## IMG_256

### Colleague of computing

### Department of software engineering

Course name : Fundamental of Big Data analysis and Business intelligence

Course code :SEng5112

Title:Building an End-to-End Data Pipeline

Name : Samuel Amsalu

Id: 0448/13

### **Submitted to :Derbew F.**

**Submission date:02/06/2017**

### Overview

This script performs an end-to-end process of extracting, transforming, aggregating, and visualizing eCommerce data. It involves multiple stages:

1. **Data Extraction** using Pandas.
2. **Data Transformation** using PySpark.
3. **Aggregation** using DuckDB.
4. **Visualization** of the results using Matplotlib.

The final output of the script is a plot that shows the total sales and commission earned by car make.

### Step 1: Import Necessary Libraries

The script begins by importing the required libraries for data extraction, transformation, and visualization.

python

Copy

import pandas as pdimport duckdbfrom pyspark.sql import SparkSessionimport matplotlib.pyplot as plt

* **pandas**: Used for data extraction from CSV and basic manipulation.
* **duckdb**: A lightweight database for running SQL queries on in-memory data.
* **pyspark**: A distributed data processing framework for handling large datasets.
* **matplotlib**: A popular plotting library used to visualize the results.

### Step 2: Data Extraction with Pandas

The data extraction step involves loading the dataset from a CSV file using Pandas, performing basic inspections, and identifying any issues with missing data.

# Set the correct file path to where the CSV is located

file\_path = 'C:/Users/POST LAB/ecommerce\_project/ecommerce\_data.csv'

# Load the dataset using pandas

df = pd.read\_csv(file\_path)

# Display the first 5 rows to inspect the datasetprint("First 5 rows of the dataset (Pandas):")print(df.head())

# Get the shape of the dataset (rows and columns)print(f"\nShape of the dataset: {df.shape}")

# Check for missing valuesprint("\nMissing values in each column:")print(df.isnull().sum())

# Show basic statistics for numerical columnsprint("\nBasic statistics of numerical columns:")print(df.describe())

* df.head(): Displays the first 5 rows of the dataset.
* df.shape: Prints the dimensions (rows, columns) of the dataset.
* df.isnull().sum(): Checks for missing values in the dataset.
* df.describe(): Provides basic statistics for numerical columns, such as mean, standard deviation, min, and max values.

### Step 3: Data Transformation using PySpark

In this step, the script initializes a Spark session, loads the data into a Spark DataFrame, cleans the data by removing rows with missing values, and then inspects the cleaned data.

# Initialize PySpark session

spark = SparkSession.builder \

.appName("DataTransformation") \

.getOrCreate()

# Read the dataset into a Spark DataFrame

df\_spark = spark.read.csv(file\_path, header=True, inferSchema=True)

# Show the first few rows to check if data is loaded correctlyprint("\nFirst 5 rows from PySpark DataFrame:")

df\_spark.show(5)

# Remove rows with missing values

df\_spark\_clean = df\_spark.dropna()

# Show cleaned dataprint("\nCleaned data (no missing values) from PySpark:")

df\_spark\_clean.show(5)

# Check the schema (data types) of the cleaned dataprint("\nSchema of the cleaned PySpark DataFrame:")

df\_spark\_clean.printSchema()

* **Spark Session**: The session is initialized to perform distributed data processing using PySpark.
* spark.read.csv(): Reads the CSV data into a Spark DataFrame.
* df\_spark.dropna(): Removes rows with missing values from the Spark DataFrame.
* df\_spark\_clean.printSchema(): Displays the schema (column names and data types) of the cleaned DataFrame.

### Step 4: Convert PySpark DataFrame to Pandas DataFrame

After cleaning the data, we convert the Spark DataFrame into a Pandas DataFrame for further manipulation.

# Convert PySpark DataFrame to Pandas DataFrame

df\_clean\_pandas = df\_spark\_clean.toPandas()

* df\_spark\_clean.toPandas(): Converts the cleaned PySpark DataFrame to a Pandas DataFrame for easier processing and manipulation.

### Step 5: Load Data into DuckDB and Perform Aggregation

In this step, the Pandas DataFrame is loaded into an in-memory DuckDB instance, and a SQL query is run to calculate the total sales and commission by car make.

# Connect to DuckDB and create a table for the cleaned data

con = duckdb.connect(database=':memory:', read\_only=False)

# Register the DataFrame as a table in DuckDB

con.execute("CREATE TABLE ecommerce\_sales AS SELECT \* FROM df\_clean\_pandas")

# Perform an aggregation (e.g., total sales and commission by car make)

query = """

SELECT

"Car Make",

SUM("Sale Price") AS total\_sales,

SUM("Commission Earned") AS total\_commission

FROM ecommerce\_sales

GROUP BY "Car Make"

ORDER BY total\_sales DESC

"""

# Execute the query and fetch the result

result = con.execute(query).fetchall()

# Display the result of the aggregationprint("\nTotal Sales and Commission Earned by Car Make:")for row in result:

print(row)

* **DuckDB**: A fast in-memory database that supports SQL queries. We use DuckDB to perform SQL-based aggregation.
* **SQL Query**: Aggregates total sales and commissions by car make, ordered by total sales in descending order.
* con.execute(query).fetchall(): Executes the SQL query and fetches the result.

### ****Step 6: Visualize the Results****

This step visualizes the aggregated results, including **total sales** and **commission earned** by car make. The following visualizations are included:

#### ****1. Bar Chart with Line Plot: Total Sales and Commission Earned by Car Make****

This visualization compares the **total sales** and **commission earned** by each car make.

* A **bar chart** (on the left y-axis) represents **total sales** for each car make.
* A **line plot** (on the right y-axis) shows **total commission earned** for each car make, with a dashed orange line.

# Create a figure with two subplots

fig, ax1 = plt.subplots(figsize=(10, 6))

# Plot total sales as a bar chart

ax1.bar(car\_makes, total\_sales, color='skyblue')

ax1.set\_xlabel('Car Make')

ax1.set\_ylabel('Total Sales ($)', color='skyblue')

ax1.tick\_params(axis='y', labelcolor='skyblue')

# Create a second y-axis to plot the total commission

ax2 = ax1.twinx()

ax2.plot(car\_makes, total\_commission, color='orange', marker='o', linestyle='dashed')

ax2.set\_ylabel('Total Commission Earned ($)', color='orange')

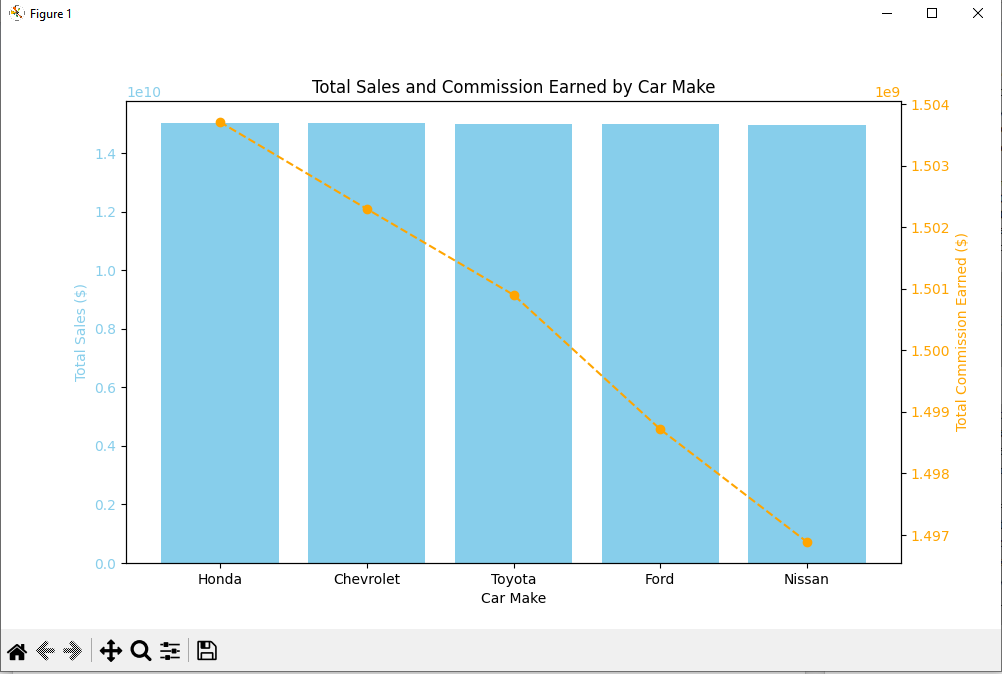
ax2.tick\_params(axis='y', labelcolor='orange')

# Set the title

plt.title('Total Sales and Commission Earned by Car Make')

plt.show()

* ax1.bar(): Plots the total sales as a bar chart.
* ax2.plot(): Plots the total commission as a dashed line on a secondary y-axis.
* **Two y-axes**: Left for total sales, right for total commission.



#### ****2. Pie Chart: Sales Distribution by Car Make****

This chart shows how total sales are distributed among different car makes.

* The **pie chart** shows the relative percentage of total sales contributed by each car make.

fig, ax = plt.subplots(figsize=(8, 8))

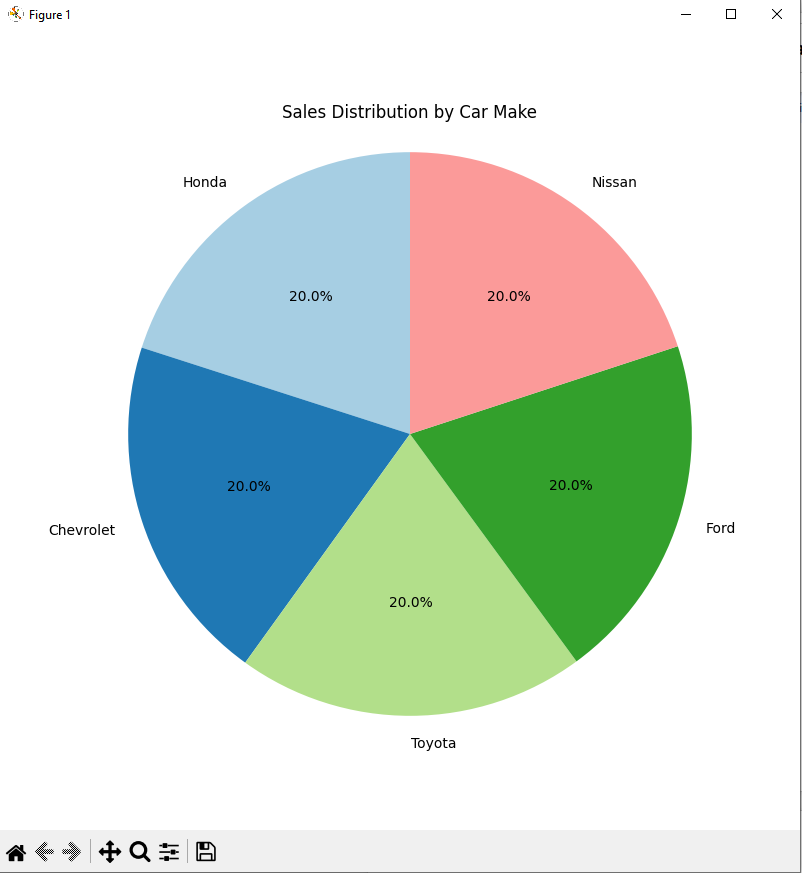
ax.pie(total\_sales, labels=car\_makes, autopct='%1.1f%%', startangle=90, colors=plt.cm.Paired.colors)

ax.axis('equal') # Equal aspect ratio ensures the pie chart is circular.

plt.title('Sales Distribution by Car Make')

plt.show()

* autopct='%1.1f%%': Shows the percentage of total sales for each car make.
* colors=plt.cm.Paired.colors: Ensures distinct colors for each car make in the pie chart.



#### ****3. Stacked Bar Chart: Total Sales and Commission Earned by Car Make****

This chart compares both **total sales** and **commission earned** by each car make using a stacked bar representation.

* The **stacked bar chart** shows total sales as the base, and commission earned stacked on top of sales.

fig, ax = plt.subplots(figsize=(10, 6))

# Plot stacked bar chart

ax.bar(car\_makes, total\_sales, label='Total Sales', color='skyblue')

ax.bar(car\_makes, total\_commission, bottom=total\_sales, label='Commission Earned', color='orange')

# Set labels and title

ax.set\_xlabel('Car Make')

ax.set\_ylabel('Amount ($)')

ax.set\_title('Stacked Bar Chart: Total Sales and Commission by Car Make')

# Display legend

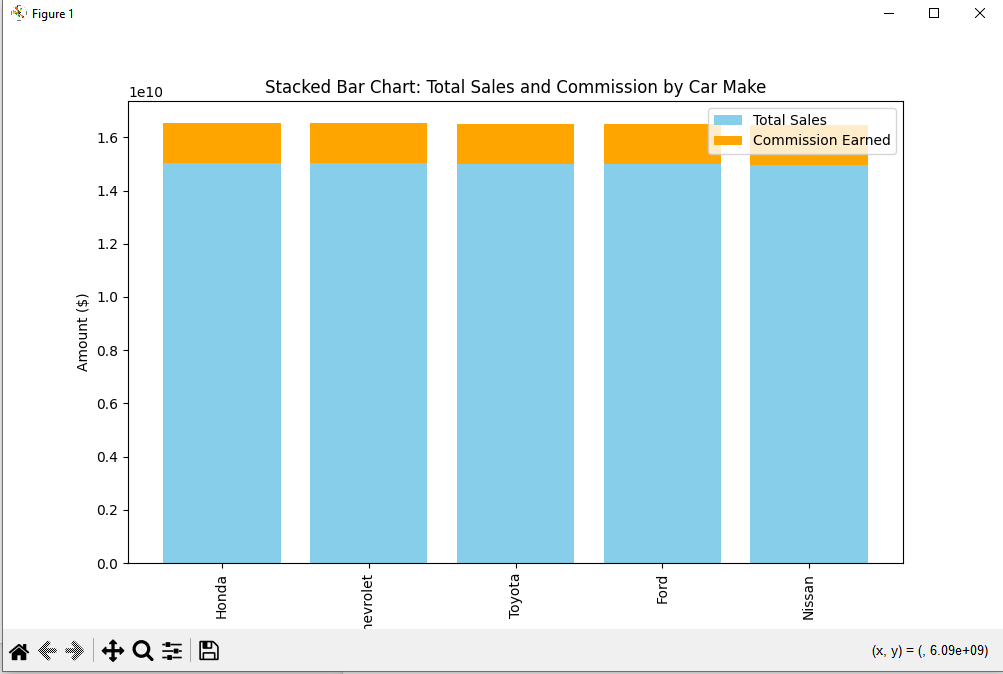
ax.legend()

# Display the plot

plt.xticks(rotation=90)

plt.show()

* **Stacked bars**: Allow easy comparison of sales and commission for each car make.
* bottom=total\_sales: Ensures the commission is stacked on top of the sales values.



#### ****4. Area Chart: Total Sales and Commission Earned by Car Make****

This chart visually compares **total sales** and **commission earned** using overlapping areas.

* The **area chart** uses the fill\_between() function to visualize total sales and commission earned with transparency to show overlap.

fig, ax = plt.subplots(figsize=(10, 6))

# Plot area chart

ax.fill\_between(car\_makes, total\_sales, color='skyblue', alpha=0.5, label='Total Sales')

ax.fill\_between(car\_makes, total\_commission, color='orange', alpha=0.5, label='Commission Earned')

# Set labels and title

ax.set\_xlabel('Car Make')

ax.set\_ylabel('Amount ($)')

ax.set\_title('Area Chart: Total Sales and Commission by Car Make')

# Display legend

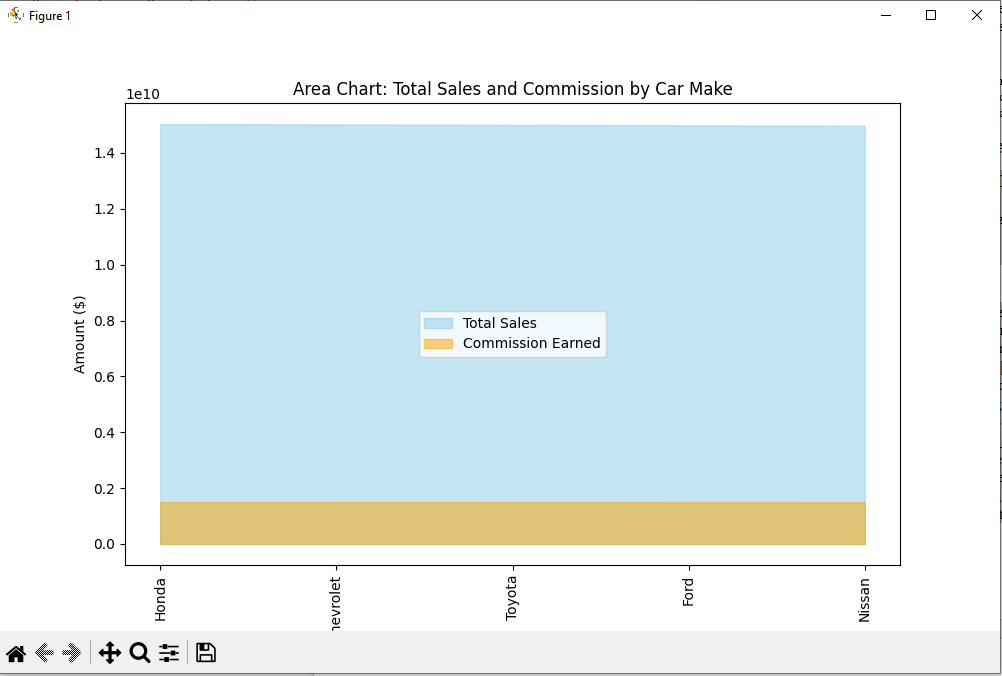
ax.legend()

# Display the plot

plt.xticks(rotation=90)

plt.show()

* alpha=0.5: Adds transparency to the areas, so both datasets can be seen overlapping.
* This chart helps in visually comparing the cumulative amounts for total sales and commission earned.



#### ****5. Venn Diagram: Sales and Commission****

This Venn diagram visualizes the overlap between **car makes with sales** and **car makes with commission earned**.

* The **Venn diagram** represents two sets: car makes with **sales** and car makes with **commission earned**.

# Defining car makes for each set

set\_sales = set(car\_makes[:10]) # Top 10 car makes by sales

set\_commission = set(car\_makes[5:15]) # Top 10 car makes by commission (assuming overlap)

# Create a Venn diagram showing the intersection of car makes with sales and commission

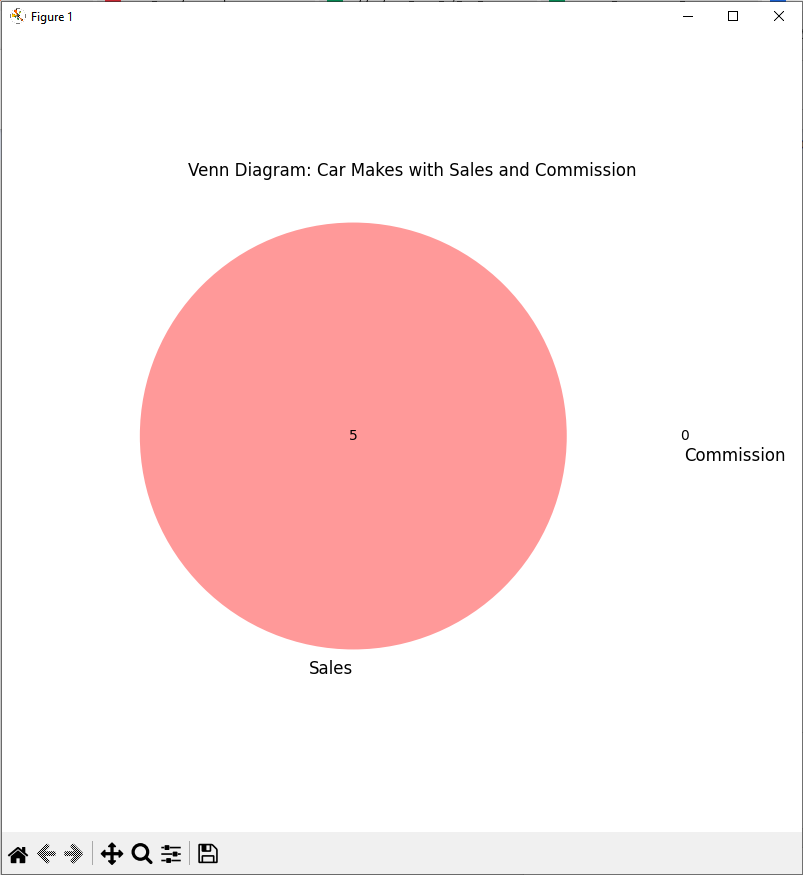
plt.figure(figsize=(8, 8))

venn2([set\_sales, set\_commission], set\_labels=('Sales', 'Commission'))

plt.title('Venn Diagram: Car Makes with Sales and Commission')

plt.show()

* venn2(): Draws the Venn diagram representing the intersection between the two sets.
* **Intersection**: The diagram shows which car makes appear in both the sales and commission sets.



### Closing the DuckDB Connection

# Close the DuckDB connection

con.close()

* con.close(): Closes the DuckDB connection once the data has been processed and the results have been visualized.

### ****code Repository****

The complete code for this assignment is available on GitHub:  
**[ ]**