Zepto Sales Data Analysis Project Guide

Project Overview

Welcome, aspiring Data Analysts! This project is designed to give you hands-on experience with real-world, industry-grade sales data from Zepto, a popular quick-commerce platform. You will apply your Python skills, particularly using the Pandas and Matplotlib libraries, to explore, clean, analyze, and visualize a dataset containing over 200,000 sales records and a product catalog.

This project will reinforce your understanding of data manipulation, data cleaning techniques, and fundamental data visualization, preparing you for more complex analytical tasks.

Dataset Description

YouYou will be working with two CSV files:

1. `zepto_sales.csv`: Contains individual sales transactions.
2. `zepto_products.csv`: Contains details about the products sold.

`zepto_sales.csv` Columns:

- `order_id`: Unique identifier for each order.
- `order_date`: Date and time of the order.
- `product_id`: Identifier for the product sold in the transaction.
- `quantity`: Number of units of the product sold.
- `city`: City where the order was placed.
- `delivery_status`: Status of the delivery (e.g., Delivered, Cancelled, Returned).
- `customer_id`: Unique identifier for the customer.

- `delivery_time_mins` : Time taken for delivery in minutes.
- `total_amount` : Total amount of the transaction.

`zepto_products.csv` Columns:

- `product_id` : Unique identifier for the product.
- `product_name` : Name of the product.
- `category` : Category of the product.
- `base_price` : Base price of the product.

Project Steps

Step 1: Load and Initial Exploration

First, you need to load the datasets into Pandas DataFrames and perform an initial exploration to understand their structure and content.

```python
import pandas as pd

# Load the datasets
df_sales = pd.read_csv("zepto_sales.csv")
df_products = pd.read_csv("zepto_products.csv")

# Display basic information for sales data
print("\n--- Sales Data Info ---")
df_sales.info()
print("\n--- Sales Data Description ---")
print(df_sales.describe())
print("\n--- Sales Data Shape ---")
print(df_sales.shape)
print("\n--- Sales Data Head ---")
print(df_sales.head())
print("\n--- Sales Data Tail ---")
print(df_sales.tail())

# Display basic information for products data
print("\n--- Products Data Info ---")
df_products.info()
print("\n--- Products Data Description ---")
print(df_products.describe())
print("\n--- Products Data Head ---")
print(df_products.head())
```

Step 2: Data Cleaning - Handling Missing Values and Duplicates

Real-world data often contains missing values and duplicate entries. You will identify and handle these issues. Specifically, for missing values:

- For the `city` and `delivery_status` columns, remove the records (rows) that contain `NaN` values.
- For the `delivery_time_mins` column, fill missing values with the mean of the column.

```
# Check for null values in sales data
print("\n--- Null Values in Sales Data ---")
print(df_sales.isnull().sum())

# Handle null values in 'city' and 'delivery_status' by dropping rows
df_sales.dropna(subset=["city", "delivery_status"], inplace=True)
print("\nNulls after dropping rows in city/delivery_status:")
print(df_sales[["city", "delivery_status"]].isnull().sum())

# Handle null values in 'delivery_time_mins' by filling with the mean
mean_delivery_time = df_sales["delivery_time_mins"].mean()
df_sales["delivery_time_mins"].fillna(mean_delivery_time, inplace=True)
print("\nNulls after filling mean in delivery_time_mins:")
print(df_sales["delivery_time_mins"].isnull().sum())

# Check for duplicate records
print("\n--- Duplicate Records in Sales Data ---")
print(f"Number of duplicate rows: {df_sales.duplicated().sum()}")

# Remove duplicate records
df_sales.drop_duplicates(inplace=True)
print(f"Number of rows after removing duplicates: {df_sales.shape[0]}")

# Convert 'order_date' to datetime objects
df_sales["order_date"] = pd.to_datetime(df_sales["order_date"])
```

Step 3: Data Analysis - Aggregations and Grouping

Perform various aggregations and grouping operations to extract insights from the data.

```python
# Find min, max, and average total amount
min_amount = df_sales["total_amount"].min()
max_amount = df_sales["total_amount"].max()
avg_amount = df_sales["total_amount"].mean()
print(f"\nMin Total Amount: {min_amount:.2f}")
print(f"Max Total Amount: {max_amount:.2f}")
print(f"Average Total Amount: {avg_amount:.2f}")

# Top 5 products by total sales amount
top_products = df_sales.groupby("product_id")
["total_amount"].sum().nlargest(5)
print("\n--- Top 5 Products by Sales Amount ---")
print(top_products)

# Merge with product details to get product names
top_products_details = top_products.reset_index().merge(df_products,
on="product_id")
print("\n--- Top 5 Products by Sales Amount (with names) ---")
print(top_products_details[["product_name", "category", "total_amount"]])

# Total sales by city
sales_by_city = df_sales.groupby("city")
["total_amount"].sum().sort_values(ascending=False)
print("\n--- Total Sales by City ---")
print(sales_by_city)

# Average delivery time by city
avg_delivery_time_by_city = df_sales.groupby("city")
["delivery_time_mins"].mean().sort_values()
print("\n--- Average Delivery Time by City (minutes) ---")
print(avg_delivery_time_by_city)

# Sales trend over time (e.g., monthly sales)
df_sales["month"] = df_sales["order_date"].dt.to_period("M")
monthly_sales = df_sales.groupby("month")["total_amount"].sum()
print("\n--- Monthly Sales Trend ---")
print(monthly_sales)

# Sales by product category
sales_by_category = df_sales.merge(df_products, on="product_id")
sales_by_category = sales_by_category.groupby("category")
["total_amount"].sum().sort_values(ascending=False)
print("\n--- Total Sales by Product Category ---")
print(sales_by_category)
```

Step 4: Data Visualization with Matplotlib

Create various charts to visualize the insights gained from your analysis. Ensure your plots are clearly labeled and easy to understand.

```
import matplotlib.pyplot as plt
import seaborn as sns # For better aesthetics

sns.set_style("whitegrid")
plt.rcParams["figure.figsize"] = (10, 6)

# Plot 1: Top 5 Products by Sales Amount
plt.figure(figsize=(12, 7))
sns.barplot(x="product_name", y="total_amount", data=top_products_details,
palette="viridis")
plt.title("Top 5 Products by Total Sales Amount")
plt.xlabel("Product Name")
plt.ylabel("Total Sales Amount")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

# Plot 2: Total Sales by City
plt.figure(figsize=(12, 7))
sns.barplot(x=sales_by_city.index, y=sales_by_city.values, palette="magma")
plt.title("Total Sales by City")
plt.xlabel("City")
plt.ylabel("Total Sales Amount")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

# Plot 3: Monthly Sales Trend
plt.figure(figsize=(14, 7))
monthly_sales.plot(kind="line", marker="o", color="#3498DB")
plt.title("Monthly Sales Trend")
plt.xlabel("Month")
plt.ylabel("Total Sales Amount")
plt.grid(True)
plt.tight_layout()
plt.show()

# Plot 4: Sales by Product Category (Pie Chart)
plt.figure(figsize=(10, 10))
plt.pie(sales_by_category, labels=sales_by_category.index,
autopct="%1.1f%%", startangle=140, colors=sns.color_palette("pastel"))
plt.title("Total Sales by Product Category")
plt.axis("equal") # Equal aspect ratio ensures that pie is drawn as a
circle.
plt.tight_layout()
```

```
plt.show()

# Plot 5: Distribution of Delivery Times
plt.figure(figsize=(10, 6))
sns.histplot(df_sales["delivery_time_mins"], bins=20, kde=True,
color="#2ECC71")
plt.title("Distribution of Delivery Times")
plt.xlabel("Delivery Time (minutes)")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```

Conclusion

Upon completing this project, you will have successfully navigated a realistic data analysis workflow, from raw data to actionable insights and visualizations. This foundational experience is crucial for any data analyst. Remember to interpret your findings and consider what business questions these visualizations might answer for Zepto!