LAB-6

AIM: Import Purchases.txt Dataset from Kaggle

- Instead of breaking the sales down by store, give us a sales breakdown by product category across all of our stores. What is the value of total sales for the following categories?
  - Toys
  - Consumer Electronics

- Find the monetary value for the highest individual sale for each separate store. What are the values for the following stores?
  - ✓ Reno
  - ✓ Toledo
  - ✓ Chandler

Step 1: First Importing The data to Spark Shell and creating the dataframe

1. **Define the path to the purchases.txt file**
   val filePath = "D:\\purchases.txt"

2. **Define the case class to represent each purchase record**
   case class Purchase(Date: String, Time: String, Store_Name: String, Product_Category: String, Sale_Value: Double, Payment_Method: String)

3. **Load the data from the text file into an RDD (Resilient Distributed Dataset)**
   val purchasesRDD = sc.textFile(filePath)

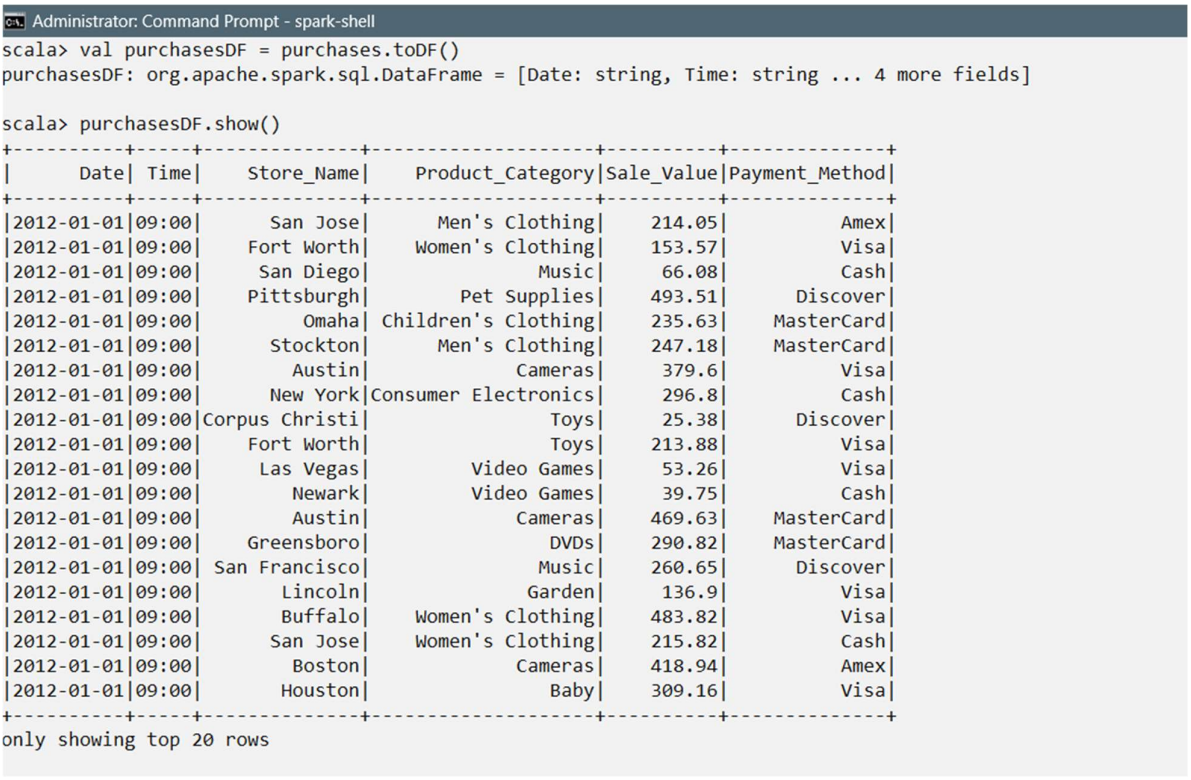
4. **Transform the RDD by splitting each line based on the tab delimiter and mapping it to the case class "Purchase"**
   val purchases = purchasesRDD.map(line => { val cols = line.split("\t") Purchase(cols(0), cols(1), cols(2), cols(3), cols(4).toDouble, cols(5))})

5. **Import implicit conversions for converting RDDs to DataFrames**
   import spark.implicits._

6. **Convert the RDD of `Purchase` case class objects to a DataFrame**
   val purchasesDF = purchases.toDF()

7. **Show the first few rows of the DataFrame to verify that the data is loaded correctly**
   purchasesDF.show()

Output of Step 1: Showing top 20 rows in output



```
Administrator: Command Prompt - spark-shell
scala> val purchasesDF = purchases.toDF()
purchasesDF: org.apache.spark.sql.DataFrame = [Date: string, Time: string ... 4 more fields]

scala> purchasesDF.show()
+----------+-----+--------------+--------------------+----------+--------------+
|      Date| Time|    Store_Name|    Product_Category|Sale_Value|Payment_Method|
+----------+-----+--------------+--------------------+----------+--------------+
|2012-01-01|09:00|      San Jose|      Men's Clothing|    214.05|          Amex|
|2012-01-01|09:00|    Fort Worth|    Women's Clothing|    153.57|          Visa|
|2012-01-01|09:00|     San Diego|               Music|     66.08|          Cash|
|2012-01-01|09:00|    Pittsburgh|        Pet Supplies|    493.51|      Discover|
|2012-01-01|09:00|         Omaha| Children's Clothing|    235.63|    MasterCard|
|2012-01-01|09:00|      Stockton|      Men's Clothing|    247.18|    MasterCard|
|2012-01-01|09:00|        Austin|             Cameras|     379.6|          Visa|
|2012-01-01|09:00|      New York|Consumer Electronics|     296.8|          Cash|
|2012-01-01|09:00|Corpus Christi|                Toys|     25.38|      Discover|
|2012-01-01|09:00|    Fort Worth|                Toys|    213.88|          Visa|
|2012-01-01|09:00|     Las Vegas|         Video Games|     53.26|          Visa|
|2012-01-01|09:00|        Newark|         Video Games|     39.75|          Cash|
|2012-01-01|09:00|        Austin|             Cameras|    469.63|    MasterCard|
|2012-01-01|09:00|    Greensboro|                DVDs|    290.82|    MasterCard|
|2012-01-01|09:00| San Francisco|               Music|    260.65|      Discover|
|2012-01-01|09:00|       Lincoln|              Garden|     136.9|          Visa|
|2012-01-01|09:00|       Buffalo|    Women's Clothing|    483.82|          Visa|
|2012-01-01|09:00|      San Jose|    Women's Clothing|    215.82|          Cash|
|2012-01-01|09:00|        Boston|             Cameras|    418.94|          Amex|
|2012-01-01|09:00|       Houston|                Baby|    309.16|          Visa|
+----------+-----+--------------+--------------------+----------+--------------+
only showing top 20 rows
```

Step 2: Filter the DataFrame for 'Toys' and 'Consumer Electronics' category and calculate the total sales for 'Sale_Value'.

1. **Group the data by 'Product_Category', calculate the sum of 'Sale_Value', and order by total sales in descending order.**

   val categorySales = purchasesDF.groupBy("Product_Category")
   Group by the 'Product_Category' column

   .sum("Sale_Value")
   Calculate the total sales for each product category

   .orderBy($"sum(Sale_Value)".desc)
   Sort the results in descending order of total sales

   categorySales.show()
   Display the grouped and sorted results

1. **Filter the DataFrame for 'Toys' category and calculate the total sales for 'Sale_Value'.**

   val toysSales = purchasesDF.filter($"Product_Category" === "Toys") .agg(Map("Sale_Value" -> "sum"))

   .first.get(0)

2. **Print the total sales for 'Toys' category.**

   println(s"Total Sales for Toys: $$${toysSales}")

3. **Filter the DataFrame for 'Consumer Electronics' category and calculate the total sales for 'Sale_Value'.**

   val electronicsSales = purchasesDF.filter($"Product_Category" === "Consumer Electronics") .agg(Map("Sale_Value" -> "sum")).first.get(0)

4. **Print the total sales for 'Consumer Electronics' category.**

   println(s"Total Sales for Consumer Electronics: $$${electronicsSales}")

Output of Step 2:

```
scala> val toysSales = purchasesDF.filter($"Product_Category" === "Toys") .agg(Map("Sale_Value" -> "sum")).first.get(0)
toysSales: Any = 5.7463477109999925E7

scala> println(s"Total Sales for Toys: $$${toysSales}")
Total Sales for Toys: $5.7463477109999925E7

scala> val electronicsSales = purchasesDF.filter($"Product_Category" === "Consumer Electronics") .agg(Map("Sale_Value" -> "sum")).first.get(0)
electronicsSales: Any = 5.745237412999981E7

scala> println(s"Total Sales for Consumer Electronics: $$${electronicsSales}")
Total Sales for Consumer Electronics: $5.745237412999981E7
```

Step 3: Code for Calculating the Highest Individual Sale for Each Store

1. Group the data by 'Store_Name', calculate the maximum 'Sale_Value' for each store, and order by the highest sale in descending order.

val highestSalePerStore = purchasesDF.groupBy("Store_Name") .max("Sale_Value")

Calculate the maximum sale value for each store

.orderBy($"max(Sale_Value)".desc)

Sort the results in descending order of the maximum sale value

highestSalePerStore.show()

Display the grouped and sorted results

2. Filter the DataFrame for 'Reno' store and calculate the highest sale value.

```
val renoSale = purchasesDF.filter($"Store_Name" === "Reno").agg(Map("Sale_Value" -> "max"))
 .first.get(0)
```

3. Print the highest sale value for the 'Reno' store.

```
println(s"Highest Sale in Reno: $$${renoSale}")
```

4. Filter the DataFrame for 'Toledo' store and calculate the highest sale value.

```
val toledoSale = purchasesDF.filter($"Store_Name" === "Toledo")
.agg(Map("Sale_Value" -> "max"))
 .first.get(0)
```

5. Print the highest sale value for the 'Toledo' store.

```
println(s"Highest Sale in Toledo: $$${toledoSale}")
```

6. Filter the DataFrame for 'Chandler' store and calculate the highest sale value.

```
val chandlerSale = purchasesDF.filter($"Store_Name" === "Chandler")
 .agg(Map("Sale_Value" -> "max"))
 .first.get(0)
```

7. Print the highest sale value for the 'Chandler' store.

```
println(s"Highest Sale in Chandler: $$${chandlerSale}")
```

Output of Step 3 :