Tecnológico Nacional De México Instituto Tecnológico De Tijuana

Subdirección Académica

Departamento De Sistemas Y Computación

SEMESTRE:

Enero – junio 2020

CARRERA:

Ing. Tecnologías De Información Y Comunicación

NOMBRE DEL TRABAJO:

Ejercicios De DF

UNIDAD A EVALUAR:

Unidad 1

NOMBRE DEL ALUMNO:

Garcia Bautista Ana Laura # 15210793

Enciso Maldonado Aileen Yurely #15210329

MAESTRO (A):

Romero Hernández José Christian

EJERCICIOS DE 20 FUNCION DE DF

```
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder().getOrCreate()
val df = spark.read.option("header", "true").option("inferSchema","true")csv("CitiGroup2006
_2008")

//1 print schema
df.printSchema()

//2 show the dataset
df.show()

//3 show the columns the dataset has
df.columns

//4 show the volume data
df.select("Volume").show()

//5 show the first record of the dataset
df.first()

//6 show the 10 records that head the data set
df.head(10)

//7 show interesting facts about the data
df.describe()

//8 count the total data the data set has
df.count()

//9 order the data
df.sort()

//10 show data that is between the conditions
df.filter($"Close" < 490 && $"low" < 300).show()

//11 draw the correlation
df.select(corr("High", "Low")).show()

//12 sum all data of high
df.select(sum("High")).show()

//13 mean of data
df.select(mean("Low")).show()
```

```
//14 max of data
df.select(max("High")).show()

//15 min of data
df.select(min("Low")).show()

//16 variance of data
df.select(variance("Low")).show()

//17 look for an exact data in the column
df.filter($"High" === 487.0).show()

//18 count the values that meet the condition
df.filter($"High" > 480).count()

//19 sample for months
df.select(month(df("Date"))).show()

//20 sample for years
df.select(year(df("Date"))).show()
```

EJERCICIO DE 20 FUNCIONES DE DF

```
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder().getOrCreate()
val df = spark.read.option("header", "true").option("inferSchema","true")csv("CitiGroup2006
_2008")

//1.sumDistinct
df.select(sumDistinct("Sales")).show()

//2.last
df.select(last("Company")).show() //last data in company

//3.first
df.select(first("Person")).show() first data in person

//4.var_pop
df.select(var_pop("Sales")).show()

//5.avg
df.select(avg("Sales")).show()

//6.collect_list
df.select(collect_list("Sales")).show()
```

```
//7.var_samp
df.select(var_samp("Sales")).show()

//8.sum
df.select(sum("Sales")).show()

//9.stddev_pop
df.select(stddev_pop("Sales")).show()

//10.skewness
df.select(skewness("Sales")).show()

//11.min
df.select(min("Sales")).show()

//12.kurtosis
df.select(kurtosis("Sales")).show()

//13.collect_set
df.select(collect_set("Sales")).show()

//14.approx_count_distinct
df.select(approx_count_distinct("Company")).show()

//15.mean
df.select(mean("Sales")).show()

//16 return the first column of the dataframe
df.first

//17 Returns the dataframe columns
df.columns

//18 Add a column that derives from the high and Volume column
val df2 = df.withColumn("HV Ratio", df("High")+df("Volume"))

//19 Choose the volume column min
df.select(min("Volume")).show()

//20 Choose the volume column max
df.select(max("Volume")).show()
```

EJERCICIO DE 20 FUNCIONES DE DF

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

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

val df = spark.read.option("header", "true").option("inferSchema","true")csv("Sales.csv")

//Company, Person, Sales


  //Group By on single column
  df.groupBy("Person").count().show(false)
  df.groupBy("Person").avg("Sales").show(false)
  df.groupBy("Person").sum("Sales").show(false)
  df.groupBy("Person").min("Sales").show(false)
  df.groupBy("Person").max("Sales").show(false)
  df.groupBy("Person").mean("Sales").show(false)

  //GroupBy on multiple columns
  df.groupBy("Company", "Person")
    .sum("Sales","Sales")
    .show(false)
  df.groupBy("department","state")
    .avg("salary","bonus")
    .show(false)
  df.groupBy("department","state")
    .max("salary","bonus")
    .show(false)
  df.groupBy("department","state")
    .min("salary","bonus")
    .show(false)
  df.groupBy("department","state")
    .mean("salary","bonus")
    .show(false)

    //Running Filter
  df.groupBy("department","state")
    .sum("salary","bonus")
    .show(false)

     //using agg function
  df.groupBy("Company")
    .agg(
      sum("Person").as("sum_salary"),
      avg("Person").as("avg_salary"),
      sum("Sales").as("sum_bonus"),
```

```
      max("Sales").as("max_bonus"))
    .show(false)

 df.groupBy("Company")
    .agg( sum("Person").as("sum_salary")).show(false)




  df.groupBy("department")
    .agg(
      sum("salary").as("sum_salary"),
      avg("salary").as("avg_salary"),
      sum("bonus").as("sum_bonus"),
      stddev("bonus").as("stddev_bonus"))
    .where(col("sum_bonus") > 50000)
    .show(false)
}
```

GROUPBY 1 - Aileen

```
//Usaremos este Spark DataFrame para ejecutar groupBy () en columnas de "departamento" y ca
lcular agregados como mínimo,
//máximo, promedio, salario total para cada grupo usando las funciones de agregado min (),
max () y sum ()
//respectivamente. y finalmente, también veremos cómo agrupar y agregar en múltiples column
as
import spark.implicits._
 val simpleData = Seq(("CristianR","Sales","NY",90000,34,10000),
   ("Aileen","Sales","NY",86000,56,20000),
   ("Laura","Sales","CA",81000,30,23000),
   ("Alexis","Finance","CA",90000,24,23000),
   ("Rubensito","Finance","CA",99000,40,24000),
   ("Afedito","Finance","NY",83000,36,19000),
   ("Cynthia","Finance","NY",79000,53,15000),
   ("Irving","Marketing","CA",80000,25,18000),
   ("Ramon","Marketing","NY",91000,50,21000)
 )
 val df = simpleData.toDF("employee_name","department","salary","state","age","bonus")
 df.show()
```

GROUPBY 2 - Aileen

```
Verificación de tipo de dato
// Secuencia de elementos donut donde cada elemento de la secuencia es de tipo String

println("Step 1: How to initialize a Sequence of donuts")
val donuts: Seq[String] = Seq("Plain Donut", "Strawberry Donut", "Glazed Donut")
println(s"Elements of donuts = $donuts"
```

GROUPBY 3 - Aileen

```
// método groupBy para agrupar elementos en la secuencia de donas por el primer carácter de
 cada donut

println("\nStep 2: How to group elements in a sequence using the groupBy function")
val donutsGroup: Map[Char, Seq[String]] = donuts.groupBy(_.charAt(0))
println(s"Group elements in the donut sequence by the first letter of the donut name = $don
utsGroup")
```

GROUPBY 4 - Aileen

```
Ejemplo de creación de clase con objeto
//clase de caso para representar objetos donut.

println("\nStep 3: How to create a case class to represent Donut objects")
case class Donut(name: String, price: Double)
```

GROUPBY 5 - Aileen

```
//Clase de caso Donut del Paso 3 y crear una secuencia de elementos donut de tipo Donut.

println("\nStep 4: How to create a Sequence of type Donut")
val donuts2: Seq[Donut] = Seq(Donut("Plain Donut", 1.5), Donut("Strawberry Donut", 2.0), Do
nut("Glazed Donut", 2.5))
println(s"Elements of donuts2 = $donuts2")
```

GROUPBY 6 - Aileen

```
 //agrupar los objetos donut representados por la clase de caso Donut del Paso 3 por su pro
piedad de nombre usando el método groupBy



println(s"\nStep 5: How to group case classes donut objects by the name property")
```

```
val donutsGroup2: Map[String, Seq[Donut]] = donuts2.groupBy(_.name)
println(s"Group element in the sequence of type Donut grouped by the donut name = $donutsGr
oup2"
```