Experiment 2

Problem Statement: Data Visualization/ Exploratory data Analysis using Matplotlib and Seaborn.

Theory:

We first mounted Google Drive to the Colab environment to access files stored in drive. Then, we used read_csv to read the contents of cafe_sales.csv and load it into the DataFrame df for analysis.

```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

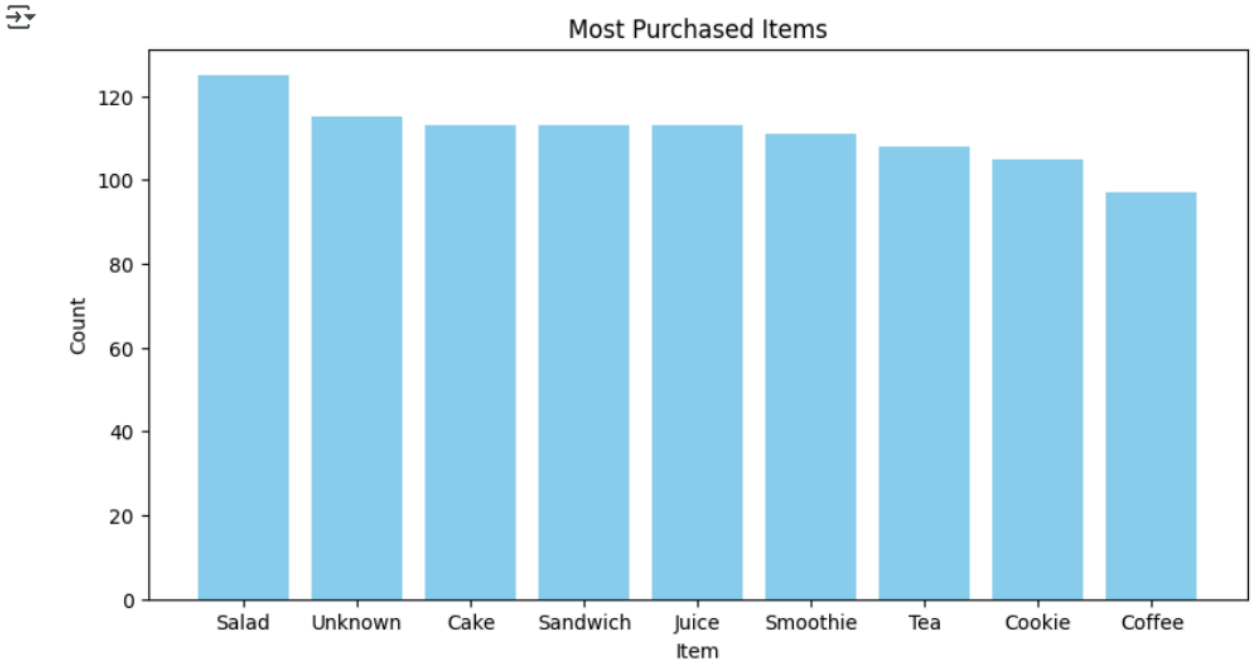
```
Mounted at /content/drive
```

```
[ ]  path="/content/drive/MyDrive/dataset-cafe.csv"
     df=pd.read_csv(path)
     df.head(5)
```

1. Create bar graph

This Python script uses **Matplotlib** and **Seaborn** to create a **bar graph** visualizing the most purchased items. It first imports the necessary libraries and defines two lists: `items`, containing different item names, and `counts`, representing their corresponding purchase frequencies. A **bar plot** is then created using `sns.barplot()`, with the x-axis displaying item names and the y-axis showing their counts. The figure size is adjusted for clarity, and labels for the x-axis, y-axis, and title are added to make the graph more informative. To improve readability, the x-axis labels are rotated by 45 degrees. Finally, `plt.show()` is used to render the graph, displaying a visually appealing and well-structured bar chart.

```
plt.figure(figsize=(10, 5))
item_counts = df_subset['Item'].value_counts()
plt.bar(item_counts.index, item_counts.values, color='skyblue')
#plt.xticks(rotation=45)
plt.xlabel("Item")
plt.ylabel("Count")
plt.title("Most Purchased Items")
plt.show()
```



2. Contingency table using any 2 features

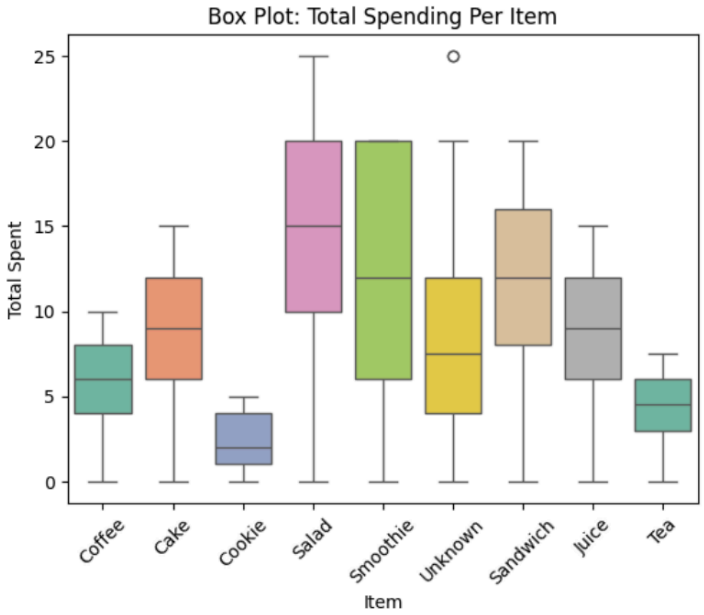
A contingency table in Python can be created using **pandas**. The code typically starts by importing pandas and loading the dataset. The pd.crosstab() function is then used to generate the table, where one categorical variable (e.g., "Item") is placed as rows and another (e.g., "Payment Method") as columns. This function counts occurrences of each combination, effectively summarizing relationships between the two variables. The table helps analyze patterns, such as which payment method is most used for each item.

Payment Method	Cash	Credit Card	Digital Wallet	Unknown
Item				
Cake	26	24	24	39
Coffee	29	19	24	25
Cookie	19	24	32	30
Juice	33	25	20	35
Salad	34	30	26	35
Sandwich	24	20	31	38
Smoothie	18	33	20	40
Tea	24	21	22	41
Unknown	26	26	27	36

3. Box Plot

a box plot that displays the distribution of total spending across different item categories. The x-axis represents various items such as Coffee, Cake, Cookie, Salad, and more, while the y-axis indicates the total amount spent. The box plot captures key statistical insights, including the median, interquartile range (IQR), and potential outliers. Taller boxes suggest greater variability in spending for that item. The code uses the `sns.boxplot()` function from Seaborn, applying a color palette (`Set2`) for differentiation. This visualization helps identify spending patterns, outliers, and item-wise expenditure trends.

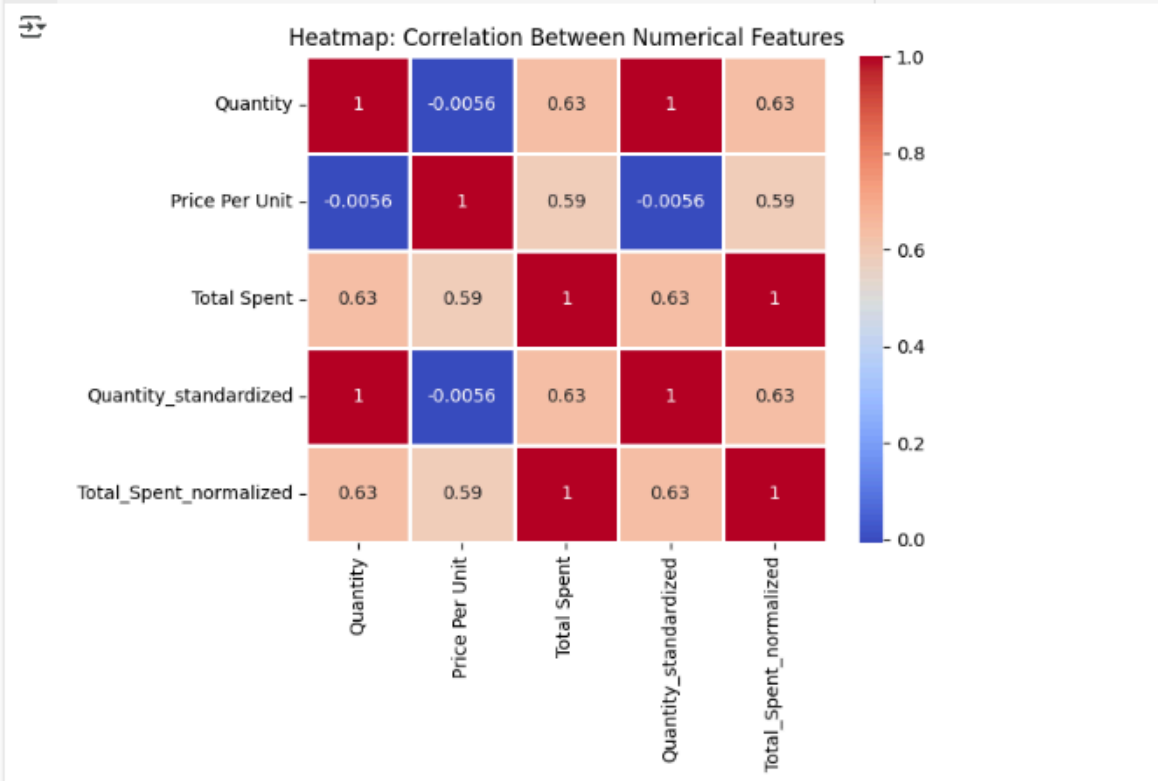
```
sns.boxplot(x=df_subset['Item'], y=df_subset['Total Spent'], palette="Set2")
```



4. Heat map

a heatmap that represents the correlation between numerical features in the dataset. It highlights the strength and direction of relationships between variables such as Quantity, Price Per Unit, and Total Spent. The correlation values range from -1 to 1, where positive values indicate direct relationships, and negative values suggest inverse correlations. The code uses Seaborn's `sns.heatmap()` function with the "coolwarm" colormap, annotating correlation values for better readability. The strong correlation between Quantity and Total Spent (0.63) suggests that higher purchases lead to increased spending. This visualization is useful for identifying dependencies and patterns among numerical features.

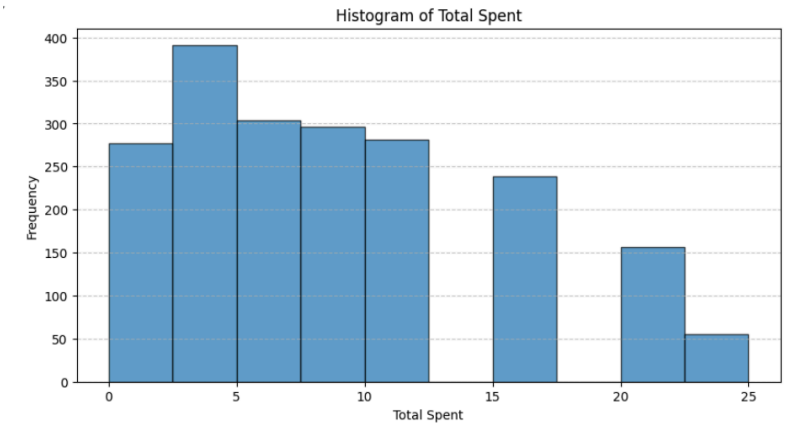
```
sns.heatmap(df_subset.corr(numeric_only=True), annot=True, cmap="coolwarm", linewidths=1)
plt.title("Heatmap: Correlation Between Numerical Features")
plt.show()
```



5. Basic Histogram

histogram created using plt.hist(). The code likely set bins across the range 0-25, with the y-axis showing raw frequency counts. The blue color suggests using something like color='skyblue', and the code included proper axis labels using plt.xlabel() and plt.ylabel(). The gridlines were likely added using plt.grid(linestyle='--', alpha=0.7).

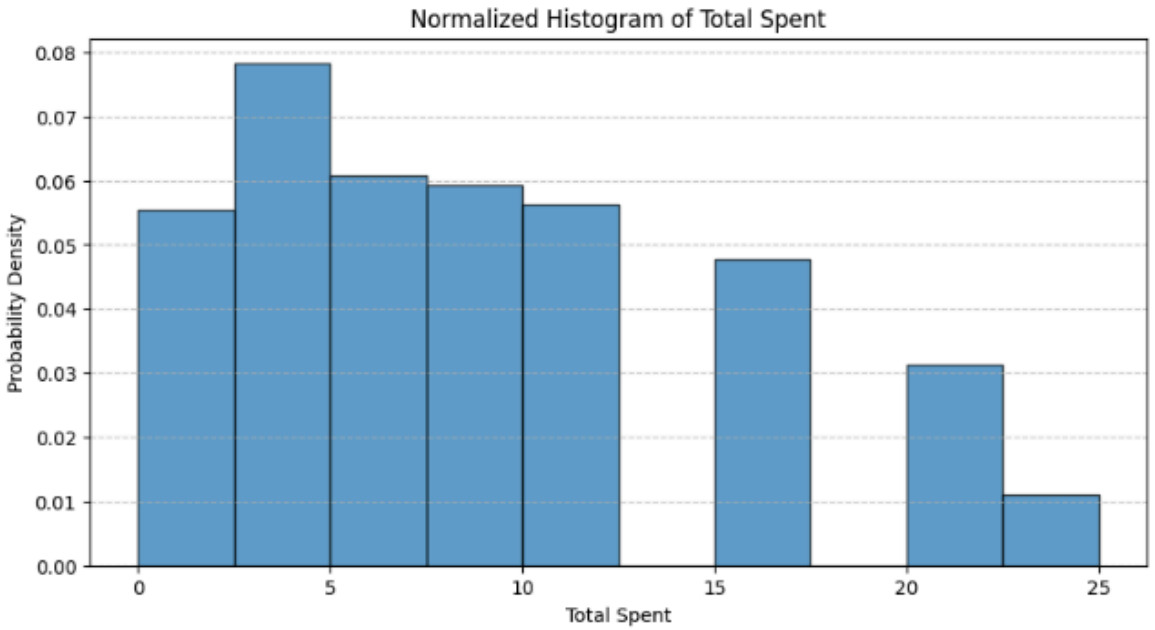
```
column_name = "Total Spent"

# Plot the histogram
plt.figure(figsize=(10, 5))
plt.hist(df[column_name], bins=10, edgecolor='black', alpha=0.7)
plt.xlabel(column_name)
plt.ylabel("Frequency")
plt.title(f"Histogram of {column_name}")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



6. Normalized Histogram

This is similar to the first histogram but uses normalized values (probability density) on the y-axis. The code would be nearly identical but with the addition of the density=True parameter in the plt.hist() function. This normalizes the data so the total area of the histogram equals 1, making it easier to interpret as a probability distribution.

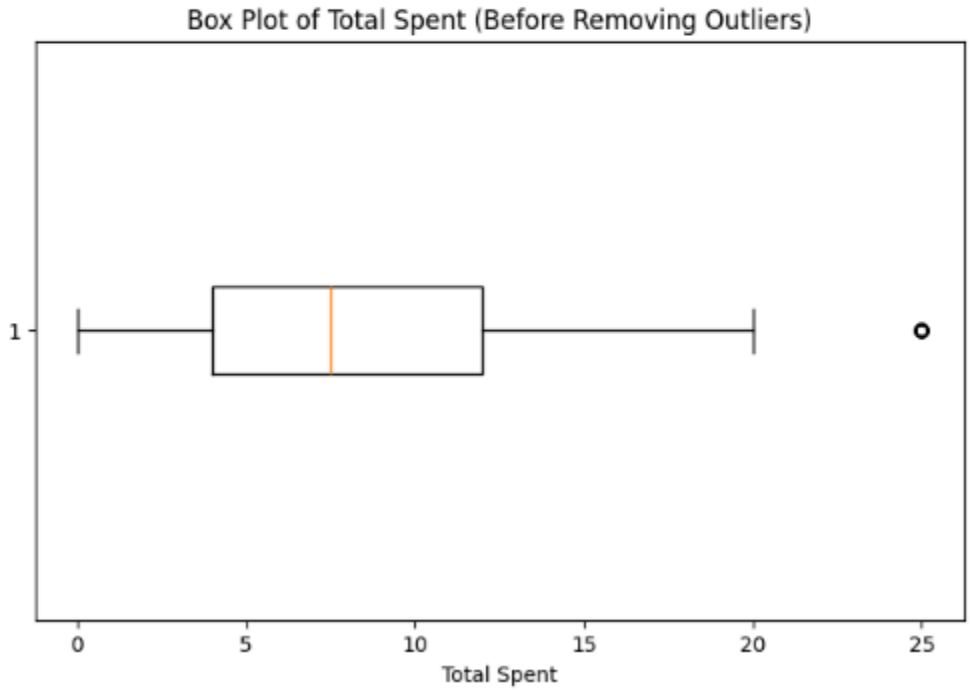
```
plt.figure(figsize=(10, 5))
plt.hist(df[column_name], bins=10, edgecolor='black', alpha=0.7, density=True)
plt.xlabel(column_name)
plt.ylabel("Probability Density")
plt.title(f"Normalized Histogram of {column_name}")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



7. Box plot

This was created using plt.boxplot() or sns.boxplot(). The code shows the distribution's quartiles, with whiskers extending to the data's limits. The single dot at around 25 represents an outlier. The orange line in the box represents the median. The code likely included showfliers=True to display outliers and set the figure orientation to horizontal.

```
column_name = "Total Spent"

# Plot initial Box Plot to visualize outliers
plt.figure(figsize=(8, 5))
plt.boxplot(df[column_name], vert=False)
plt.xlabel(column_name)
plt.title(f"Box Plot of {column_name} (Before Removing Outliers)")
plt.show()
```

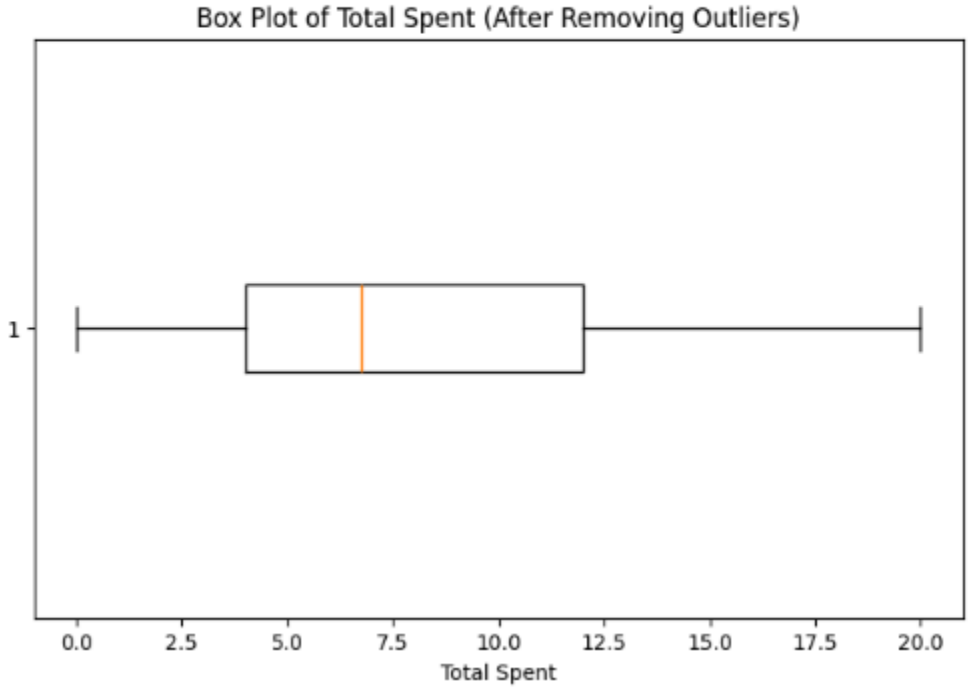


8. Box plot (after removing outlier)

This is identical in code structure to the previous box plot, but the data was first processed to remove outliers. Common outlier removal techniques in Python include using IQR (Interquartile Range) method with something like df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR)))] before creating the plot.

```python
# Calculate IQR (Interquartile Range)
Q1 = df[column_name].quantile(0.25)  # First quartile (25th percentile)
Q3 = df[column_name].quantile(0.75)  # Third quartile (75th percentile)
IQR = Q3 - Q1  # Interquartile Range

# Define lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter the dataset (Remove outliers)
df_filtered = df[(df[column_name] >= lower_bound) & (df[column_name] <= upper_bound)]
```

```python
plt.figure(figsize=(8, 5))
plt.boxplot(df_filtered[column_name], vert=False)
plt.xlabel(column_name)
plt.title(f"Box Plot of {column_name} (After Removing Outliers)")
plt.show()
```



This final graph combines two histograms using either multiple plt.hist() calls or a single call with two data sets. The transparency effect (alpha) was likely set to around 0.5 to make the overlapping visible. The legend was added using plt.legend(), and different colors were specified for each histogram. The code probably used plt.hist() twice with different data sets, once for pre-outlier removal and once for post-outlier removal data.

```
# Compare histograms before and after removing outliers
plt.figure(figsize=(10, 5))
plt.hist(df[column_name], bins=10, alpha=0.5, label="Before Outlier Removal", edgecolor='black')
plt.hist(df_filtered[column_name], bins=10, alpha=0.7, label="After Outlier Removal", edgecolor='black')
plt.xlabel(column_name)
plt.ylabel("Frequency")
plt.title(f"Histogram of {column_name} (Before & After Outlier Removal)")
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

Conclusion:

This experiment helped us understand visualisation of data in the form of bar graphs,box plots,heatmaps and histogram using the matplotlib and seaborn library. We plotted the preprocessed data from experiment1 in the form of charts and graphs to gain better insights from the information.