## **Table of Contents**

1. **Introduction**
2. **Dataset and Processing**
3. **Analysis Pipeline**
4. **Visualization and Insights**
5. **Report Generation**
6. **How to run the code?**
7. **Output**
8. **Future Enhancements**

## **Chapter 1: Introduction**

The **Sales Data Analysis and Visualization Dashboard** is a project designed to analyze sales data and present actionable insights via an interactive dashboard. By using Python, Pandas, Matplotlib, Seaborn, Plotly, and Dash, this project highlights trends, customer behavior, and key business metrics.

Project Outline

```

Sales\_Dashboard\_Project/

├── data/

│ ├── sales\_data.csv

│ └── cleaned\_sales\_data.csv

├── scripts/

│ ├── \_\_init\_\_.py

│ ├── load\_and\_preprocess\_data.py

│ ├── analysis.py

│ ├── static\_visualizations.py

│ └── dash\_app.py

├── visualizations/

├── outputs/

├── requirements.txt

├── README.md

└── app\_main.py

## 

## **Chapter 2: Dataset and Processing**

#### **1. Dataset Description**

The dataset contains sales transactions from a retail business with columns like:

* **Invoice Details**: Invoice ID, Date, and Time
* **Sales Metrics**: Unit price, Quantity, Total, Tax 5%, gross income
* **Customer Info**: City, Gender, Customer type
* **Product Info**: Branch, Product line, and Payment

#### **2. Code: Data Loading and Preprocessing**

**File:** scripts/load\_and\_preprocess\_data.py

| import pandas as pd import logging  *# Configure logging* logging.basicConfig(level=logging.INFO)  def load\_and\_preprocess\_data(file\_path):  """Load and preprocess sales data."""  try:  logging.info("Loading data...")  df = pd.read\_csv(file\_path) *# Load data*    *# Check for missing values*  if df.isnull().sum().any():  logging.info("Missing values detected. Filling them...")  df.fillna(method='ffill', inplace=True) *# Fill missing values*    *# Remove duplicates*  duplicates = df.duplicated(subset='Invoice ID').sum()  if duplicates > 0:  logging.info(f"Found {duplicates} duplicates. Removing them...")  df.drop\_duplicates(subset='Invoice ID', inplace=True)   *# Feature Engineering*  logging.info("Generating additional time-based features...")  df['Date'] = pd.to\_datetime(df['Date'])  df['Month'] = df['Date'].dt.month\_name()  df['Hour'] = pd.to\_datetime(df['Time']).dt.hour  return df   except Exception as e:  logging.error(f"Error processing data: {e}")  return None |
| --- |

**Explanation:**

1. **Data Loading**: Reads the dataset into a Pandas DataFrame.
2. **Handling Missing Values**:
   * Checks for NaN values using isnull().
   * Fills missing values using forward fill (ffill).
3. **Removing Duplicates**: Identifies and removes duplicate rows using duplicated() and drop\_duplicates().
4. **Feature Engineering**:
   * Converts Date to a datetime object.
   * Adds columns for Month and Hour for trend analysis.

## **Chapter 3: Analysis Pipeline**

#### **1. Code: Analyzing Sales Data**

**File:** scripts/analysis.py

| def analyze\_data(df):  """Analyze sales data to calculate key metrics."""  return {  'total\_sales': df['Total'].sum(), *# Calculate total revenue*  'average\_sales': df['Total'].mean(), *# Calculate average sale*  'sales\_by\_city': df.groupby('City')['Total'].sum(), *# Revenue by city*  'sales\_by\_product': df.groupby('Product line')['Total'].sum(), *# Revenue by product*  'monthly\_sales': df.groupby('Month')['Total'].sum() *# Monthly revenue*  } |
| --- |

**Explanation:**

* Groups data by categories like City, Product line, and Month using Pandas' groupby().
* Aggregates Total sales using sum() to compute total revenue by group.
* Calculates averages for metrics like sales.

## **Chapter 4: Visualization and Insights**

#### **1. Code: Static Visualizations**

**File:** scripts/static\_visualizations.py

### **1. Import Libraries**

| import os *# For directory and file management* import pandas as pd *# Data manipulation and analysis* from scripts.load\_and\_preprocess\_data import load\_and\_preprocess\_data *# Custom preprocessing function* from scripts.analysis import analyze\_data *# Custom analysis function* import matplotlib.pyplot as plt *# Visualization library for static plots* import seaborn as sns *# Advanced visualization library for heatmaps and styled plots* |
| --- |

* **Purpose**: These libraries are required for creating and saving static visualizations.
  + os: Helps create directories and manage file paths.
  + pandas: Used for grouping and manipulating the data.
  + matplotlib and seaborn: Libraries for generating visualizations.

### **2. Define the Function**

| def create\_and\_save\_visualizations(df, output\_dir):  """  Generate static visualizations and save them to the specified directory.   Args:  df (pd.DataFrame): Cleaned sales data.  output\_dir (str): Directory to save the visualizations.  """  os.makedirs(output\_dir, exist\_ok=True) *# Ensure the output directory exists* |
| --- |

* **create\_and\_save\_visualizations()**: This function generates and saves static visualizations from the cleaned sales data.
* **os.makedirs(output\_dir, exist\_ok=True)**: Creates the output\_dir if it doesn’t exist.

### **3. Visualization 1: Sales Trend Over Time**

| *# Sales Trend Over Time*  sales\_trend = df.groupby('Date')['Total'].sum() *# Group data by date and calculate total sales*  plt.figure(figsize=(12, 6)) *# Set the figure size*  sales\_trend.plot(title='Sales Trend Over Time', color='blue') *# Plot the trend as a line chart*  plt.xlabel('Date') *# Label for x-axis*  plt.ylabel('Total Sales') *# Label for y-axis*  plt.grid(True) *# Add gridlines for better readability*  sales\_trend\_path = os.path.join(output\_dir, 'sales\_trend\_over\_time.png') *# Define the output path*  plt.savefig(sales\_trend\_path) *# Save the plot as a PNG file*  print(f"Saved: {sales\_trend\_path}") *# Print confirmation*  plt.close() *# Close the figure to free memory* |
| --- |

#### **Explanation:**

1. **df.groupby('Date')['Total'].sum()**:
   * Groups the data by the Date column.
   * Calculates the sum of the Total column (total sales) for each date.
2. **plt.figure(figsize=(12, 6))**:
   * Sets the size of the figure to 12 inches wide and 6 inches tall.
3. **sales\_trend.plot()**:
   * Plots a line chart showing the trend of total sales over time.
4. **plt.savefig()**:
   * Saves the plot to the specified output path (sales\_trend\_over\_time.png).
5. **plt.close()**:
   * Closes the current plot to release memory and avoid overlapping plots.

### **4. Visualization 2: Heatmap of Sales by Region and Product Line**

| *# Heatmap of Sales by Region and Product Line*  region\_product\_sales = df.pivot\_table(  index='City', columns='Product line', values='Total', aggfunc='sum'  ) *# Create a pivot table to summarize sales*  plt.figure(figsize=(12, 8)) *# Set the figure size*  sns.heatmap(  region\_product\_sales, annot=True, fmt='.1f', cmap='YlGnBu', linewidths=0.5  ) *# Generate a heatmap*  plt.title('Sales Heatmap by Region and Product Line') *# Title of the plot*  plt.xlabel('Product Line') *# Label for x-axis*  plt.ylabel('City') *# Label for y-axis*  heatmap\_path = os.path.join(output\_dir, 'sales\_heatmap\_region\_product.png') *# Output path*  plt.savefig(heatmap\_path) *# Save the heatmap as a PNG file*  print(f"Saved: {heatmap\_path}") *# Print confirmation*  plt.close() *# Close the figure* |
| --- |

#### **Explanation:**

1. **df.pivot\_table()**:
   * Creates a pivot table where rows are City, columns are Product line, and values are the sum of Total sales.
2. **sns.heatmap()**:
   * Visualizes the pivot table as a heatmap with annotations (annot=True).
   * **fmt='.1f'**: Formats the values with one decimal place.
   * **cmap='YlGnBu'**: Sets the color scheme to yellow-green-blue.
   * **linewidths=0.5**: Adds a thin border between cells for better readability.
3. **plt.savefig()**:
   * Saves the heatmap to the output path (sales\_heatmap\_region\_product.png).

### **5. Visualization 3: Sales by Month**

| *# Sales by Month*  sales\_by\_month = df.groupby('Month')['Total'].sum().sort\_index() *# Group data by month and calculate total sales*  plt.figure(figsize=(12, 6)) *# Set the figure size*  sales\_by\_month.plot(kind='bar', color='orange', title='Total Sales by Month') *# Plot a bar chart*  plt.xlabel('Month') *# Label for x-axis*  plt.ylabel('Total Sales') *# Label for y-axis*  sales\_by\_month\_path = os.path.join(output\_dir, 'sales\_by\_month.png') *# Output path*  plt.savefig(sales\_by\_month\_path) *# Save the bar chart as a PNG file*  print(f"Saved: {sales\_by\_month\_path}") *# Print confirmation*  plt.close() *# Close the figure* |
| --- |

#### **Explanation:**

1. **df.groupby('Month')['Total'].sum()**:
   * Groups the data by Month and calculates the sum of Total sales for each month.
2. **sales\_by\_month.plot(kind='bar')**:
   * Plots a bar chart to show total sales for each month.
   * **color='orange'**: Sets the bar color to orange.
3. **plt.savefig()**:
   * Saves the bar chart to the output path (sales\_by\_month.png).

### **6. Summary of the Function**

| create\_and\_save\_visualizations(df, output\_dir) |
| --- |

#### **Steps Performed by the Function:**

1. **Create Output Directory**:
   * Ensures the specified output directory exists.
2. **Generate Visualizations**:
   * Creates three visualizations:
     + **Sales Trend Over Time**: A line chart.
     + **Sales Heatmap**: A heatmap comparing sales by region and product line.
     + **Sales by Month**: A bar chart showing total sales by month.
3. **Save Visualizations**:
   * Saves each visualization as a .png file in the specified directory.
4. **Print Confirmation**:
   * Confirms that each visualization is saved.

#### **2. Code: Interactive Visualizations (Dash)**

**File:** scripts/dash\_app.py

Here’s a detailed explanation of the **Dash application code**, with comments and descriptions for each part of the code:

#### **2. 1. Import Libraries**

| from dash import Dash, dcc, html, Input, Output, State *# Dash components for interactivity and layout* import dash\_bootstrap\_components as dbc *# Bootstrap styling for responsive layouts* import pandas as pd *# Data handling and preprocessing* import plotly.express as px *# Plotly for interactive visualizations* import os *# OS operations* |
| --- |

* These libraries provide essential functionality for creating a Dash-based interactive web application.
* Dash: Framework for building web apps in Python.
* Dash Bootstrap Components (DBC): Adds Bootstrap styling for better design and layout.

### **2.2. Define the Dash App**

| def create\_dash\_app():  """Create and configure Dash application."""  app = Dash(\_\_name\_\_, external\_stylesheets=[dbc.themes.FLATLY]) |
| --- |

* **Dash(\_\_name\_\_)**: Initializes the Dash application.
* **external\_stylesheets**: Applies the "Flatly" Bootstrap theme for styling.

### **2.3. Load the Dataset**

| df = pd.read\_csv('data/cleaned\_sales\_data.csv', parse\_dates=['Date']) |
| --- |

* **pd.read\_csv()**: Loads the cleaned sales dataset.
* **parse\_dates=['Date']**: Ensures that the Date column is parsed as datetime objects for easier manipulation.

### **2.4. Layout of the Dashboard**

#### **2.4.1 Header Section**

| app.layout = dbc.Container([  dbc.Row([  dbc.Col(html.H1("Retail Sales Analytics Dashboard", className="text-center mb-4"), width=12)  ]), |
| --- |

* **Header**: Displays the title of the dashboard centered at the top.
* **dbc.Row and dbc.Col**: Bootstrap components for grid-based layout.

#### **2.4.2 Filters Section**

| *# Filters*  dbc.Row([  dbc.Col([dcc.DatePickerRange(  id='date-range',  start\_date=df['Date'].min(),  end\_date=df['Date'].max(),  display\_format='YYYY-MM-DD'  )], width=3),  dbc.Col([dcc.Dropdown(  id='city-select',  options=[{'label': city, 'value': city} for city in df['City'].unique()],  multi=True,  placeholder='Select Cities'  )], width=3),  dbc.Col([dcc.Dropdown(  id='product-select',  options=[{'label': product, 'value': product} for product in df['Product line'].unique()],  multi=True,  placeholder='Select Product Lines'  )], width=3),  dbc.Col([dcc.Dropdown(  id='gender-select',  options=[{'label': gender, 'value': gender} for gender in df['Gender'].unique()],  multi=True,  placeholder='Select Genders'  )], width=3)  ], className="mb-4"), |
| --- |

* **dcc.DatePickerRange**: Allows users to filter data by date range.
* **dcc.Dropdown**: Provides dropdown filters for selecting cities, product lines, and genders.
* **options**: Dynamically generated based on unique values in the dataset.

#### **2.4.3 Key Metrics Section**

| *# Key Metrics*  dbc.Row([  dbc.Col([dbc.Card([dbc.CardBody([html.H5("Total Sales"), html.H4(id='total-sales')])])], width=3),  dbc.Col([dbc.Card([dbc.CardBody([html.H5("Gross Income"), html.H4(id='gross-income')])])], width=3),  dbc.Col([dbc.Card([dbc.CardBody([html.H5("Total Quantity"), html.H4(id='total-quantity')])])], width=3),  dbc.Col([dbc.Card([dbc.CardBody([html.H5("Avg Rating"), html.H4(id='avg-rating')])])], width=3)  ], className="mb-4"), |
| --- |

* **Bootstrap Cards**: Display key business metrics such as Total Sales, Gross Income, Total Quantity, and Average Rating.
* **Dynamic Text Fields (id)**: Placeholders ('total-sales', 'gross-income', etc.) for callback functions to update metrics.

#### **2.4.4 Visualizations Section**

| *# Visualizations*  dbc.Row([  dbc.Col(dcc.Graph(id='sales-trend'), width=6),  dbc.Col(dcc.Graph(id='product-performance'), width=6)  ], className="mb-4"),  dbc.Row([  dbc.Col(dcc.Graph(id='payment-method'), width=4),  dbc.Col(dcc.Graph(id='customer-type'), width=4),  dbc.Col(dcc.Graph(id='hourly-sales'), width=4)  ], className="mb-4"), |
| --- |

* **dcc.Graph**: Placeholders for dynamic graphs. Each graph is updated via callbacks.

#### **2.4.5 Export Section**

| *# Export Section*  dbc.Row([  dbc.Col([html.Button("Export Filtered Data", id='export-btn', className="btn btn-primary"),  html.Div(id='export-status', className="mt-2")], width=12)  ]), |
| --- |

* **Export Button**: Allows filtered data to be exported to CSV.

#### **2.4.6 Store Component**

| dcc.Store(id='filtered-data')  ], fluid=True) |
| --- |

* **dcc.Store**: Stores filtered data for sharing across callbacks.

### **2.5. Callback Functions**

#### **2.5.1 Update Metrics**

| @app.callback(  [Output('total-sales', 'children'),  Output('gross-income', 'children'),  Output('total-quantity', 'children'),  Output('avg-rating', 'children'),  Output('filtered-data', 'data')],  [Input('date-range', 'start\_date'),  Input('date-range', 'end\_date'),  Input('city-select', 'value'),  Input('product-select', 'value'),  Input('gender-select', 'value')]  )  def update\_metrics(start\_date, end\_date, cities, products, genders):  filtered\_df = df.copy()    *# Apply filters*  filtered\_df = filtered\_df[(filtered\_df['Date'] >= start\_date) & (filtered\_df['Date'] <= end\_date)]  if cities: filtered\_df = filtered\_df[filtered\_df['City'].isin(cities)]  if products: filtered\_df = filtered\_df[filtered\_df['Product line'].isin(products)]  if genders: filtered\_df = filtered\_df[filtered\_df['Gender'].isin(genders)]    *# Calculate metrics*  return (  f"${filtered\_df['Total'].sum():,.2f}",  f"${filtered\_df['gross income'].sum():,.2f}",  f"{filtered\_df['Quantity'].sum():,}",  f"{filtered\_df['Rating'].mean():.1f}/10",  filtered\_df.to\_json(date\_format='iso', orient='split')  ) |
| --- |

* Filters the dataset based on user selections and computes:
  + **Total Sales**: Sum of the Total column.
  + **Gross Income**: Sum of the gross income column.
  + **Total Quantity**: Sum of Quantity.
  + **Average Rating**: Mean of Rating.

#### **2.5.2 Update Visualizations**

| @app.callback(  [Output('sales-trend', 'figure'),  Output('product-performance', 'figure'),  Output('payment-method', 'figure'),  Output('customer-type', 'figure'),  Output('hourly-sales', 'figure')],  [Input('filtered-data', 'data')]  )  def update\_visualizations(data):  filtered\_df = pd.read\_json(data, orient='split')  ...  *# (Visualizations logic as shown earlier)* |
| --- |

* Updates graphs dynamically based on filtered data.

#### **2.5.3 Export Filtered Data**

| @app.callback(  Output('export-status', 'children'),  [Input('export-btn', 'n\_clicks')],  [State('filtered-data', 'data')]  )  def export\_data(n\_clicks, data):  if n\_clicks:  pd.read\_json(data, orient='split').to\_csv('outputs/exported\_data.csv', index=False)  return "Data exported to outputs/exported\_data.csv"  return "" |
| --- |

* Exports filtered data to outputs/exported\_data.csv when the export button is clicked.

### **2.6. Return the App**

| return app |
| --- |

This function returns the fully configured Dash app for launching. Let me know if more details are needed!

### **Chapter 5: Report Generation**

#### **1. Export Filtered Data**

The Dash app includes an export feature where users can download filtered sales data as CSV files.

| @dash\_app.callback(  Output('export-status', 'children'),  Input('export-btn', 'n\_clicks'),  State('filtered-data', 'data') ) def export\_data(n\_clicks, data):  if n\_clicks:  df = pd.read\_json(data)  df.to\_csv('outputs/exported\_data.csv', index=False)  return "Data exported successfully!"  return "" |
| --- |

**Explanation**: Converts filtered data to CSV format for sharing or further analysis.

### **Chapter 6: How to Run the Code?**

### **1. Clone the Repository**

| git clone https://github.com/MasteriNeuron/Retail-Sales-Analytics.git cd retail-sales-analytics |
| --- |

#### **2. Create & Activate a Virtual Environment**

| conda create -n venv python=3.11 -y conda activate venv/ |
| --- |

#### **3. Install Dependencies**

| pip install -r requirements.txt |
| --- |

#### **4. Add Your Data**

* Place the raw sales data file (sales\_data.csv) in the data/ directory.
* Ensure the file has the required columns: Date, City, Product line, Total, Quantity, Payment, Gender, etc.

### **Steps to Run the Application**

#### **1. Preprocessing**

The script automatically cleans and preprocesses the raw data.

| python app.py |
| --- |

#### **2. Automated Workflow**

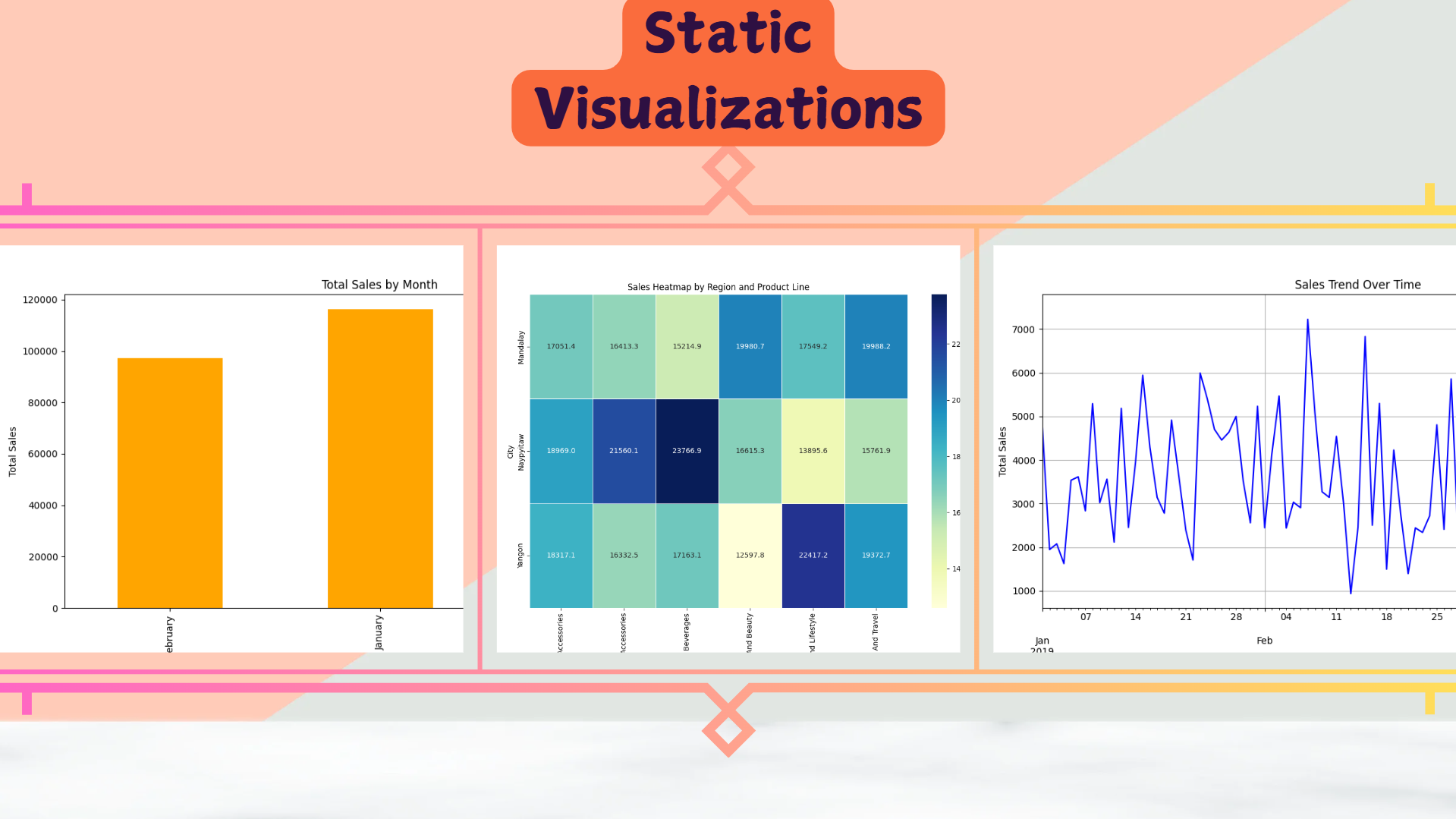
When you run app.py, the following steps are automated:

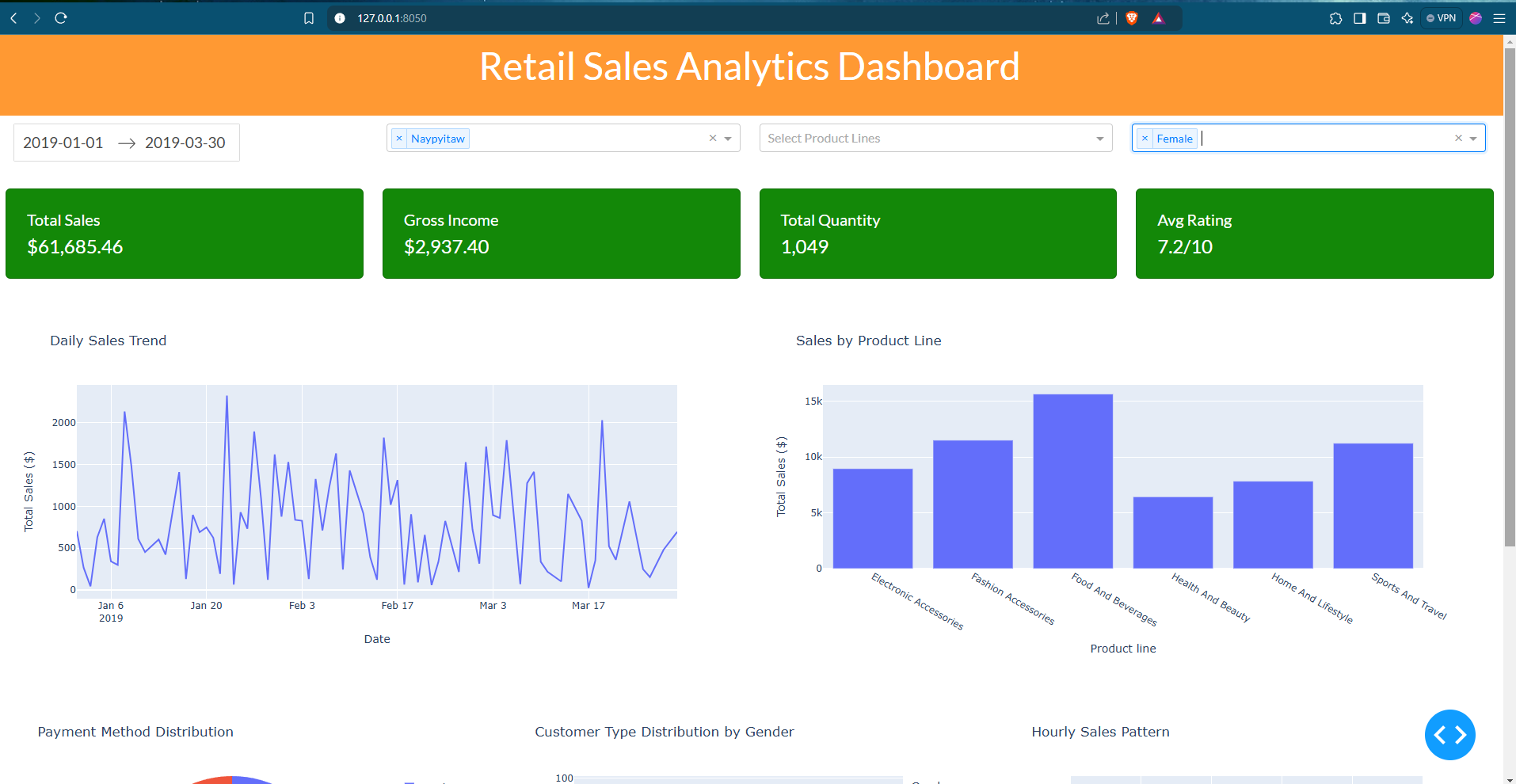
1. **Preprocessing**: Cleans raw data and saves it as data/cleaned\_sales\_data.csv.
2. **Analysis**: Outputs key metrics such as total sales and average sales in the console.
3. **Visualization**: Generates static visualizations and saves them in the visualizations/ folder.
4. **Dashboard**: Launches an interactive dashboard on http://localhost:8050.

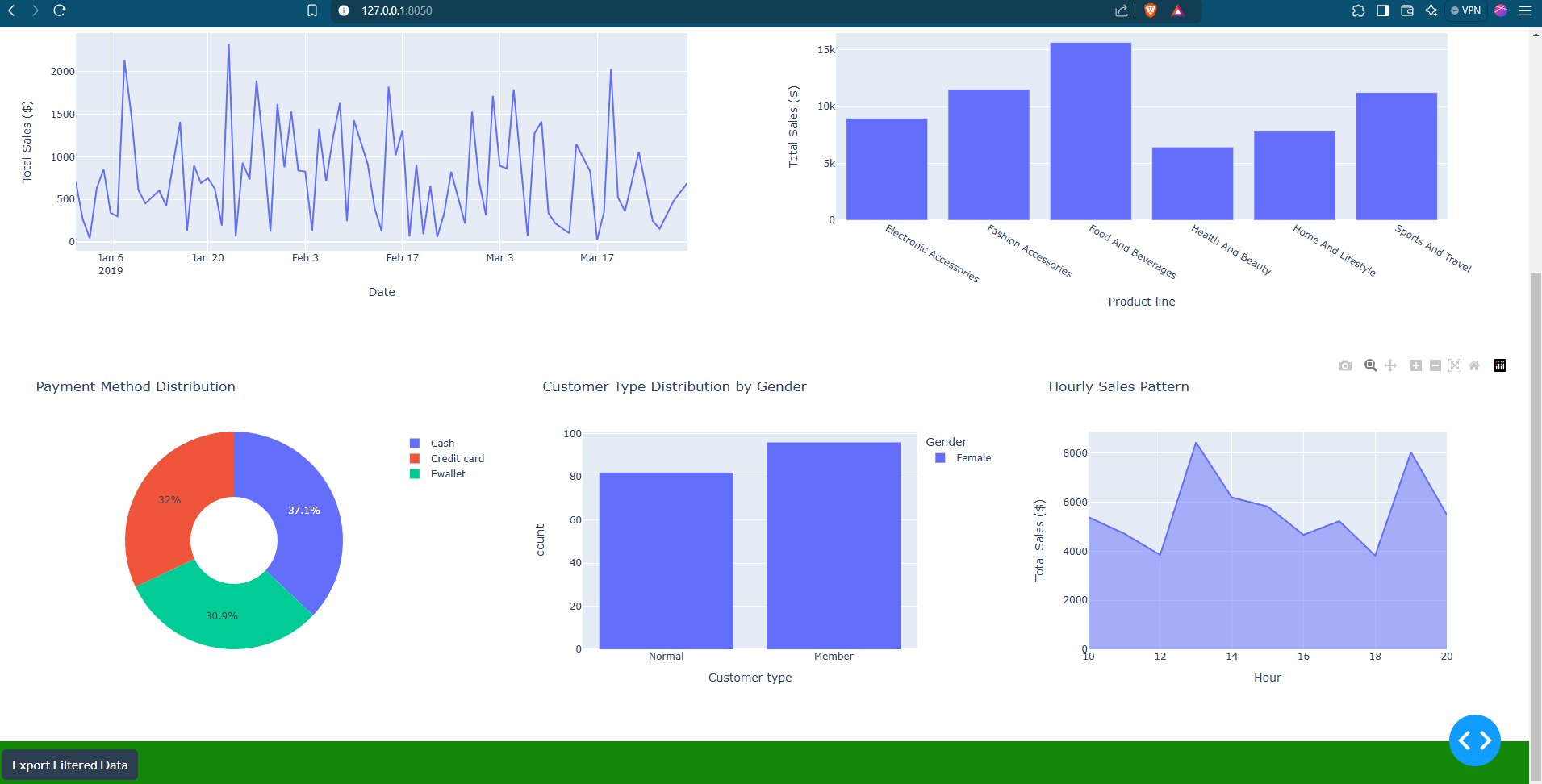
#### **3. Access the Dashboard**

* Open your browser and go to [http://localhost:8050](http://localhost:8050/) to explore the interactive dashboard.

### **Chapter 7: Output**







Note : Use Different filters and and at last export Filtered Data (Click on the Button).

& see the Outputs folder.

### **Chapter 8 : Future Enhancements**

### 

1. **Add User Authentication**:  
   Implement secure user authentication to restrict dashboard access and personalize user experiences.
2. **Enhance Data Visualization**:  
   Introduce more advanced and dynamic plots (e.g., geographical maps, trend forecasts, and interactive drilldowns) to provide deeper insights.
3. **Extend Support for Multi-File Data Preprocessing**:  
   Enable the application to handle and preprocess multiple input files, merging them seamlessly for comprehensive analysis.
4. **Deploy on Cloud Platforms**:  
   Host the application on a cloud platform like AWS, Render,Vercel or Azure to make it accessible from anywhere, ensuring scalability and availability.

THANK YOU !!