      628.000008|
| 26|      667.100021|
| 27|      617.000023|
| 12| 664.4100269999999|
| 22|      665.29998|
|  1| 663.6400219999999|
| 13| 686.6900019999999|
| 16|      659.700012|
|  6|      654.309982|
|  3|      624.700012|
| 20|      617.500008|
|  5|      624.500008|
| 19|      673.700012|
| 15|      649.999977|
|  9|      664.300011|
| 17|      665.930008|
|  4|      618.649994|
|  8|      654.299995|
| 23|      674.350014|
+---+-----+
only showing top 20 rows
```

9. ¿Cuál es el significado de la columna Cerrar “Close” en el contexto de información financiera, explíquelo que no hay que codificar nada?

La columna cerrar significa que la empresa finalizó el día o en este caso que netflix finalizó el día

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

Del archivo netflix mandamos a llamar el valor maximo y minimo de la

```
scala> netflix.select(max("Volume"), min("Volume")).show()
+-----+-----+
|max(Volume)|min(Volume)|
+-----+-----+
| 315541800|  3531300|
+-----+-----+
```



## 11. Con Scala/Spark Syntax \$ responda lo siguiente:

### a. ¿Cuántos días estuvo la columna "Close" menos de \$600?

Para esto solo necesitamos un filtro "cerrar" con una condición y esa condición es menor.

```
val result =netflix.filter($"Close" < 600 ).count()
```

#### Result:

aquí podemos ver el resultado

```
scala> val result =netflix.filter($"Close" < 600 ).count()  
result: Long = 1218
```

### b. ¿Qué porcentaje de veces la columna "High" superó los \$500?

Este es el caso contrario que la pregunta anterior.

```
val result = (netflix.filter( $"High" >  
500).count()*1.0/netflix.count())*100
```

#### Result:

Aquí podemos ver el resultado.

```
scala> val result = (netflix.filter( $"High" > 500).count()*1.0/netflix.count())*100  
result: Double = 4.924543288324067
```

```
scala> |
```

### c. ¿Cuál es la correlación de Pearson entre la columna "High" y la columna "Volume"?

El coeficiente de correlación de Pearson es una prueba que mide la relación estadística entre dos variables continuas. así que en este caso específicamente



estamos usando para dos columnas High y Volume

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

### Result:

Este es el resultado de usar la correlación de Pearson.

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

### d. ¿Cuál es el máximo de la columna "High" por año?

En este caso necesitamos obtener un máximo por lo que usamos una función max para obtener el resultado

```
netflix.select(max($"High")).show()
```

### Result:

Estos son los resultados

```
scala> netflix.select(max($"High")).show()
+-----+
| max(High)|
+-----+
|716.159996|
+-----+
```





## e. ¿Cuál es el promedio de columna “Close” para cada mes del calendario?

Este es diferente a todos los demás ya que debemos obtener ciertos valores los cuales necesitan ser modificados varias veces, cómo agregar una nueva columna que almacene los meses y usar métodos como obtener el mes, usar el promedio y agrupar para obtener el resultado deseado.

```
val meses = netflix.withColumn("Meses",month(netflix("Date")))

val porcentajemes=
meses.select($"Meses",$"Close").groupBy("Meses").mean()

porcentajemes.select($"Meses",$"avg(Close)").show()
```

### Result:

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

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

scala> porcentajemes.select($"Meses",$"avg(Close)").show()
+-----+-----+
|Meses|      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|
+-----+-----+
```

URL: <https://www.youtube.com/watch?v=02E6qq6Crpc>