ETL PIPELINE

DWHM PROJECT

–

Course Code: CT-463

Submitted to: Dr. Umer Farooq

Prepared by:

Hamza Ahmed Khan CT-035

Arqam Khursheed CT-018

Saim Ahmed CT-019

Muhammad Abdul Rafay CT-050

Business Process:

The business process depicted in the dataset revolves around the end-to-end cycle of sales operations. This process encompasses various stages, starting from the customer placing an order to the shipment of products, and includes aspects such as order processing, inventory management, and financial transactions. The dataset captures crucial details like order dates, shipping information, product categories, sales quantities, discounts, and profits, providing a comprehensive view of the sales lifecycle.

Analyzing this dataset through the lens of E-commerce Sales Management can yield valuable insights for businesses aiming to enhance customer satisfaction, streamline operations, and improve overall financial performance.

Overview of Datasets:

- ## Dataset 1 (Product Sales):

  Overview of the Dataset:
  Explanation of the dataset's content and purpose, highlighting its focus on sales transactions involving various products.

  Data Fields:
  Breakdown of the key information included in each row, such as product ID, category, sub-category, sales amount, quantity, discount, and profit.

  Product Classification:
  Description of the classification system used for products, covering categories and sub-categories. Examples may include office supplies, furniture items, and other relevant classifications.

  Sales Channels:
  Insight into the different sales channels represented in the dataset, including third-party channels, direct sales, and pre-booked orders.

  Product Range:

Overview of the types of products present in the dataset, ranging from office supplies like paper, binders, and labels to furniture items such as chairs, bookcases, and tables.

Transaction Details:
Explanation of how each row represents a specific transaction, with details on the quantities sold, discounts applied, and profits or losses incurred.

Analytical Potential:
Emphasis on the dataset's value for analyzing sales performance, identifying trends, and making informed business decisions related to product offerings and pricing strategies.

Business Insights:
Highlighting the potential applications of the dataset for gaining insights into overall business performance and making strategic decisions.

- ## Dataset 2 (Customer):

Overview of the Customer Orders Dataset:

Introduction to the dataset and its focus on customer orders, highlighting key details such as Row ID, Customer ID, Customer Name, Segment, Country/Region, City, State/Province, Postal Code, and Region.

Transaction Representation:

Explanation of how each row in the dataset represents a specific transaction or interaction with a customer, providing a snapshot of customer-related activities.

Customer Segmentation:

Description of the various customer segments present in the dataset, such as Replacement, Other, and Servicing, indicating different types of interactions or transactions.

Geographical Coverage:

Overview of the dataset's geographical coverage, specifying that transactions involve customers from different regions within the United States, including Central, East, South, and West.

City and State Diversity:

Emphasis on the diverse cities and states included in the dataset, illustrating a broad geographical representation of customer interactions.

Postal Code Information:

Mention of the inclusion of postal codes in the dataset, providing granular details about the location of customer transactions.

Customer Repetitions:

Explanation of the presence of repetitions in customer entries, suggesting multiple interactions or orders from the same customers and highlighting the potential for analyzing customer behavior over time.

Analytical Value:

Recognition of the dataset's value as a resource for analyzing customer behavior, preferences, and geographical trends, offering insights into customer interactions within specified regions.

- ## Dataset 3 (Order):

Overview of the Sales Orders Dataset:

Introduction to the dataset and its focus on sales orders, highlighting key information such as order details, dates, shipping methods, quantities, discounts, and profits.

Transaction Representation:

Explanation of how each row in the dataset represents a distinct sales transaction, with specific columns like "Order ID," "Order Date," "Ship Date," and "Ship Mode" providing details about the order and its delivery.

Quantities Ordered:

Description of the "Quantity" column, indicating the number of items ordered in each sales transaction, providing insights into the scale of individual orders.

Discounts Applied:

Explanation of the "Discount" column, highlighting its role in reflecting any price reductions applied to the purchase, influencing the overall transaction value.

Profit Analysis:

Overview of the "Profit" column, emphasizing its significance in quantifying the financial outcome of each sales transaction, indicating whether a profit or loss was incurred.

Product Category Diversity:

Recognition of the dataset's coverage of a range of product categories, illustrating the diversity of products involved in the sales transactions.

Regional Coverage:

Mention of the dataset encompassing various regions, capturing diverse scenarios in which sales occurred and providing a geographical context to the sales data.

Analytical Potential:

Emphasis on the potential for analyzing the dataset to derive valuable insights into sales performance, patterns, and factors influencing profitability, aiding in strategic decision-making.

Performing ETL:

STEP 1

- ### Data Extraction:

Getting started with our ETL (Extract, Transform, Load) journey, the first step is data extraction. With Python, we're gathering valuable pieces of data from three different sources.

Importing Dataset 1:

The first dataset, `Customer_Book.csv`, is loaded into a DataFrame named `customer_df`. The file is read using the `pd.read_csv` function with the specified file path and encoding. The resulting DataFrame, `customer_df`, contains the customer-related information from the imported dataset.

```
[ ] # Importing dataset 1

    import pandas as pd
    customer_df= pd.read_csv('/content/Customer_Book.csv', encoding="latin-1")

    customer_df
```

	Row ID	Customer ID	Customer Name	Segment	Country/Region	City	State/Province	Postal Code	Region
0	1	DP-13000	Darren Powers	Replacement	United States	Houston	Texas	77095	Central
1	2	PO-19195	Phillina Ober	Other	United States	Naperville	Illinois	60540	Central
2	3	PO-19195	Phillina Ober	Other	United States	Naperville	Illinois	60540	Central
3	4	PO-19195	Phillina Ober	Other	United States	Naperville	Illinois	60540	Central
4	5	MB-18085	Mick Brown	Replacement	United States	Philadelphia	Pennsylvania	19143	East
...	...	...	...	...	...	...	...	...	...
292	293	AH-10690	Anna Häberlin	Servicing	United States	Virginia Beach	Virginia	23464	South
293	294	RD-19585	Rob Dowd	Replacement	United States	Athens	Georgia	30605	South
294	295	SC-20020	Sam Craven	Replacement	United States	Houston	Texas	77095	Central
295	296	SC-20020	Sam Craven	Replacement	United States	Houston	Texas	77095	Central
296	297	RD-19585	Rob Dowd	Replacement	United States	Athens	Georgia	30605	South

297 rows × 9 columns

Importing Dataset 2:

Similarly, the second dataset is loaded from the file named `Product_Sales_Book.csv` and stored in a DataFrame called `product_df`. This dataset likely contains information related to product sales, and the Pandas `read_csv` function is employed to read the CSV file with the specified encoding.

```
# Importing dataset 2

import pandas as pd

product_df = pd.read_csv('/content/Product_Sales_Book.csv', encoding='latin-1')

product_df
```

	Row ID	Product ID	Category	Sub-Category	Sales	Quantity	Discount	Profit
0	1	OFF-PA-10000174	Third Party	Paper	16.448	2	0.2	5.5512
1	2	OFF-BI-10004094	Third Party	Binders	3.540	2	0.8	-5.4870
2	3	OFF-LA-10003223	Third Party	Labels	11.784	3	0.2	4.2717
3	4	OFF-ST-10002743	Third Party	Storage	272.736	3	0.2	-64.7748
4	5	OFF-AR-10003478	Third Party	Art	19.536	3	0.2	4.8840
...	...	...	...	...	...	...	...	...
292	293	FUR-FU-10003192	Direct Sale	Furnishings	177.680	2	0.0	46.1968
293	294	OFF-AP-10003842	Third Party	Appliances	154.900	5	0.0	40.2740
294	295	OFF-PA-10001593	Third Party	Paper	33.488	7	0.2	10.4650
295	296	OFF-PA-10002986	Third Party	Paper	26.720	5	0.2	9.3520
296	297	OFF-PA-10004248	Third Party	Paper	15.840	3	0.0	7.1280

297 rows × 8 columns

Importing Dataset 3:

The third dataset, represented by the file `Order_Book.csv`, is imported into a DataFrame named `order_df`. This dataset likely includes details about sales orders. The `pd.read_csv` function is once again utilized with the appropriate file path and encoding to read the CSV file and store its contents in the `order_df` DataFrame.

```
# Importing dataset 3

import pandas as pd
order_df= pd.read_csv('/content/Order_Book.csv', encoding='latin-1')
order_df
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode
0	1	US-2019-103800	03/01/2019	07/01/2019	Standard Class
1	2	US-2019-112326	04/01/2019	08/01/2019	Standard Class
2	3	US-2019-112326	04/01/2019	08/01/2019	Standard Class
3	4	US-2019-112326	04/01/2019	08/01/2019	Standard Class
4	5	US-2019-141817	05/01/2019	12/01/2019	Standard Class
...	...	...	...	...	...
292	293	US-2019-160276	02/04/2019	08/04/2019	Standard Class
293	294	US-2019-164315	02/04/2019	08/04/2019	Standard Class
294	295	US-2019-157847	02/04/2019	06/04/2019	Second Class
295	296	US-2019-157847	02/04/2019	06/04/2019	Second Class
296	297	US-2019-164315	02/04/2019	08/04/2019	Standard Class

297 rows × 5 columns

In summary, these code snippets illustrate the initial steps in the ETL process, focusing on the extraction phase by importing three distinct datasets related to customers, product sales, and sales orders into Pandas DataFrames for further analysis and manipulation in a Python environment.

STEP 2

- ## Data Transformation:

1. Data deduplication:

Checking if theres any duplicates in the datasets.

```
# Check duplicate values in Customer Analysis
duplicate_customers = customer_df[customer_df.duplicated()]
print("Duplicate Customers:")
print(duplicate_customers)

# Check duplicate values in Product Analysis
duplicate_products = product_df[product_df.duplicated()]
print("\nDuplicate Products:")
print(duplicate_products)

# Check duplicate values in Order Analysis
duplicate_orders = order_df[order_df.duplicated()]
print("\nDuplicate Orders:")
print(duplicate_orders)
```

```
Duplicate Customers:
Empty DataFrame
Columns: [Row ID, Customer ID, Customer Name, Segment, Country/Region, City, State/Province, Postal Code, Region]
Index: []

Duplicate Products:
Empty DataFrame
Columns: [Row ID, Product ID, Category, Sub-Category, Sales, Quantity, Discount, Profit]
Index: []

Duplicate Orders:
Empty DataFrame
Columns: [Row ID, Order ID, Order Date, Ship Date, Ship Mode]
Index: []
```

This output indicates that there are no duplicate values in each of the datasets. The empty DataFrames for Duplicate Customers, Duplicate Products, and Duplicate Orders suggest that there are no rows with identical values in the specified columns for each respective dataset.

2. INNER JOINING / MERGING:

Performing Inner Join/Merging on all three datasets

```
# Perform inner joins on common columns
merged_df = pd.merge(order_df, customer_df, how='inner')
merged_df = pd.merge(merged_df, product_df, how='inner')

# Display the merged DataFrame
merged_df.head()
```

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country/Region	City	State/Province	Postal Code	Region	Product ID	Category	Sub-Category	Sales	Quantity	Discount	Profit
1	US-2019-103800	03/01/2019	07/01/2019	Standard Class	DP-13000	Darren Powers	Replacement	United States	Houston	Texas	77095	Central	OFF-PA-10000174	Third Party	Paper	16.448	2	0.2	5.5512
2	US-2019-112326	04/01/2019	08/01/2019	Standard Class	PO-19195	Phillina Ober	Other	United States	Naperville	Illinois	60540	Central	OFF-BI-10004094	Third Party	Binders	3.540	2	0.8	-5.4870
3	US-2019-112326	04/01/2019	08/01/2019	Standard Class	PO-19195	Phillina Ober	Other	United States	Naperville	Illinois	60540	Central	OFF-LA-10003223	Third Party	Labels	11.784	3	0.2	4.2717
4	US-2019-112326	04/01/2019	08/01/2019	Standard Class	PO-19195	Phillina Ober	Other	United States	Naperville	Illinois	60540	Central	OFF-ST-10002743	Third Party	Storage	272.736	3	0.2	-64.7748
5	US-2019-141817	05/01/2019	12/01/2019	Standard Class	MB-18085	Mick Brown	Replacement	United States	Philadelphia	Pennsylvania	19143	East	OFF-AR-10003478	Third Party	Art	19.536	3	0.2	4.8840

Datasets are merged together. The merge function automatically identifying common columns between the first two DataFrames (customer_df and product_df) and then merge the result with the third DataFrame (order_df).

3. DATA ENCRICHMENT:

Data Enrichment refers to adding additional information or deriving new features from existing data.

Assuming we want to calculate the ratio of 'Total Sales' from 'Sales' and 'Profit Margin' from 'Profit'.

```python
# Add a Total Sales column
merged_df['Total Sales'] = merged_df['Sales'] * merged_df['Quantity']

# Add a Profit Margin column
merged_df['Profit Margin'] = (merged_df['Profit'] / merged_df['Total Sales']) * 100

# Display the updated DataFrame with the new columns
merged_df
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country/Region	City	...	Region	Product ID	Category	Sub-Category	Sales	Quantity	Discount	Profit	Total Sales	Profit Margin
	1	US-2019-103800	03/01/2019	07/01/2019	Standard Class	DP-13000	Darren Powers	Replacement	United States	Houston	...	Central	OFF-PA-10000174	Third Party	Paper	16.448	2	0.2	5.5512	32.896	16.875000
	2	US-2019-112326	04/01/2019	08/01/2019	Standard Class	PO-19195	Phillina Ober	Other	United States	Naperville	...	Central	OFF-BI-10004094	Third Party	Binders	3.540	2	0.8	-5.4870	7.080	-77.500000
	3	US-2019-112326	04/01/2019	08/01/2019	Standard Class	PO-19195	Phillina Ober	Other	United States	Naperville	...	Central	OFF-LA-10003223	Third Party	Labels	11.784	3	0.2	4.2717	35.352	12.083333
	4	US-2019-112326	04/01/2019	08/01/2019	Standard Class	PO-19195	Phillina Ober	Other	United States	Naperville	...	Central	OFF-ST-10002743	Third Party	Storage	272.736	3	0.2	-64.7748	818.208	-7.916667
	5	US-2019-141817	05/01/2019	12/01/2019	Standard Class	MB-18085	Mick Brown	Replacement	United States	Philadelphia	...	East	OFF-AR-10003478	Third Party	Art	19.536	3	0.2	4.8840	58.608	8.333333
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	293	US-2019-160276	02/04/2019	08/04/2019	Standard Class	AH-10690	Anna Häberlin	Servicing	United States	Virginia Beach	...	South	FUR-FU-10003192	Direct Sale	Furnishings	177.680	2	0.0	46.1968	355.360	13.000000
	294	US-2019-164315	02/04/2019	08/04/2019	Standard Class	RD-19585	Rob Dowd	Replacement	United States	Athens	...	South	OFF-AP-10003842	Third Party	Appliances	154.900	5	0.0	40.2740	774.500	5.200000
	295	US-2019-157847	02/04/2019	06/04/2019	Second Class	SC-20020	Sam Craven	Replacement	United States	Houston	...	Central	OFF-PA-10001593	Third Party	Paper	33.488	7	0.2	10.4650	234.416	4.464286
	296	US-2019-157847	02/04/2019	06/04/2019	Second Class	SC-20020	Sam Craven	Replacement	United States	Houston	...	Central	OFF-PA-10002986	Third Party	Paper	26.720	5	0.2	9.3520	133.600	7.000000
	297	US-2019-164315	02/04/2019	08/04/2019	Standard Class	RD-19585	Rob Dowd	Replacement	United States	Athens	...	South	OFF-PA-10004248	Third Party	Paper	15.840	3	0.0	7.1280	47.520	15.000000

In summary, the code appends 'Total Sales' (calculated as the product of 'Sales' and 'Quantity') and 'Profit Margin' (expressed as a percentage of profit to total sales) columns to the DataFrame `merged_df`. These additions provide a quick overview of total sales and profit margins for each entry in the dataset.

4. DATA SUMMARIZATION:

i. Descriptive Summarization:

```
# Calculate summary statistics
summary_stats = merged_df.describe()

# Transpose the summary statistics DataFrame to make it suitable for appending
summary_stats = summary_stats.transpose()

# Add the summary statistics row to the original DataFrame
merged_df = merged_df.append(summary_stats)

# Display the updated DataFrame with summary statistics
merged_df
```

...	Total Sales	Profit Margin	count	mean	std	min	25%	50%	75%	max
...	32.896	16.875000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	7.080	-77.500000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	35.352	12.083333	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	818.208	-7.916667	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	58.608	8.333333	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...
...	NaN	NaN	297.0	3.676768	2.145840	1.0000	2.0000	3.0000	5.0000	14.000000
...	NaN	NaN	297.0	0.150842	0.211113	0.0000	0.0000	0.0000	0.2000	0.800000
...	NaN	NaN	297.0	16.122229	146.561443	-1811.0784	1.8792	6.8724	22.4316	909.981800
...	NaN	NaN	297.0	1319.851684	8219.660616	0.8520	44.8200	137.0880	655.1280	135830.880000
...	NaN	NaN	297.0	5.061956	18.857204	-87.5000	2.0000	8.0000	13.1250	47.993827

We have alculated basic statistics such as mean, median, mode, standard deviation, etc., for numerical columns (e.g., Sales, Quantity, Profit).

ii. Segmented Summarization:

```
# Calculate summary statistics by segment
segment_summary = merged_df.groupby('Segment').agg({
    'Sales': 'sum',
    'Profit': 'mean',
    'Quantity': 'sum'
}).reset_index()

# Rename the index to match the columns of the original DataFrame
segment_summary = segment_summary.rename(columns={'Sales': 'Sales_Summary',
                  'Profit': 'Profit_Mean', 'Quantity': 'Quantity_Summary'})

# Add the segment summary row to the original DataFrame
merged_df = pd.concat([merged_df, segment_summary])

# Display the updated DataFrame with segment summary
merged_df
```

Here we have summarized the data based on different segments (e.g., Segment, Region, Category, Sub-Category).

STEP 3

- ## Data Loading:

For loading the data into any database getting the csv file of the merged dataframe i.e 'merged df'.

```
# Saving the merged DataFrame to a new JSON file
merged_df.to_csv('merged_data.csv')
```

To connect to MongoDB from Google Colab and send a CSV file to a MongoDB database, we need to use the pymongo library for Python

```
import pymongo
import json

# Replace these values with your MongoDB Atlas connection string
mongo_uri = "mongodb+srv://abdulrafay12364:HdWJXMIazvKvBjKJ@cluster0.818yghy.mongodb.net/?retryWrites=true&w=majority&ssl=true"
database_name = "etl"
collection_name = "etl_updated"

# Connect to MongoDB Atlas
client = pymongo.MongoClient(mongo_uri)

# Access the database and collection
db = client[database_name]
collection = db[collection_name]

# Path to your JSON file
json_file_path = "merged_data.json"

# Load JSON data from file
with open(json_file_path, 'r') as file:
    json_data = json.load(file)

# Insert JSON data into MongoDB collection
collection.insert_many(json_data)

# Close the MongoDB connection
client.close()

print("Data loaded Succesfully!!")
```

STEP 4

Data Warehousing Queries:

Assuming we have a fact table named 'Sales_Fact' and a dimension table named 'Time_Dimension'. The fact table has columns like 'Profit,' 'Sales,' and a foreign key 'Time_ID' referencing 'Time_Dimension'

- **Derived Attributes:**

  In data warehousing, derived attributes are often calculated or derived from existing attributes in the data. To illustrate the concept, let's consider a scenario where you have a fact table containing sales data with columns like 'Profit' and 'Sales.' You want to create a derived attribute 'Profit Margin,' which is calculated based on the 'Profit' and 'Total Sales,' which is the sum of sales over a period.

I. **First we create a derived attribute 'Total Sales' by summing sales over a period (e.g., a month)**

```
CREATE TABLE Derived_Attributes AS
SELECT
   Time_Dimension.Month,
   SUM(Sales_Fact.Sales) AS Total_Sales
FROM
   Sales_Fact
JOIN
   Time_Dimension ON Sales_Fact.Time_ID = Time_Dimension.Time_ID
GROUP BY
   Time_Dimension.Month;
```

II. **Create a derived attribute 'Profit Margin' by calculating the profit margin percentage**

```
CREATE TABLE Derived_Attributes AS
SELECT
   Time_Dimension.Month,
   Total_Sales,
   SUM(Sales_Fact.Profit) AS Total_Profit,
   (SUM(Sales_Fact.Profit) / Total_Sales) * 100 AS Profit_Margin
FROM
   Sales_Fact
JOIN
   Time_Dimension ON Sales_Fact.Time_ID = Time_Dimension.Time_ID
JOIN
   Derived_Attributes ON Derived_Attributes.Month = Time_Dimension.Month
GROUP BY
   Time_Dimension.Month, Total_Sales;
```

- **Roll Up:**
  A roll-up query aggregates data at a higher level of hierarchy or dimension, summarizing values. For example, rolling up monthly sales data to quarterly or yearly totals.

I. **By Month within a year**

```
SELECT
   YEAR(OrderDate) AS Year,
   MONTH(OrderDate) AS Month,
   SUM(Sales) AS TotalSales,
   AVG(Profit) AS AvgProfit
FROM
   SalesData
GROUP BY
   YEAR(OrderDate), MONTH(OrderDate);
```

II. **By Product Category**

```
SELECT
   ProductCategory,
   SUM(Sales) AS TotalSales,
   AVG(Profit) AS AvgProfit
FROM
   SalesData
GROUP BY
   ProductCategory;
```

- **Drill Down:**
  A drill-down query provides detailed information by breaking down aggregated data into a lower level of granularity. For instance, drilling down yearly sales to monthly or daily details.

I. By Product Sub-category

```
SELECT
   ProductCategory,
```

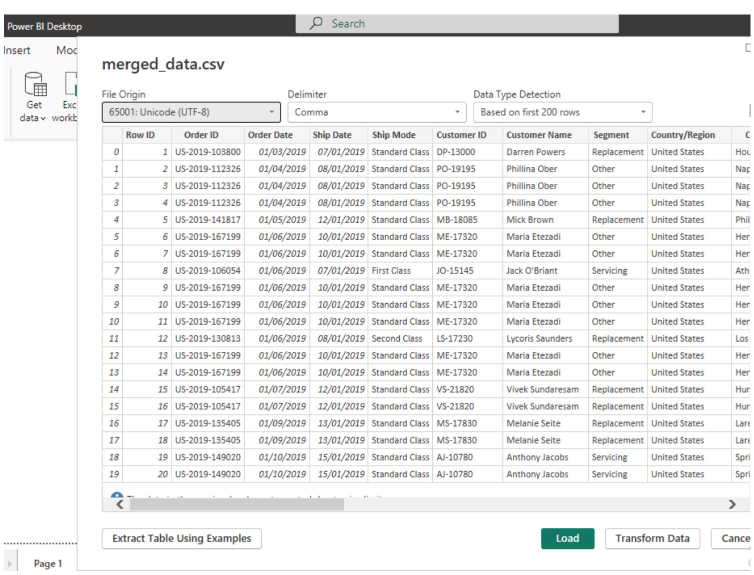
```
    ProductSubcategory,
    SUM(Sales) AS TotalSales,
    AVG(Profit) AS AvgProfit
FROM
    SalesData
GROUP BY
    ProductCategory, ProductSubcategory;
```

II. By Day within a month

```
SELECT
    YEAR(OrderDate) AS Year,
    MONTH(OrderDate) AS Month,
    DAY(OrderDate) AS Day,
    SUM(Sales) AS TotalSales,
    AVG(Profit) AS AvgProfit
FROM
    SalesData
GROUP BY
    YEAR(OrderDate), MONTH(OrderDate), DAY(OrderDate);
```

STEP 5

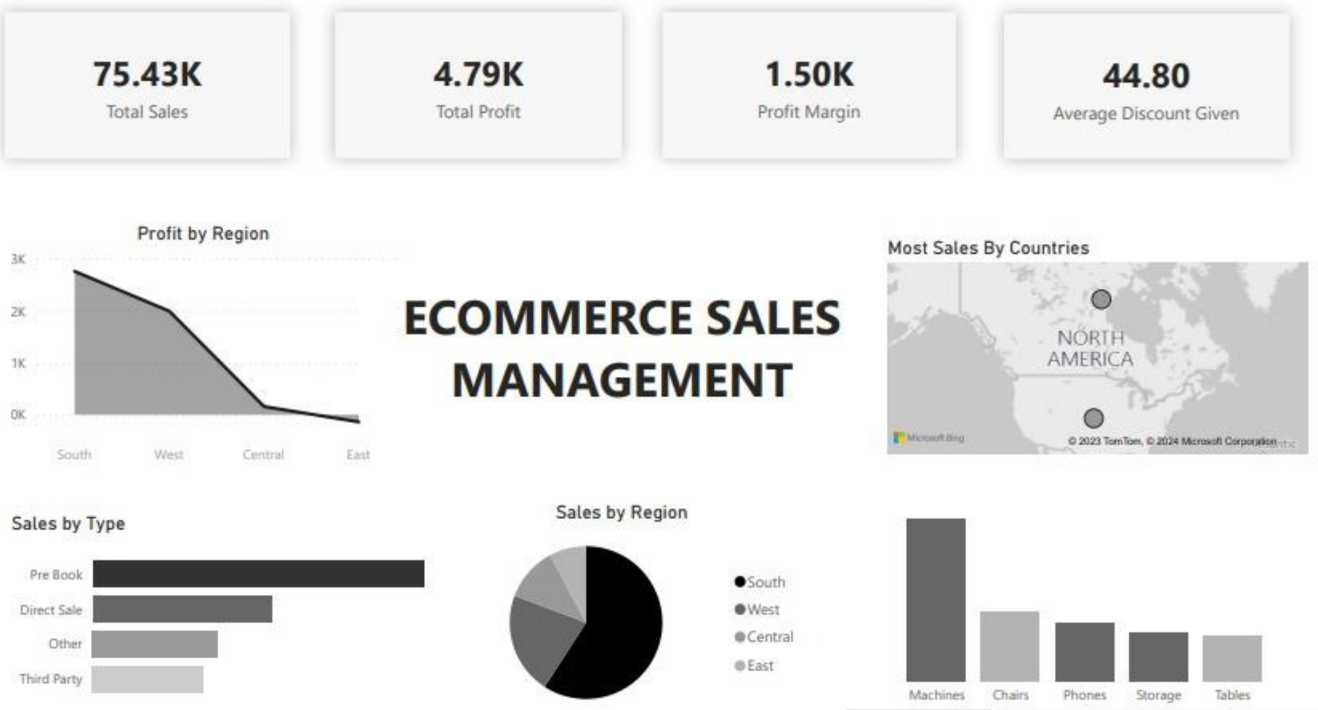
Import Results into Power BI:



	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country/Region
0	1	US-2019-103800	01/03/2019	07/01/2019	Standard Class	DP-13000	Darren Powers	Replacement	United States
1	2	US-2019-112326	01/04/2019	08/01/2019	Standard Class	PO-19195	Phillina Ober	Other	United States
2	3	US-2019-112326	01/04/2019	08/01/2019	Standard Class	PO-19195	Phillina Ober	Other	United States
3	4	US-2019-112326	01/04/2019	08/01/2019	Standard Class	PO-19195	Phillina Ober	Other	United States
4	5	US-2019-141817	01/05/2019	12/01/2019	Standard Class	MB-18085	Mick Brown	Replacement	United States
5	6	US-2019-167199	01/06/2019	10/01/2019	Standard Class	ME-17320	Maria Etezadi	Other	United States
6	7	US-2019-167199	01/06/2019	10/01/2019	Standard Class	ME-17320	Maria Etezadi	Other	United States
7	8	US-2019-106054	01/06/2019	07/01/2019	First Class	JO-15145	Jack O'Briant	Servicing	United States
8	9	US-2019-167199	01/06/2019	10/01/2019	Standard Class	ME-17320	Maria Etezadi	Other	United States
9	10	US-2019-167199	01/06/2019	10/01/2019	Standard Class	ME-17320	Maria Etezadi	Other	United States
10	11	US-2019-167199	01/06/2019	10/01/2019	Standard Class	ME-17320	Maria Etezadi	Other	United States
11	12	US-2019-130813	01/06/2019	08/01/2019	Second Class	LS-17230	Lycoris Saunders	Replacement	United States
12	13	US-2019-167199	01/06/2019	10/01/2019	Standard Class	ME-17320	Maria Etezadi	Other	United States
13	14	US-2019-167199	01/06/2019	10/01/2019	Standard Class	ME-17320	Maria Etezadi	Other	United States
14	15	US-2019-105417	01/07/2019	12/01/2019	Standard Class	VS-21820	Vivek Sundaresam	Replacement	United States
15	16	US-2019-105417	01/07/2019	12/01/2019	Standard Class	VS-21820	Vivek Sundaresam	Replacement	United States
16	17	US-2019-135405	01/09/2019	13/01/2019	Standard Class	MS-17830	Melanie Seite	Replacement	United States
17	18	US-2019-135405	01/09/2019	13/01/2019	Standard Class	MS-17830	Melanie Seite	Replacement	United States
18	19	US-2019-149020	01/10/2019	15/01/2019	Standard Class	AJ-10780	Anthony Jacobs	Servicing	United States
19	20	US-2019-149020	01/10/2019	15/01/2019	Standard Class	AJ-10780	Anthony Jacobs	Servicing	United States

STEP 6 and 7

Power BI Dashboard:



Conclusion:

In short, Implementation of ETL pipeline was all about handling different sets of data. First, we collected information from various places, like Product Sales, Customer Orders, and Sales Orders datasets. Then, we looked closely at each dataset to understand things like product types, customer groups, and transaction details.

After getting the data, we made some changes to make it better and more useful. For example, in the Product Sales dataset, we learned about sales and products. The Customer Orders dataset showed us how customers interact, and the Sales Orders dataset gave details about sales transactions.

The last step was bringing all this data together into MongoDB Atlas. This made a combined and improved dataset. To make it easier to understand, we also created a PowerBI dashboard. This simple and clear tool helps see important information easily. This project not only shows we know how to handle data well but also how to use it to make smart decisions in our work.