Retail Sales Analysis

Santanu Adhikary	2320846
A. N. V. Rohit	2320408
Sanmith Chetan N	2320907
E. Divya Sree	2320625
Shashank K Hulmani	2320613
Vivechan Gowda G.C	2320755

Contents

1.Introduction --- 03

a) Pyspark

2.Business Requirements --- 04

3.Overview of the project --- 05

4.Implementation ---06

5.Data Analysis & Visualisation ---34

6.Conclusion ---45

Introduction

Data has become an essential part of our daily lives in today's digital age. From searching for a product on e-commerce platforms to placing an order and receiving it at home, we are constantly generating and consuming data. Today's data-driven world generates data from various sectors like retail, automobile, finance, technology, aviation, food, media, etc. These data sets are in different forms and massive in quantity.

Extracting valuable insights from these data sets requires using modern tools and technologies to handle the data's scale and complexity. With the right tools, organizations can gain a deeper understanding of their operations and make data-driven decisions to improve their performance.

In this Project, we will work on a case study, and the data we will use is from the retail industry. The tool we use for this case study is PySpark and Databricks. To work on this case study, I am using the Databricks Community Edition.

Retail Data :

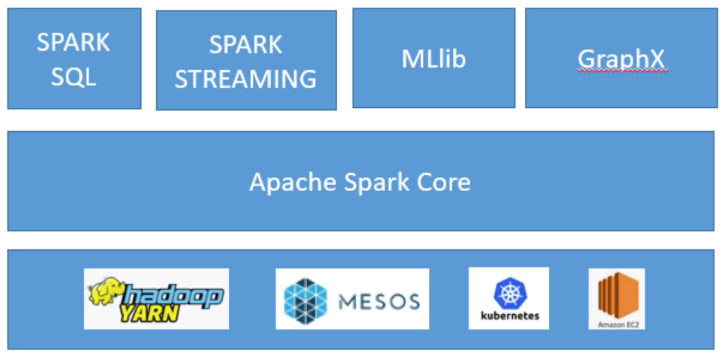
Retail data is information that a retail shop owner might collect to better their firm. This information gives merchants information about their customers, sales patterns, and inventories across the retail industry.

Retail data plays a crucial role in the decision-making process of retailers. By collecting and analyzing various forms of data, retailers can gain valuable insights into their business performance and make informed decisions to improve their bottom line. By using the retail data, we will see how a retail company can improve its operations and increase its profits with the help of data analysis. We are going to find the hidden insight from this data.

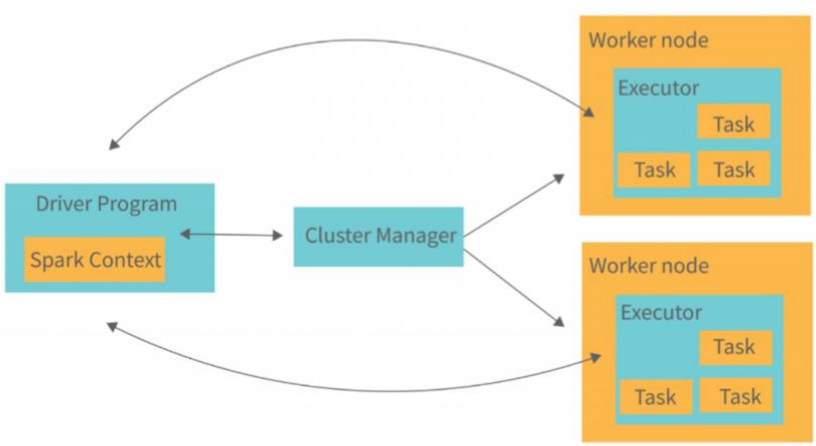
Pyspark :

PySpark is a Python API to support Python with Apache Spark. PySpark provides **Py4j library,** with the help of this library, Python can be easily integrated with Apache Spark. PySpark plays an essential role when it needs to work with a vast dataset or analyze them. This feature of PySpark makes it a very demanding tool among data engineers.

A large amount of data is generated offline and online. It is necessary to extract valuable information from the raw data.We require a more efficient tool to perform different types of operations on the big data. There are various tools to perform the multiple tasks on the huge dataset but these tools are not so appealing anymore. It is needed some scalable and flexible tools to crack big data and gain benefit from it.



Architechure Overview :



1. Driver Program
2. Cluster Manager
3. Worker Node

Business Requirements

Description:

Build a retail sales analytics platform to analyze sales data, customer behavior, and product trends.

Features:

Data Ingestion: Ingest sales data from various retail channels, POS systems, or databases.
Data Transformation: Clean, transform, and aggregate sales data to calculate metrics like total sales, average order value, and customer lifetime value.
Customer Segmentation: Segment customers based on purchase history, demographics, or behavior.
Trend Analysis: Identify product trends, seasonal patterns, and sales trends over time.

Challenges:

- Handling and joining multiple data sources.
- Efficiently aggregating and summarizing large volumes of sales data.
- Performing complex analytical queries to derive meaningful insights.

Implementation

We are taking some random data and we will create a dataframe from the provided dataset. We have six datasets with customer details, product order details, product details, and product category information.

```
import os
import pandas as pd
import numpy as np

from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession, SQLContext

from pyspark.sql.types import *
from pyspark.sql.window import Window


import pyspark.sql.functions as F
from pyspark.sql.functions import udf, col , count , date_format
```

For Visualisation :

```
# Visualization
import seaborn as sns
import matplotlib.pyplot as plt
```

We are going to create six data frames. Which contains the following information:-

1. **Customer Dataframe:** This dataframe contains information related to the customer. It has nine columns which are as follows:-

- **customer_id**: This column contains the id of the customer. Ex:- 1, 2, 3, etc.
- **customer_fname**: This column has the customer's first name details.
- **customer_lname**: This column has the customer's last name details.
- **customer_email**: This column includes the customer's email info.
- **customer_password**: This column has customer password information. It's encrypted.
- **customer_street**: This has customer address-related info, which is street in this case.
- **customer_city**: This has city-related information.
- **customer_state**: The state info of the customer.
- **customer_zipcode**: The zip code of the customer location.

Now we will create the schema for the customer dataframe.

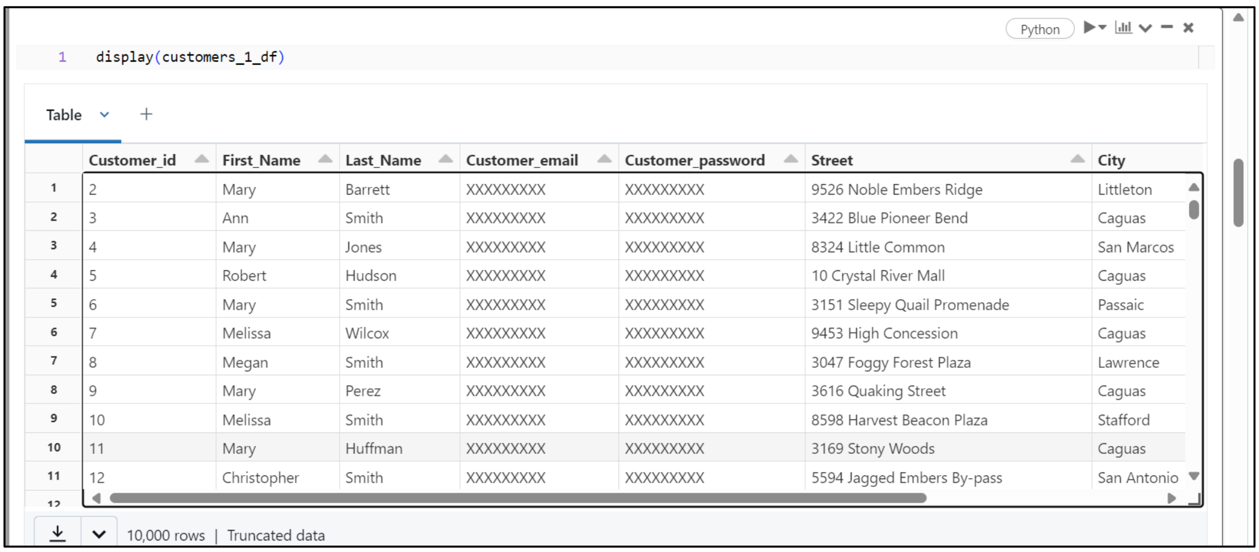
```
# define the schema for customer dataset
customers_1_schema=StructType(fields=[StructField("Customer_id",IntegerType(
),nullable=False),
                    StructField("First_Name",StringType(),nullable=True),
                    StructField("Last_Name",StringType(),nullable= True),
                    StructField("Customer_email",StringType(),nullable= True),
                    StructField("Customer_password",StringType(),nullable= True),
                    StructField("Street",StringType(),nullable= True),
                    StructField("City",StringType(),nullable= True),
                    StructField("State",StringType(),nullable= True),
                    StructField("Zipcode",IntegerType(),nullable= True),
                         ])
```

We will make the dataframe using the above schema.

Reading the data through a CSV file.

customers_1_df = spark.read.csv\
("dbfs:/FileStore/tables/customers_1.csv",header=True,schema=customers_1_schema)

To show the data frame the command is
display(customers_1_df)
The customer dataframe will look like this:



```
1  display(customers_1_df)
```

Table

	Customer_id	First_Name	Last_Name	Customer_email	Customer_password	Street	City
1	2	Mary	Barrett	XXXXXXXXX	XXXXXXXXX	9526 Noble Embers Ridge	Littleton
2	3	Ann	Smith	XXXXXXXXX	XXXXXXXXX	3422 Blue Pioneer Bend	Caguas
3	4	Mary	Jones	XXXXXXXXX	XXXXXXXXX	8324 Little Common	San Marcos
4	5	Robert	Hudson	XXXXXXXXX	XXXXXXXXX	10 Crystal River Mall	Caguas
5	6	Mary	Smith	XXXXXXXXX	XXXXXXXXX	3151 Sleepy Quail Promenade	Passaic
6	7	Melissa	Wilcox	XXXXXXXXX	XXXXXXXXX	9453 High Concession	Caguas
7	8	Megan	Smith	XXXXXXXXX	XXXXXXXXX	3047 Foggy Forest Plaza	Lawrence
8	9	Mary	Perez	XXXXXXXXX	XXXXXXXXX	3616 Quaking Street	Caguas
9	10	Melissa	Smith	XXXXXXXXX	XXXXXXXXX	8598 Harvest Beacon Plaza	Stafford
10	11	Mary	Huffman	XXXXXXXXX	XXXXXXXXX	3169 Stony Woods	Caguas
11	12	Christopher	Smith	XXXXXXXXX	XXXXXXXXX	5594 Jagged Embers By-pass	San Antonio

10,000 rows | Truncated data

Now we will create a Product dataframe.

2. **Product dataframe**: The product information contained in this dataframe. The six columns within it are as follows:-

- **product_id:** This column contains the product ids.
- **product_category_id:** This column help in finding the category of the product.
- **product_name**:- This column includes product names that we have in store.
- **product_description**: It consists of the product details.

- **product_price**: The product price is in this column.
- **product_image**: Product image URLs are present in this column.

First, we will create the schema for this dataframe. Which will look like this:-

```
products_schema = StructType(fields=[StructField('product_id',IntegerType(),
nullable=False),
    StructField('product_category_id', IntegerType(), nullable=False),
    StructField('product_name',StringType(), nullable=False),
    StructField('product_description',StringType(), nullable=False),
    StructField('product_price',FloatType(), nullable=False),
    StructField('product_image',StringType(), nullable=False)
    ])
```

We will make the dataframe using the above schema.

Reading the data through a CSV file.

```
products_df = spark.read.csv\
("dbfs:/FileStore/tables/products_2.csv",header=True,schema=products_schema)
```

To show the data frame the command is
display(products_df)
The product dataframe will look like this:-

```
display(products_df)
```

(1) Spark Jobs

Table

	product_id	product_category_id	product_name	product_description	product_price	product_
1	2	2	Under Armour Men's Highlight MC Football Clea	null	129.99	http://ima
2	3	2	Under Armour Men's Renegade D Mid Football Cl	null	89.99	http://ima
3	4	2	Under Armour Men's Renegade D Mid Football Cl	null	89.99	http://ima
4	5	2	Riddell Youth Revolution Speed Custom Footbal	null	199.99	http://ima
5	6	2	Jordan Men's VI Retro TD Football Cleat	null	134.99	http://ima
6	7	2	Schutt Youth Recruit Hybrid Custom Football H	null	99.99	http://ima
7	8	2	Nike Men's Vapor Carbon Elite TD Football Cle	null	129.99	http://ima
8	9	2	Nike Adult Vapor Jet 3.0 Receiver Gloves	null	50	http://ima
9	10	2	Under Armour Men's Highlight MC Football Clea	null	129.99	http://ima
10	11	2	Fitness Gear 300 lb Olympic Weight Set	null	209.99	http://ima
11	12	2	Under Armour Men's Highlight MC Alter Ego Fla	null	139.99	http://ima
12	13	2	Under Armour Men's Renegade D Mid Football Cl	null	89.99	http://ima

3. Categories DataFrame: This dataframe has a list of product categories. It has three columns product_id, product_category_id, and category_name.

- **category_name:** The categories to which a product may belong are in this column. The product name **"Under Armour Men's Highlight MC Football Clean"** with product_id and category_id two will belong to **the "Soccer"** category.

First, we will create the schema for this dataframe. Which will look like this:-

```
categories_schema = StructType(fields=[StructField('category_id',IntegerType(), nullable=True),
    StructField('category_department_id', IntegerType(), nullable=True),
    StructField('category_name',StringType(), nullable=True)])
```

We will make the dataframe using the above schema.

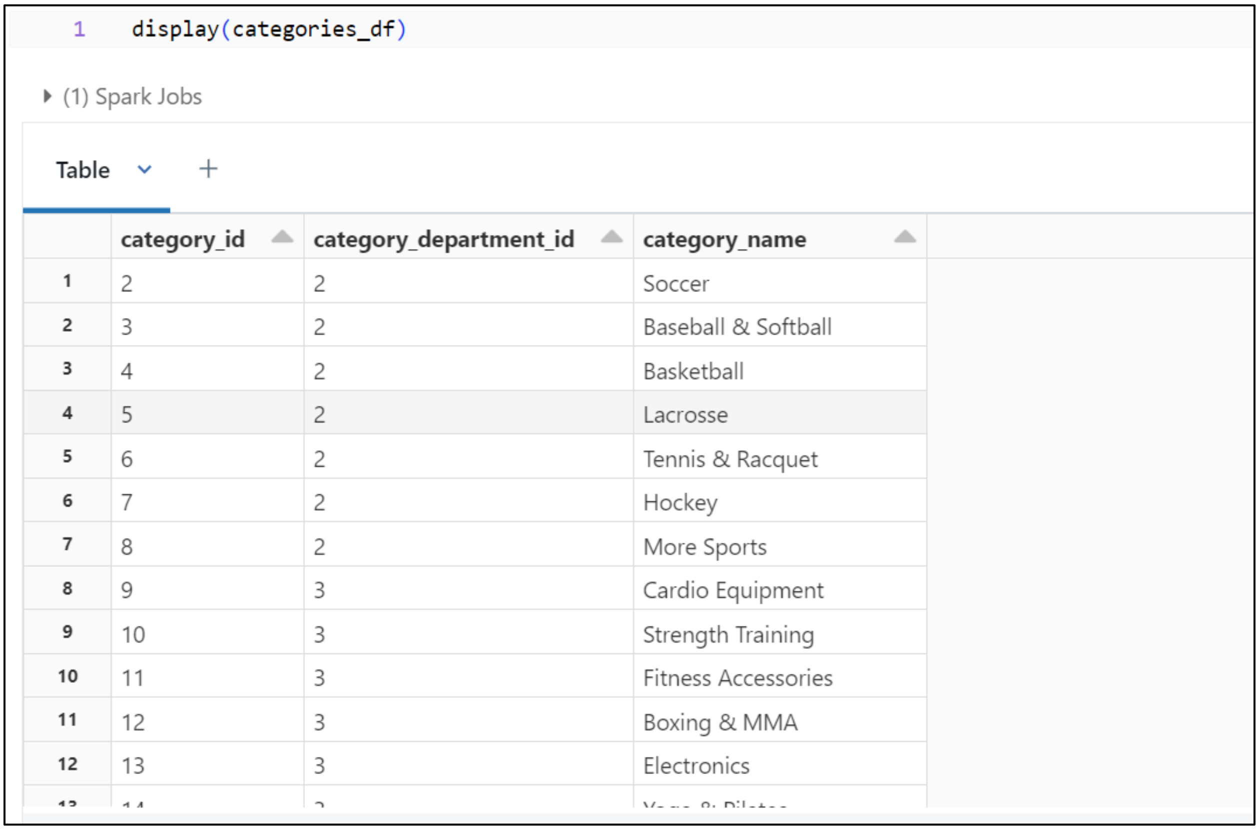
Reading the data through a CSV file.

```
categories_df = spark.read.csv\
```

("dbfs:/FileStore/tables/categories_1.csv",header=True,schema=categories_schema
)

To show the data frame the command is
display(categories_df)

The product dataframe will look like this:-



```
1 display(categories_df)
```

(1) Spark Jobs

Table

	category_id	category_department_id	category_name
1	2	2	Soccer
2	3	2	Baseball & Softball
3	4	2	Basketball
4	5	2	Lacrosse
5	6	2	Tennis & Racquet
6	7	2	Hockey
7	8	2	More Sports
8	9	3	Cardio Equipment
9	10	3	Strength Training
10	11	3	Fitness Accessories
11	12	3	Boxing & MMA
12	13	3	Electronics

4. Orders Dataframe: In this dataframe, we have details related to item orders and their payment status. It has four columns which are as follows:-

- **order_id:** It has the Ids of the ordered item.
- **order_date**: The date and time values are included in this column.
- **order_customer_id**: The customer order Ids is contained in this column.

- **order_status**: The payment status details are in this column.

Let's create a schema and '**orders_df**' dataframe using this dataset.

```
# define the schema for orders
orders_schema = StructType(fields=[
    StructField('order_id',IntegerType(), nullable=True),
    StructField('order_date',StringType(), nullable=True),
    StructField('order_customer_id',IntegerType(), nullable=True),
    StructField('order_status',StringType(), nullable=True)])
```

```
# Reading the data through a CSV file.

orders_df = spark.read.csv\
("dbfs:/FileStore/tables/orders_1.csv",header=True,schema=orders_schema)
```

The '**orders_df**' will look like this:-

Python

```
display(orders_df)
```

(1) Spark Jobs

Table

	order_id	order_date	order_customer_id	order_status
1	2	2013-07-25 00:00:00.0	256	PENDING_PAYMENT
2	3	2013-07-25 00:00:00.0	12111	COMPLETE
3	4	2013-07-25 00:00:00.0	8827	CLOSED
4	5	2013-07-25 00:00:00.0	11318	COMPLETE
5	6	2013-07-25 00:00:00.0	7130	COMPLETE
6	7	2013-07-25 00:00:00.0	4530	COMPLETE
7	8	2013-07-25 00:00:00.0	2911	PROCESSING
8	9	2013-07-25 00:00:00.0	5657	PENDING_PAYMENT
9	10	2013-07-25 00:00:00.0	5648	PENDING_PAYMENT
10	11	2013-07-25 00:00:00.0	918	PAYMENT_REVIEW
11	12	2013-07-25 00:00:00.0	1837	CLOSED

10,000 rows | Truncated data | 1.42 seconds runtime Refreshed now

5. Departments Dataframe: In this dataframe, we have department details. It has two columns which are as follows:-

- **department_id**: It has the ID's information.
- **department_name**: It has a list of department names. Ex:- Footwear, Apparel, Golf, Outdoors, etc.

Now we will create a schema and dataframe for the department dataset.

```
departments_schema = StructType(fields=[
   StructField('department_id',   IntegerType(), nullable=True),
   StructField('department_name', StringType(), nullable=True)])
```

Reading the data through a CSV file.

```
departments_df = spark.read.csv\
("dbfs:/FileStore/tables/departments_1.csv",header=True,schema=departments_sch
ema)
```

The sample data will look like this:-

Cmd 15

```
display(departments_df)
```

(1) Spark Jobs

Table

	department_id	department_name
1	3	Footwear
2	4	Apparel
3	5	Golf
4	6	Outdoors
5	7	Fan Shop

6. Order Items Dataframe: A collection of information about items ordered on an e-commerce platform or retail store is present in the order items dataframe. It comprises several columns providing details about each order item. The columns in the dataframe are as follows:-

- “**order_item_id**“: This column contains a unique identifier for each order item.
- “**order_item_order_id**“: This column contains the unique identifier of the order that the ordered item belongs.
- **“order_item_product_id”**: The unique identifier of the ordered products is stored in this column.
- **“order_item_quantity”:** The ordered product quantity is recorded in this column.
- **“order_item_subtotal**“: This column contains the total cost of the ordered item, calculated by multiplying the quantity by the product price.
- “**order_item_product_price**“: The cost of each product is recorded in this column.

Now we will create the schema and dataframe from the dataset.

define the schema for order items

```
order_items_schema = StructType(fields=[
    StructField('order_item_id',          IntegerType(), nullable=True),
    StructField('order_item_order_id',     IntegerType(), nullable=True),
    StructField('order_item_product_id',    IntegerType(), nullable=True),
```

```
    StructField('order_item_quantity',      IntegerType(), nullable=True),
    StructField('order_item_subtotal',      FloatType(), nullable=True),
    StructField('order_item_product_price', FloatType(), nullable=True)])
```

Reading the data through a CSV file.

```
order_items_df = spark.read.csv\
("dbfs:/FileStore/tables/order_items_1.csv",header=True,schema=order_items_sch
ema)
```

The dataframe will look like this:

```
display(order_items_df)
```

▸ (1) Spark Jobs

Table

	order_item_id	order_item_order_id	order_item_product_id	order_item_quantity	order_item_subtotal	order_item_product_price
1	2	2	1073	1	199.99	199.99
2	3	2	502	5	250	50
3	4	2	403	1	129.99	129.99
4	5	4	897	2	49.98	24.99
5	6	4	365	5	299.95	59.99
6	7	4	502	3	150	50
7	8	4	1014	4	199.92	49.98
8	9	5	957	1	299.98	299.98
9	10	5	365	5	299.95	59.99
10	11	5	1014	2	99.96	49.98
11	12	5	957	1	299.98	299.98
12	13	5	403	1	129.99	129.99
13	14	7	1073	1	199.99	199.99

Schemas :

```
orders_df.printSchema()
```

```
root
 |-- order_id: integer (nullable = true)
 |-- order_date: string (nullable = true)
 |-- order_customer_id: integer (nullable = true)
 |-- order_status: string (nullable = true)
```

Command took 0.09 seconds -- by mikancodes@gmail.com at 4/22/2024, 10:24:28 PM on My Cluster

```
1   order_items_df.printSchema()
```

```
root
 |-- order_item_id: integer (nullable = true)
 |-- order_item_order_id: integer (nullable = true)
 |-- order_item_product_id: integer (nullable = true)
 |-- order_item_quantity: integer (nullable = true)
 |-- order_item_subtotal: float (nullable = true)
 |-- order_item_product_price: float (nullable = true)
```

Command took 0.08 seconds -- by mikancodes@gmail.com at 4/22/2024, 10:28:32 PM on My Cluster

Python

```
1   products_df.printSchema()
```

```
root
 |-- product_id: integer (nullable = true)
 |-- product_category_id: integer (nullable = true)
 |-- product_name: string (nullable = true)
 |-- product_price: float (nullable = true)
 |-- product_image: string (nullable = true)
```

Command took 0.09 seconds -- by mikancodes@gmail.com at 4/22/2024, 10:29:48 PM on My Cluster

Python

```
1   categories_df.printSchema()
```

```
root
 |-- category_id: integer (nullable = true)
 |-- category_department_id: integer (nullable = true)
 |-- category_name: string (nullable = true)
```

Command took 0.12 seconds -- by mikancodes@gmail.com at 4/22/2024, 10:30:10 PM on My Cluster

Python

```
1   departments_df.printSchema()
```

```
root
 |-- department_id: integer (nullable = true)
 |-- department_name: string (nullable = true)
```

Command took 0.07 seconds -- by mikancodes@gmail.com at 4/22/2024, 10:31:01 PM on My Cluster

Python

```
1   customers_df.printSchema()
```

```
root
 |-- customer_id: integer (nullable = true)
 |-- customer_fname: string (nullable = true)
 |-- customer_lname: string (nullable = true)
 |-- customer_street: string (nullable = true)
 |-- customer_city: string (nullable = true)
 |-- customer_state: string (nullable = true)
 |-- customer_zipcode: string (nullable = true)
```

Command took 0.14 seconds -- by mikancodes@gmail.com at 4/22/2024, 10:31:30 PM on My Cluster

Creating Temporary Views:

- We can create temporary view for a Data Frame using createTempView or createOrReplaceTempView.
- createOrReplaceTempView will replace existing view, if it already exists.
- While tables in Metastore are permanent, views are temporary.
- Once the application exits, temporary views will be deleted or flushed out.

Create Temporary view

```
# customers_df.createOrReplaceTempView("customers")
# departments_df.createOrReplaceTempView("departments")
# categories_df.createOrReplaceTempView("categories")
# products_df.createOrReplaceTempView("products")
# orders_df.createOrReplaceTempView("orders")
# order_items_df.createOrReplaceTempView("order_items")
```

Checking Null Values:

Checking nulls

Python

```
# Create a list of column names
columns = customers_df.columns

# # Use select() and agg() to count null values in each column
customers_df.select([F.count(F.when(F.isnull(c), c)).alias(c) for c in columns]).show()

```

```
+-----------+--------------+--------------+--------------+-----------------+---------------+-------------+--------------+----------------+
|customer_id|customer_fname|customer_lname|customer_email|customer_password|customer_street|customer_city|customer_state|customer_zipcode|
+-----------+--------------+--------------+--------------+-----------------+---------------+-------------+--------------+----------------+
|          0|             0|             0|             0|                0|              0|            0|             0|               0|
+-----------+--------------+--------------+--------------+-----------------+---------------+-------------+--------------+----------------+
```

Command complete

Data Cleaning :

Data Cleaning

Cmd 17

```
customers_df.show(8)
```

```
+-----------+--------------+--------------+--------------+-----------------+--------------------+-------------+--------------+----------------+
|customer_id|customer_fname|customer_lname|customer_email|customer_password|     customer_street|customer_city|customer_state|customer_zipcode|
+-----------+--------------+--------------+--------------+-----------------+--------------------+-------------+--------------+----------------+
|          1|       Richard|     Hernandez|     XXXXXXXXX|        XXXXXXXXX|  6303 Heather Plaza|  Brownsville|            TX|           78521|
|          2|          Mary|       Barrett|     XXXXXXXXX|        XXXXXXXXX|9526 Noble Embers...|    Littleton|            CO|           80126|
|          3|           Ann|         Smith|     XXXXXXXXX|        XXXXXXXXX|3422 Blue Pioneer...|       Caguas|            PR|           00725|
|          4|          Mary|         Jones|     XXXXXXXXX|        XXXXXXXXX|  8324 Little Common|   San Marcos|            CA|           92069|
|          5|        Robert|        Hudson|     XXXXXXXXX|        XXXXXXXXX|10 Crystal River ...|       Caguas|            PR|           00725|
|          6|          Mary|         Smith|     XXXXXXXXX|        XXXXXXXXX|3151 Sleepy Quail...|      Passaic|            NJ|           07055|
|          7|       Melissa|        Wilcox|     XXXXXXXXX|        XXXXXXXXX|9453 High Concession|       Caguas|            PR|           00725|
|          8|         Megan|         Smith|     XXXXXXXXX|        XXXXXXXXX|3047 Foggy Forest...|     Lawrence|            MA|           01841|
+-----------+--------------+--------------+--------------+-----------------+--------------------+-------------+--------------+----------------+
only showing top 8 rows
```

Command complete

- In customer data frame, customer_email & customer_password are unnecessary for us to deal with data analysis and also data in corrupted form. So, we are removing those 2 columns.

Customer_dataframe

```
#dropping unwanted columns
customers_df = customers_df.drop("customer_email", "customer_password")

# Show the DataFrame after removing the columns
customers_df.show(10 , truncate = False )
```

```
+-----------+--------------+--------------+---------------------------+-------------+--------------+----------------+
|customer_id|customer_fname|customer_lname|customer_street            |customer_city|customer_state|customer_zipcode|
+-----------+--------------+--------------+---------------------------+-------------+--------------+----------------+
|1          |Richard       |Hernandez     |6303 Heather Plaza         |Brownsville  |TX            |78521           |
|2          |Mary          |Barrett       |9526 Noble Embers Ridge    |Littleton    |CO            |80126           |
|3          |Ann           |Smith         |3422 Blue Pioneer Bend     |Caguas       |PR            |00725           |
|4          |Mary          |Jones         |8324 Little Common         |San Marcos   |CA            |92069           |
|5          |Robert        |Hudson        |10 Crystal River Mall      |Caguas       |PR            |00725           |
|6          |Mary          |Smith         |3151 Sleepy Quail Promenade|Passaic      |NJ            |07055           |
|7          |Melissa       |Wilcox        |9453 High Concession       |Caguas       |PR            |00725           |
|8          |Megan         |Smith         |3047 Foggy Forest Plaza    |Lawrence     |MA            |01841           |
|9          |Mary          |Perez         |3616 Quaking Street        |Caguas       |PR            |00725           |
|10         |Melissa       |Smith         |8598 Harvest Beacon Plaza  |Stafford     |VA            |22554           |
+-----------+--------------+--------------+---------------------------+-------------+--------------+----------------+
only showing top 10 rows
```

Command complete

- In orders Data Frame order_date is in timestamp format we are converting it into date format(yyyy-mm-dd).

Orders Dataframe

```
orders_df.show(10,truncate=False)
```

```
+--------+---------------------+-----------------+---------------+
|order_id|order_date           |order_customer_id|order_status   |
+--------+---------------------+-----------------+---------------+
|1       |2013-07-25 00:00:00.0|11599            |CLOSED         |
|2       |2013-07-25 00:00:00.0|256              |PENDING_PAYMENT|
|3       |2013-07-25 00:00:00.0|12111            |COMPLETE       |
|4       |2013-07-25 00:00:00.0|8827             |CLOSED         |
|5       |2013-07-25 00:00:00.0|11318            |COMPLETE       |
|6       |2013-07-25 00:00:00.0|7130             |COMPLETE       |
|7       |2013-07-25 00:00:00.0|4530             |COMPLETE       |
|8       |2013-07-25 00:00:00.0|2911             |PROCESSING     |
|9       |2013-07-25 00:00:00.0|5657             |PENDING_PAYMENT|
|10      |2013-07-25 00:00:00.0|5648             |PENDING_PAYMENT|
+--------+---------------------+-----------------+---------------+
only showing top 10 rows

Command complete
```

```
orders_df = orders_df.withColumn("order_date", date_format(col("order_date"), "yyyy-MM-dd"))
orders_df.show(10)
```

```
+--------+----------+-----------------+---------------+
|order_id|order_date|order_customer_id|   order_status|
+--------+----------+-----------------+---------------+
|       1|2013-07-25|            11599|         CLOSED|
|       2|2013-07-25|              256|PENDING_PAYMENT|
|       3|2013-07-25|            12111|       COMPLETE|
|       4|2013-07-25|             8827|         CLOSED|
|       5|2013-07-25|            11318|       COMPLETE|
|       6|2013-07-25|             7130|       COMPLETE|
|       7|2013-07-25|             4530|       COMPLETE|
|       8|2013-07-25|             2911|     PROCESSING|
|       9|2013-07-25|             5657|PENDING_PAYMENT|
|      10|2013-07-25|             5648|PENDING_PAYMENT|
+--------+----------+-----------------+---------------+
only showing top 10 rows

Command complete
```

- Products Data Frame – Dropping the product description column

Products dataframe

```
products_df.show(5,truncate=False)

# Droping the product description column
products_df = products_df.drop("product_description")
products_df.show(3)
```

```
+----------+-------------------+---------------------------------------------+-------------------+-------------+-----------------------------------------------------------------------------------+
|product_id|product_category_id|product_name                                 |product_description|product_price|product_image                                                                      |
+----------+-------------------+---------------------------------------------+-------------------+-------------+-----------------------------------------------------------------------------------+
|1         |2                  |Quest Q64 10 FT. x 10 FT. Slant Leg Instant U|null               |59.98        |http://images.acmesports.sports/Quest+Q64+10+FT.+x+10+FT.+Slant+Leg+Instant+Up+Canopy|
|2         |2                  |Under Armour Men's Highlight MC Football Clea|null               |129.99       |http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat     |
|3         |2                  |Under Armour Men's Renegade D Mid Football Cl|null               |89.99        |http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat   |
|4         |2                  |Under Armour Men's Renegade D Mid Football Cl|null               |89.99        |http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat   |
|5         |2                  |Riddell Youth Revolution Speed Custom Footbal|null               |199.99       |http://images.acmesports.sports/Riddell+Youth+Revolution+Speed+Custom+Football+Helmet|
+----------+-------------------+---------------------------------------------+-------------------+-------------+-----------------------------------------------------------------------------------+
only showing top 5 rows

+----------+-------------------+--------------------+-------------+--------------------+
|product_id|product_category_id|        product_name|product_price|       product_image|
+----------+-------------------+--------------------+-------------+--------------------+
|         1|                  2|Quest Q64 10 FT. ...|        59.98|http://images.acm...|
|         2|                  2|Under Armour Men'...|       129.99|http://images.acm...|
|         3|                  2|Under Armour Men'...|        89.99|http://images.acm...|
+----------+-------------------+--------------------+-------------+--------------------+
only showing top 3 rows

Command complete
```

Santanu Adhikary 2320846

DATA ANALYSIS & VISUALISATION

1.Total Number of Orders placed :

```
orders_df.count()
```

```
Out[21]: 68883

Command complete
```

2.Total Revenue for each year :

```
total_revenue_per_year = (orders_df.filter(col('order_status')!="CANCELED").join(order_items_df, orders_df.order_id == order_items_df.order_item_order_id)
    .select([F.year('order_date').alias('order_year'), 'order_item_subtotal'])
    .groupBy(['order_year'])
    .agg(F.sum('order_item_subtotal').alias('tot_sales'))
    .orderBy(['order_year']))
display(total_revenue_per_year)
```

Table

	order_year	tot_sales
1	2013	14940106.250074387
2	2014	18686483.344932556

2 rows

Command complete

3. Total revenue for each month:

```
tot_rev_per_month_per_year = (orders_df.filter(col('order_status')!="CANCELED").join(order_items_df, orders_df.order_id == order_items_df.order_item_order_id)
    .select([F.year('order_date').alias('order_year'), F.month('order_date').alias('order_month'), 'order_item_subtotal'])
    .groupBy(['order_year', 'order_month'])
    .agg(F.sum('order_item_subtotal').alias('tot_revenue'))
    .orderBy(['order_year', 'order_month']))

display(tot_rev_per_month_per_year)
```

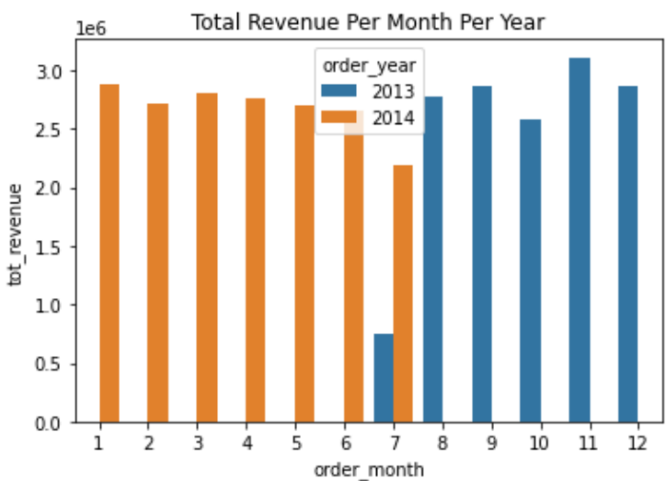
Table

	order_year	order_month	tot_revenue
1	2013	7	754899.794549942
2	2013	8	2769236.083465576
3	2013	9	2866553.3851947784
4	2013	10	2573575.719619751
5	2013	11	3105843.3310375214
6	2013	12	2869997.9362068176
7	2014	1	2870834.236070633

13 rows

Plot For Total Revenue Per Month-

```python
pdf = tot_rev_per_month_per_year.toPandas()

g = sns.barplot(x='order_month', y='tot_revenue', hue='order_year', data=pdf)
g.set_title('Total Revenue Per Month Per Year');
```




```
Command complete
```

4.Total purchase by each customer :

```python
customer_purchase_history_df = customers_df.join(orders_df, customers_df.customer_id == orders_df.order_customer_id, "inner") \
    .join(order_items_df, orders_df.order_id == order_items_df.order_item_order_id, "inner") \
    .select("customer_id", "order_date", "order_item_product_id", "order_item_subtotal")
customer_purchase_history_df.orderBy(col('customer_id')).show(10)
```

```
+-----------+----------+---------------------+-------------------+
|customer_id|order_date|order_item_product_id|order_item_subtotal|
+-----------+----------+---------------------+-------------------+
|          1|2013-12-13|                  191|             499.95|
|          2|2013-10-29|                 1014|              99.96|
|          2|2014-02-18|                  502|               50.0|
|          2|2014-02-18|                 1073|             199.99|
|          2|2014-02-18|                  957|             299.98|
|          2|2013-08-02|                 1014|              99.96|
|          2|2013-08-02|                 1014|             149.94|
|          2|2013-08-02|                  627|             199.95|
|          2|2013-08-02|                 1073|             199.99|
|          2|2013-08-02|                  365|             119.98|
+-----------+----------+---------------------+-------------------+
only showing top 10 rows

Command complete
```

```
customer_purchase_history_df = customer_purchase_history_df.withColumn("total_purchase_amount", col("order_item_subtotal")) \
    .groupBy("customer_id").agg({"total_purchase_amount": "sum"}) \
    .withColumnRenamed("sum(total_purchase_amount)", "total_purchase_amount")
customer_purchase_history_df = customer_purchase_history_df.withColumn("total_purchase_amount", F.round("total_purchase_amount", 2))
customer_purchase_history_df = customer_purchase_history_df.orderBy(F.desc("total_purchase_amount"))
customer_purchase_history_df.show(10)
```

```
+-----------+--------------------+
|customer_id|total_purchase_amount|
+-----------+--------------------+
|        791|            10524.17|
|       9371|             9299.03|
|       8766|             9296.14|
|       1657|             9223.71|
|       2641|             9130.92|
|       1288|             9019.11|
|       3710|              9019.1|
|       4249|             8918.85|
|       5654|             8904.95|
|       5624|             8761.98|
+-----------+--------------------+
only showing top 10 rows

Command complete
```

5. Customer type according to their purchase :

```
    high_value_customers = customer_purchase_history_df.filter(col("total_purchase_amount") > 4000).orderBy(col("total_purchase_amount").desc())
medium_value_customers = customer_purchase_history_df.filter((col("total_purchase_amount") <= 4000) & (col("total_purchase_amount") > 2000)).orderBy(col('total_purchase_amount').desc())
low_value_customers = customer_purchase_history_df.filter(col("total_purchase_amount") <= 2000).orderBy(col('total_purchase_amount').desc())

# Print the segments
# print("High-Value Customers:")
high_value_customers.show(10)
# print("Medium-Value Customers:")
medium_value_customers.show(10)
# print("Low-Value Customers:")
low_value_customers.show(10)
```

Types of customers

Python

```python
high_value_customers = customer_purchase_history_df.filter(col("total_purchase_amount") > 4000).orderBy(col
("total_purchase_amount").desc())

# Print the segments
print("High-Value Customers:")
high_value_customers.show(10)

```

(4) Spark Jobs

high_value_customers: pyspark.sql.dataframe.DataFrame = [customer_id: integer, total_purchase_amount: double]

```
High-Value Customers:
+-----------+--------------------+
|customer_id|total_purchase_amount|
+-----------+--------------------+
|         58|    6615.160125732422|
|         12|    6009.350101470947|
|          7|     5569.48010635376|
|         40|      5253.6201171875|
|         55|    4763.400077819824|
|         53|    4689.480113983154|
|         51|    4513.430065155029|
|         19|    4355.620079040527|
|         17|    4259.660110473633|
|         23|    4141.380058288574|
+-----------+--------------------+
```

```python
low_value_customers = customer_purchase_history_df.filter(col("total_purchase_amount") <= 2000).orderBy(col
("total_purchase_amount").desc())
print("Low-Value Customers:")
low_value_customers.show(10)
```

(4) Spark Jobs

low_value_customers: pyspark.sql.dataframe.DataFrame = [customer_id: integer, total_purchase_amount: double]

```
Low-Value Customers:
+-----------+--------------------+
|customer_id|total_purchase_amount|
+-----------+--------------------+
|          2|   1819.7300338745117|
|          4|   1719.6300296783447|
|         20|    1589.770050048828l|
|         36|   1389.8800506591797|
|         49|     1359.780014038086|
|         29|   1329.8700408935547|
|         37|   1329.7600212097168|
|          5|   1274.7500228881836|
|         10|   1264.7900123596191|
|         33|   1229.8700370788574|
+-----------+--------------------+
only showing top 10 rows
```

Command took 2.34 seconds -- by mikancodes@gmail.com at 4/23/2024, 3:29:19 PM on My Cluster

Cmd 20

```
medium_value_customers = customer_purchase_history_df.filter((col("total_purchase_amount") <= 4000) & (col
("total_purchase_amount") > 2000)).orderBy(col("total_purchase_amount").desc())
print("Medium-Value Customers:")
medium_value_customers.show(10)
```

(4) Spark Jobs

medium_value_customers: pyspark.sql.dataframe.DataFrame = [customer_id: integer, total_purchase_amount: double]

```
Medium-Value Customers:
+-----------+--------------------+
|customer_id|total_purchase_amount|
+-----------+--------------------+
|         46|   3887.620090484619|
|         31|  3774.5500564575195|
|          8|    3763.50004196167|
|          3|   3537.680093765259|
|         18|   3519.600067138672|
|         48|  3509.6200561523438|
|         32|  3409.5100421905518|
|         45|   3329.740074157715|
|         42|   3289.660041809082|
|          6|   3259.510025024414|
+-----------+--------------------+
only showing top 10 rows
```

6.Monthly sales product trends :

```
product_sales_df = products_df.join(order_items_df, products_df.product_id == order_items_df.order_item_product_id, "inner") \
    .join(orders_df, order_items_df.order_item_order_id == orders_df.order_id, "inner") \
    .select("product_id", "order_date", "order_item_quantity", "product_name")

# Extract year and month from order date
product_sales_df = product_sales_df.withColumn("order_year", F.year("order_date")) \
    .withColumn("order_month", F.month("order_date"))

# Calculate total sales per product per month
monthly_sales_df = product_sales_df.groupBy("product_id", "order_year", "order_month") \
    .agg(F.sum("order_item_quantity").alias("total_quantity_sold"))

# Identify seasonal patterns (e.g., high sales during holidays)
# You can customize this based on your business context
seasonal_patterns_df = monthly_sales_df.groupBy("product_id", "order_month") \
    .agg(F.sum("total_quantity_sold").alias("total_monthly_sales"))

# Print the seasonal patterns
seasonal_patterns_df.show(10)
```

```
# Print the seasonal patterns
seasonal_patterns_df.show(10)
```

▸ (5) Spark Jobs

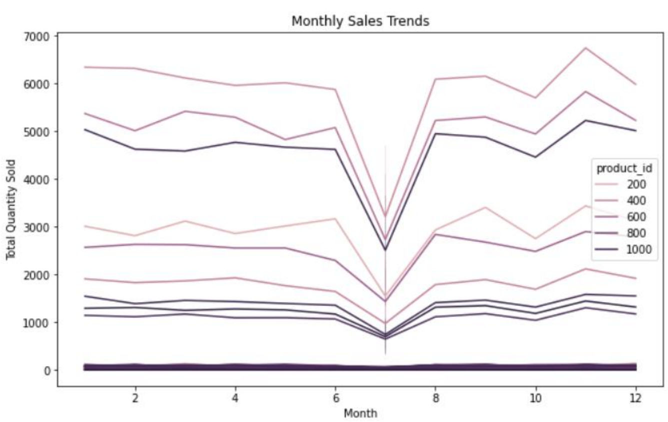
▸ product_sales_df: pyspark.sql.dataframe.DataFrame = [product_id: integer, order_date: string ... 4 more fields]
▸ monthly_sales_df: pyspark.sql.dataframe.DataFrame = [product_id: integer, order_year: integer ... 2 more fields]
▸ seasonal_patterns_df: pyspark.sql.dataframe.DataFrame = [product_id: integer, order_month: integer ... 1 more field]

```
+----------+-----------+-------------------+
|product_id|order_month|total_monthly_sales|
+----------+-----------+-------------------+
|       276|         11|                 64|
|       924|          8|                 65|
|       278|          8|                 72|
|       564|         11|                 70|
|       858|          4|                  7|
|       203|          4|                  2|
|       825|          6|                 76|
|       135|          9|                 73|
|       642|         12|                 69|
|       906|          8|                 54|
+----------+-----------+-------------------+
only showing top 10 rows

Command took 4.39 seconds -- by ambatinvrohit@gmail.com at 4/23/2024, 3:24:40 PM on My Cluster1
```

Plot-

```
pandas_df = monthly_sales_df.toPandas()

# Line chart for monthly sales trends
plt.figure(figsize=(10, 6))
sns.lineplot(x="order_month", y="total_quantity_sold", hue="product_id", data=pandas_df)
plt.title("Monthly Sales Trends")
plt.xlabel("Month")
plt.ylabel("Total Quantity Sold")
plt.show()
```

▸ (4) Spark Jobs



7.Top performing departments :

```
df = (orders_df
      .filter((orders_df.order_status != 'CANCELED') & (orders_df.order_status != 'SUSPECTED_FRAUD'))
      .join(order_items_df, orders_df.order_id == order_items_df.order_item_order_id, how='inner')
      .join(products_df, order_items_df.order_item_product_id == products_df.product_id, how='inner')
      .join(categories_df, products_df.product_category_id == categories_df.category_id, how='inner')
      .join(departments_df, categories_df.category_department_id == departments_df.department_id, how='inner')
      .select('department_name', F.year(orders_df.order_date).alias('order_year'), 'order_item_subtotal')
      .groupBy([departments_df.department_name, 'order_year'])
      .agg(F.sum(order_items_df.order_item_subtotal).alias('tot_revenue'))
      .orderBy('department_name', 'order_year'))
df.cache()

df.show(5)
```

(4) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [department_name: string, order_year: integer ... 1 more field]

```
+---------------+----------+-----------------+
|department_name|order_year|      tot_revenue|
+---------------+----------+-----------------+
|        Apparel|      2013|3090985.6535224915|
|        Apparel|      2014| 3917585.841217041|
|       Fan Shop|      2013| 7290531.899988174|
|       Fan Shop|      2014|  9095735.77280426|
|       Footwear|      2013|1711492.5186824799|
+---------------+----------+-----------------+
only showing top 5 rows
```

8. Highest priced product :

```
(products_df
 .select('*')
 .filter(col('product_price') == products_df.select(F.max('product_price')).collect()[0][0])
 .show())
```

(3) Spark Jobs

```
+----------+-------------------+-------------------+-------------------+-------------+--------------------+
|product_id|product_category_id|       product_name|product_description|product_price|       product_image|
+----------+-------------------+-------------------+-------------------+-------------+--------------------+
|       208|                 10|SOLE E35 Elliptical|               null|      1999.99|http://images.acm...|
+----------+-------------------+-------------------+-------------------+-------------+--------------------+
```

Command took 1.48 seconds -- by ambatinvrohit@gmail.com at 4/23/2024, 2:46:58 PM on My Cluster1

9.Popular Category :

Python

```
popular_category_df = order_items_df.join(products_df, col("order_item_product_id") ==  col("product_id"), how='inner').join
(categories_df, col("category_id") == col("product_category_id"), how='inner').groupBy('category_name').agg(F.sum
('order_item_quantity').alias('order_count')).orderBy('order_count', ascending=False).limit(10)
popular_category_df.show()
```

▸ (4) Spark Jobs

▸ popular_category_df: pyspark.sql.dataframe.DataFrame = [category_name: string, order_count: long]

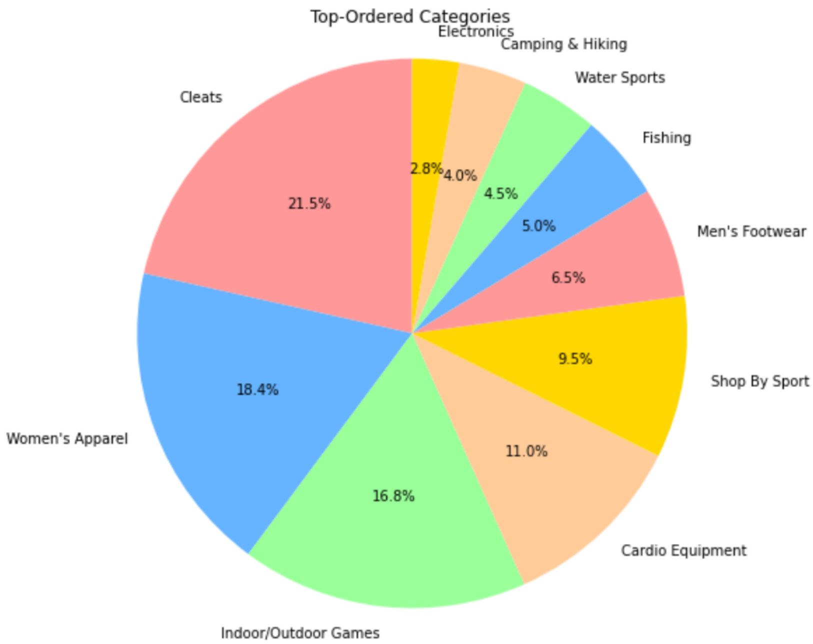
```
+--------------------+-----------+
|       category_name|order_count|
+--------------------+-----------+
|              Cleats|      73734|
|      Women's Apparel|      62956|
|Indoor/Outdoor Games|      57803|
|    Cardio Equipment|      37587|
|        Shop By Sport|      32726|
|       Men's Footwear|      22246|
|             Fishing|      17325|
|        Water Sports|      15540|
|     Camping & Hiking|      13729|
|         Electronics|       9436|
+--------------------+-----------+
```

Command took 3.70 seconds -- by mikancodes@gmail.com at 4/23/2024, 3:34:23 PM on My Cluster

```
top_categories_df = popular_category_df.orderBy(col("order_count").desc())

custom_colors = ["#FF9999", "#66B3FF", "#99FF99", "#FFCC99", "#FFD700"]


# Extract category names and order counts
category_names = top_categories_df.select("category_name").rdd.flatMap(lambda x: x).collect()
order_counts = top_categories_df.select("order_count").rdd.flatMap(lambda x: x).collect()

# Create the pie chart
plt.figure(figsize=(8, 6))
plt.pie(order_counts, labels=category_names, autopct="%1.1f%%", startangle=90 , colors=custom_colors)
plt.title("Top-Ordered Categories")
plt.axis("equal")  # Equal aspect ratio ensures that pie is drawn as a circle
plt.show()

```

▸ (8) Spark Jobs

▸ top_categories_df: pyspark.sql.dataframe.DataFrame = [category_name: string, order_count: long]



Command took 6.61 seconds -- by mikancodes@gmail.com at 4/23/2024, 3:44:47 PM on My Cluster

10. Count of Orders based on their Status :

```
orders_by_status = orders_df.groupBy(col("order_status")).agg(count(col("order_status")).alias("total_status"))
orders_by_status.show()
```

▸ (2) Spark Jobs

▸ orders_by_status: pyspark.sql.dataframe.DataFrame = [order_status: string, total_status: long]

```
+---------------+------------+
|   order_status|total_status|
+---------------+------------+
|PENDING_PAYMENT|       15030|
|       COMPLETE|       22899|
|        ON_HOLD|        3798|
| PAYMENT_REVIEW|         729|
|     PROCESSING|        8275|
|         CLOSED|        7556|
|SUSPECTED_FRAUD|        1558|
|        PENDING|        7610|
|       CANCELED|        1428|
+---------------+------------+
```

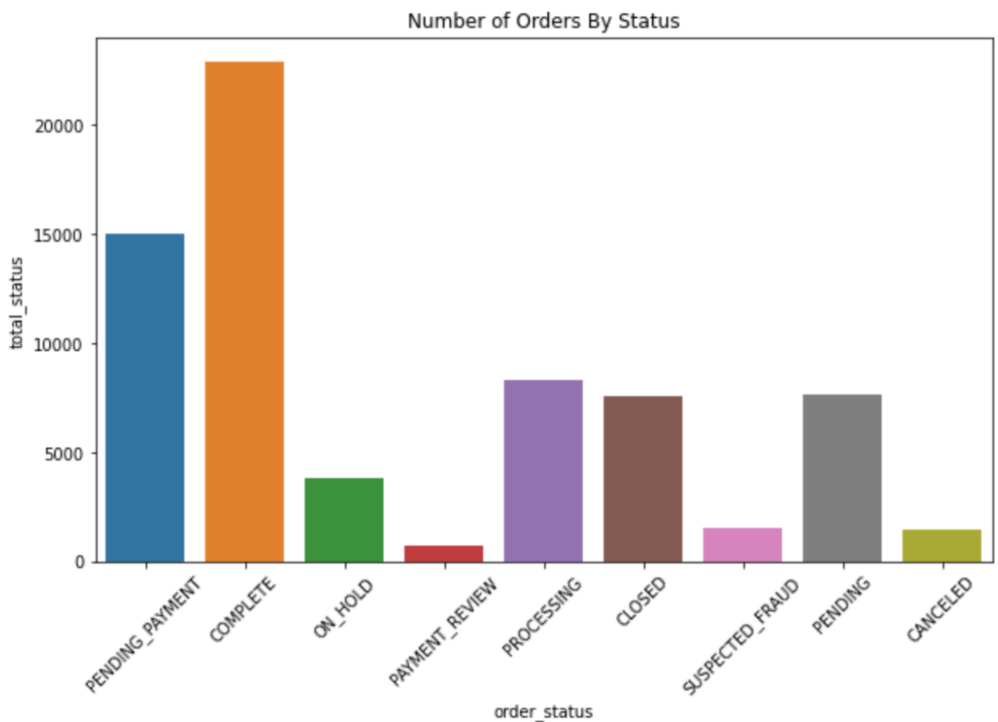
Command took 1.28 seconds -- by mikancodes@gmail.com at 4/23/2024, 2:33:35 PM on My Cluster

Plot for number of orders by status

Python

```python
# convert it in pandas dataframe
order_stat = orders_by_status.toPandas()
plt.figure(figsize=(10, 6))

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)
# plot the data using barplot
g = sns.barplot(x='order_status', y='total_status', data=order_stat)
g.set_title('Number of Orders By Status');
```

(2) Spark Jobs



Conclusion

We solve a case study utilizing retail data to uncover hidden insights that can increase sales using PySpark. This analysis allows the store manager to understand which products require more attention. The insights gained from the data can assist in tracking individual product performance, identifying top-selling items, and adjusting inventory levels accordingly. Retailers can also use sales data to analyze customer purchasing patterns and make informed decisions about product placement, promotions, and pricing strategies.

1. Working with real-world data to gather valuable information can benefit businesses in various ways.
2. We have also seen the detailed step-by-step solutions to the problem using PySpark functions and analysis of the output at the end of each problem statement.