Experiment 2

Aim: Data Visualization / Exploratory Data Analysis using Matplotlib and Seaborn

Introduction

Exploratory Data Analysis (EDA) is a crucial step in the data analysis pipeline. It involves visually and statistically exploring the data to gain insights and understand its underlying patterns, distributions, and relationships. In this section, we will use Matplotlib and Seaborn, two popular Python libraries, to create visualizations that help in uncovering these insights.

Matplotlib: A comprehensive library for creating static, animated, and interactive visualizations in Python.

Seaborn: Built on top of Matplotlib, Seaborn provides a high-level interface for drawing attractive and informative statistical graphics.

The primary objective of this analysis is to explore and visualize key patterns in the vehicle dataset, focusing on understanding relationships between different attributes and identifying significant trends.

Steps for Data Visualization and EDA

1) **Data Loading and Preprocessing:**

    - Load the dataset into a pandas DataFrame.
    - Perform initial data cleaning (handle missing values, incorrect data types, etc.).
    - Convert any necessary columns into the appropriate data types (e.g., dates, categorical variables).

2) **Summary Statistics:**

    - Use methods like df.describe() and df.info() to understand the basic statistics and data structure.
    - Check for missing values and visualize their distribution.

3) **Univariate Analysis:** This analysis focuses on visualizing individual variables to understand their distribution.

    - Histograms: Use Matplotlib and Seaborn's histplot() or hist() to display the distribution of numerical features like Vehicle Year, Vehicle Weight, and Registration Start Date.

- Box Plots: Use boxplot() from Seaborn to detect outliers in numerical variables such as Vehicle Weight and Vehicle Declared Gross Weight.

- Count Plots: For categorical features like Vehicle Type, Fuel Code, or Vehicle Category, use countplot() from Seaborn to visualize the distribution of each category.

4) **Bivariate Analysis:** This analysis explores the relationship between two variables.

    - Scatter Plots: Use scatterplot() from Seaborn to examine relationships between two continuous variables, such as Vehicle Weight vs. Vehicle Year, or Vehicle Recorded GVWR vs. Vehicle Declared Gross Weight.

    - Correlation Heatmap: Use heatmap() from Seaborn to visualize correlations between numerical variables. This can highlight which attributes have strong positive or negative relationships.

    - Pair Plots: For multiple continuous variables, use Seaborn's pairplot() to show pairwise relationships in the dataset and help identify correlations between them.

5) **Multivariate Analysis:** Analyze interactions between more than two variables at once.

    - Facet Grids: Use FacetGrid() in Seaborn to show how the relationships between two variables differ across the categories of a third variable.

    - Violin Plots: A combination of box plot and kernel density estimate, useful for understanding the distribution and density of a continuous variable against a categorical variable (e.g., Vehicle Type vs. Vehicle Weight).

6) **Categorical Variable Analysis:** Explore categorical variables to find trends or distributions.

    - Bar Plots: Use barplot() to compare the average of a numerical variable across categories, such as comparing the average Vehicle Weight for different Vehicle Types.

- Pie Charts: For categorical features like Vehicle Make or Fuel Code, use Matplotlib's pie() function to create visual representations of the proportions.

7) **Time Series Analysis (if applicable):** If there is any time-related data, such as Registration Start Date, we can visualize trends over time.

    - Line Plots: Use lineplot() from Seaborn to examine trends over time, such as the number of vehicles registered each year or the distribution of Vehicle Year over time.

    - Time-based Aggregation: Group the data by year or month to visualize trends and patterns in registrations, using groupby() in pandas followed by a line plot.

General Syntax in Python for Data Visualization

Python libraries like Matplotlib and Seaborn follow a general syntax for creating visualizations:

1. **Import the library**: Import the required libraries (e.g., `import matplotlib.pyplot as plt`).
2. **Prepare the data**: Use Pandas to manipulate and prepare the data for visualization.
3. **Create the plot**: Use functions like `plot()`, `scatter()`, `boxplot()`, etc., to create the visualization.
4. **Customize the plot**: Add titles, labels, legends, and other customizations.
5. **Display the plot**: Use `plt.show()` to display the visualization.

1) Bar Graph and Contingency Table

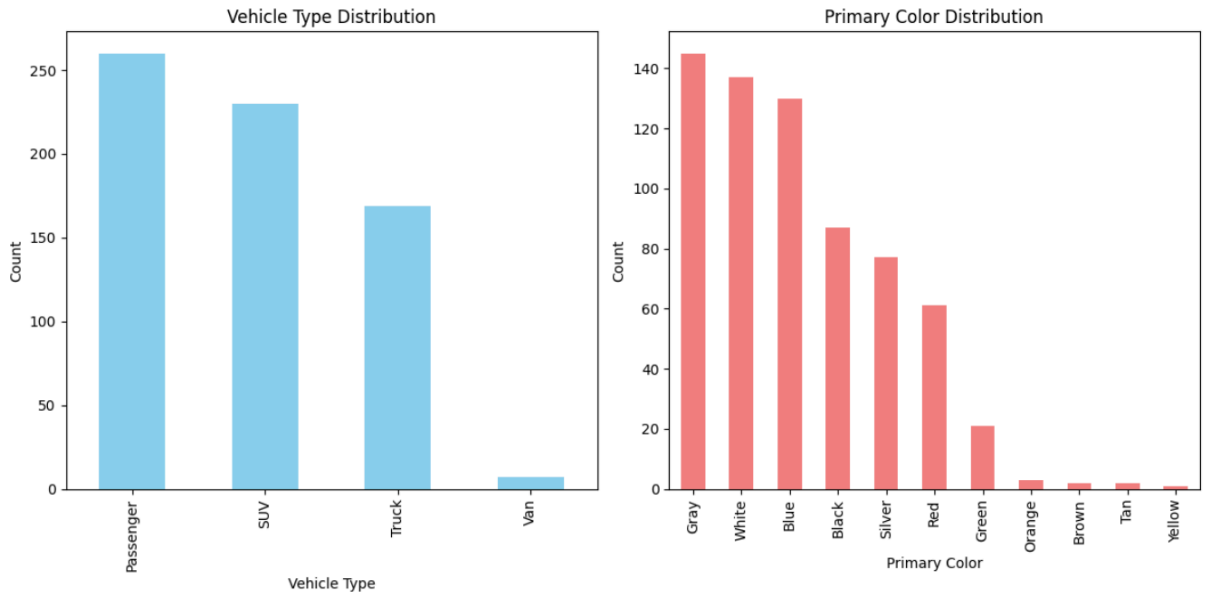
Bar Graph:

A bar graph is a chart that presents data with rectangular bars or columns. Each bar represents a category, and the height or length of the bar corresponds to the value or frequency of that category. Bar graphs are particularly useful for comparing different categories of a categorical variable.

Contingency Table:

A contingency table is a matrix that shows the frequency distribution of variables. It's especially useful for understanding the relationship between two categorical variables.

```
import matplotlib.pyplot as plt

# Count the occurrences of each category in 'Vehicle Type' and 'Primary Color'
vehicle_type_counts = df['Vehicle Type'].value_counts()
color_counts = df['Primary Color'].value_counts()

# Create bar graph
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
vehicle_type_counts.plot(kind='bar', color='skyblue')
plt.title('Vehicle Type Distribution')
plt.xlabel('Vehicle Type')
plt.ylabel('Count')

plt.subplot(1, 2, 2)
color_counts.plot(kind='bar', color='lightcoral')
plt.title('Primary Color Distribution')
plt.xlabel('Primary Color')
plt.ylabel('Count')

plt.tight_layout()
plt.show()
```



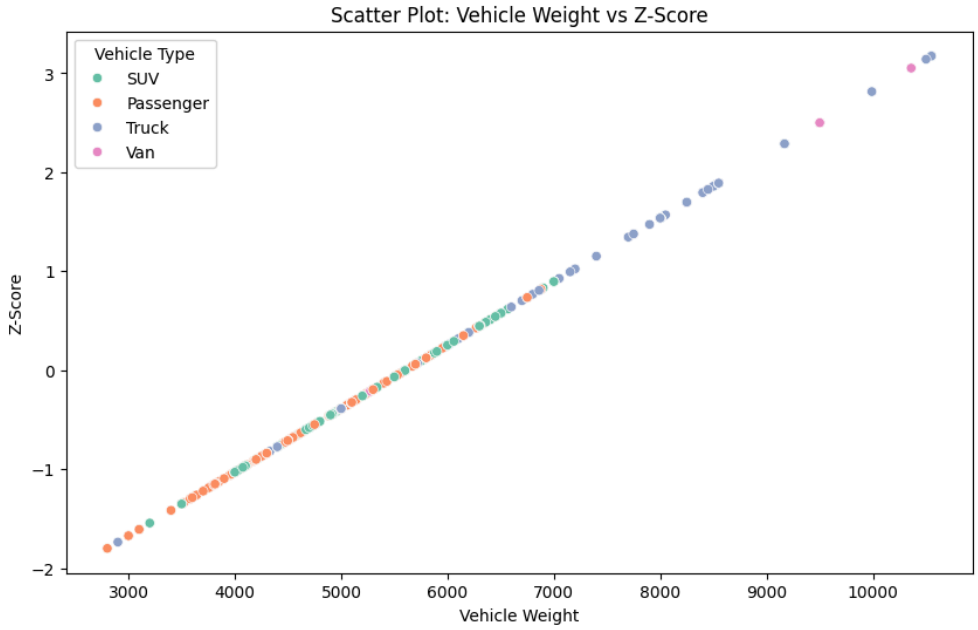
```python
# Create a contingency table for 'Vehicle Type' and 'Primary Color'
contingency_table = pd.crosstab(df['Vehicle Type'], df['Primary Color'])
print(contingency_table)
```

```
Primary Color  Black  Blue  Brown  Gray  Green  Orange  Red  Silver  Tan  \
Vehicle Type
Passenger         38    44      1    62      3       1   32      30    1
SUV               27    51      1    46      4       1   18      24    1
Truck             22    34      0    37     14       0    9      23    0
Van                0     1      0     0      0       1    2       0    0

Primary Color  White  Yellow
Vehicle Type
Passenger         48       0
SUV               57       0
Truck             29       1
Van                3       0
```

2) Scatter Plot

A scatter plot is a type of data visualization used to display values for two continuous variables. It shows how one variable is affected by another, and it's useful for identifying trends, patterns, correlations, or outliers.

```python
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(x='Vehicle Weight', y='Z_Score', data=df, hue='Vehicle Type', palette='Set2')
plt.title('Scatter Plot: Vehicle Weight vs Z-Score')
plt.xlabel('Vehicle Weight')
plt.ylabel('Z-Score')
plt.show()
```

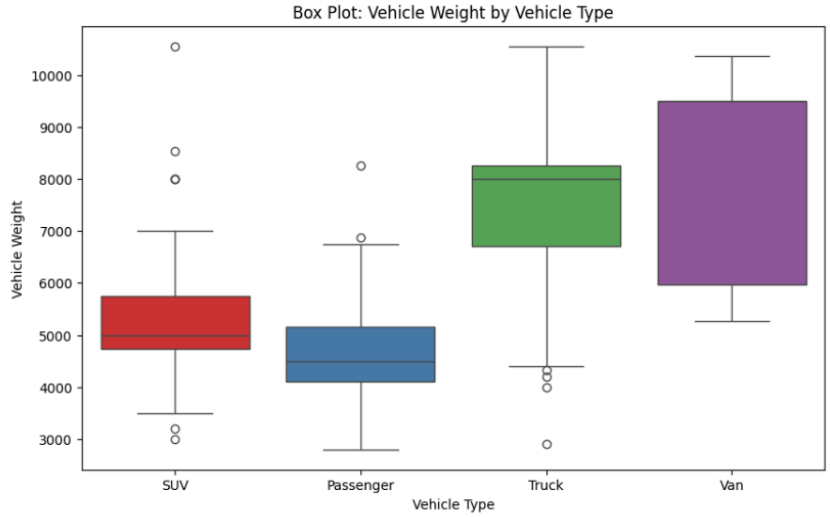


3) Box Plot

A box plot (also known as a box-and-whisker plot) is a graphical representation of the distribution of a dataset. It displays the median, upper and lower quartiles, and any potential outliers. Box plots are useful for comparing distributions across different categories, visualizing spread, and identifying skewness or outliers in the data.

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Vehicle Type', y='Vehicle Weight', data=df, palette='Set1')
plt.title('Box Plot: Vehicle Weight by Vehicle Type')
plt.xlabel('Vehicle Type')
plt.ylabel('Vehicle Weight')
plt.show()
```

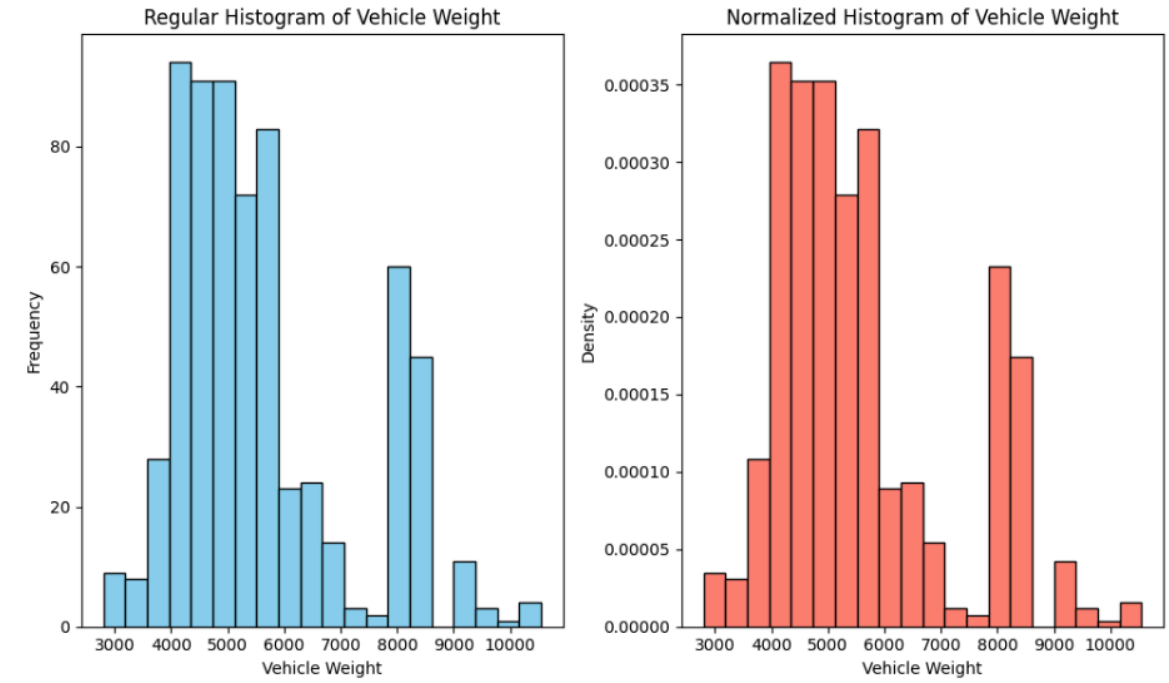
```
<ipython-input-42-dd532d0fa891>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='Vehicle Type', y='Vehicle Weight', data=df, palette='Set1')
```



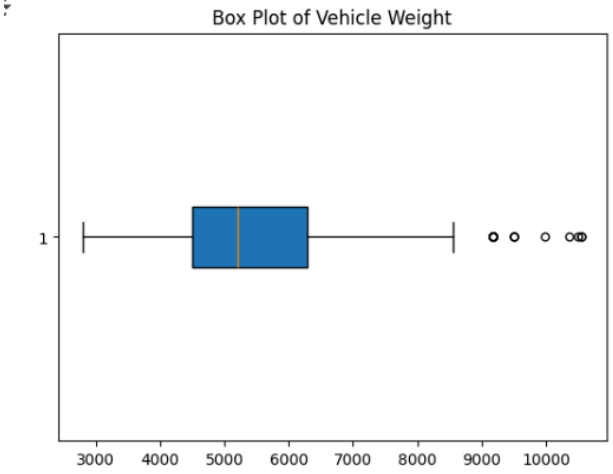
4) Histogram

A histogram is a type of graph that is used to represent the frequency distribution of a dataset. It displays the number of occurrences (or frequency) of data points that fall within certain ranges (called bins). Histograms are particularly useful for understanding the distribution of continuous data, such as the spread, skewness, and presence of outliers.

```
# Plotting the Regular Histogram
plt.figure(figsize=(10, 6))

# Regular Histogram
plt.subplot(1, 2, 1)
plt.hist(df['Vehicle Weight'], bins=20, color='skyblue', edgecolor='black')
plt.title('Regular Histogram of Vehicle Weight')
plt.xlabel('Vehicle Weight')
plt.ylabel('Frequency')

# Plotting the Normalized Histogram
plt.subplot(1, 2, 2)
plt.hist(df['Vehicle Weight'], bins=20, color='salmon', edgecolor='black', density=True)
plt.title('Normalized Histogram of Vehicle Weight')
plt.xlabel('Vehicle Weight')
plt.ylabel('Density')

# Show both histograms
plt.tight_layout()
plt.show()
```

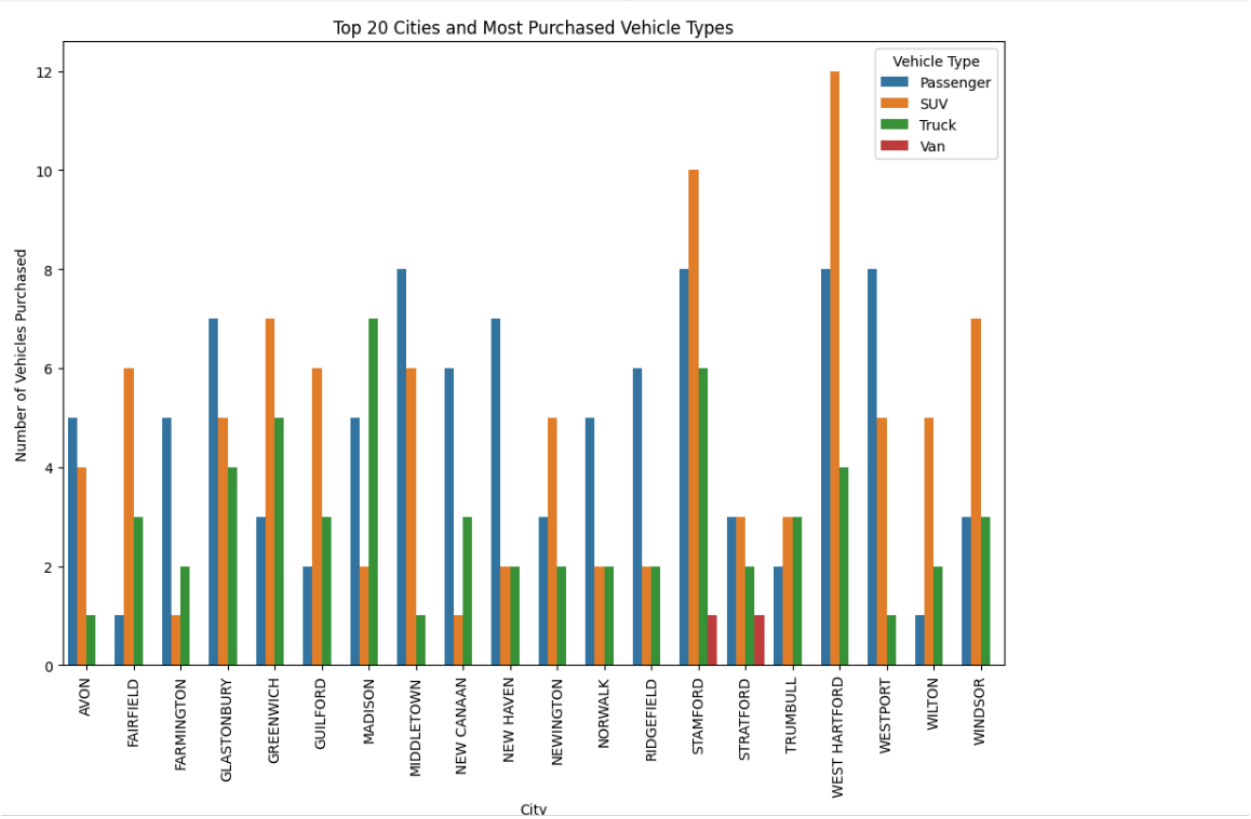


```
]
  import matplotlib.pyplot as plt

  # Plotting the box plot to identify outliers
  plt.boxplot(df['Vehicle Weight'], vert=False, patch_artist=True, flierprops=dict(marker='o', color='red', markersize=5))
  plt.title('Box Plot of Vehicle Weight')
  plt.show()

  # Calculate IQR and identify outliers
  Q1, Q3 = df['Vehicle Weight'].quantile([0.25, 0.75])
  IQR = Q3 - Q1
  lower_bound, upper_bound = Q1 - 1.5 * IQR, Q3 + 1.5 * IQR

  # Filter out outliers
  df_no_outliers = df[(df['Vehicle Weight'] >= lower_bound) & (df['Vehicle Weight'] <= upper_bound)]

  # Display results
  print(f"Original shape: {df.shape}, New shape (no outliers): {df_no_outliers.shape}")
```




```
Original shape: (666, 10), New shape (no outliers): (647, 10)
```

```
# Group by city and vehicle type, then count occurrences
city_vehicle_counts = df.groupby(['Primary Customer City', 'Vehicle Type']).size().reset_index(name='Count')

# Get the top 20 cities based on the number of vehicle purchases
top_20_cities = city_vehicle_counts.groupby('Primary Customer City')['Count'].sum().nlargest(20).index
top_20_data = city_vehicle_counts[city_vehicle_counts['Primary Customer City'].isin(top_20_cities)]

# Plot the data
plt.figure(figsize=(12, 8))
sns.barplot(data=top_20_data, x='Primary Customer City', y='Count', hue='Vehicle Type', dodge=True)
plt.xticks(rotation=90)
plt.title('Top 20 Cities and Most Purchased Vehicle Types')
plt.xlabel('City')
plt.ylabel('Number of Vehicles Purchased')
plt.legend(title='Vehicle Type')
plt.show()
```



Conclusion:

Data visualization is a powerful technique for discovering insights, identifying anomalies, and understanding complex relationships in the dataset. By leveraging Matplotlib and Seaborn, we can gain a deeper understanding of how various vehicle attributes correlate with each other and influence key metrics, such as vehicle registration trends, category distribution, and pricing strategies. The visualizations generated in this EDA can help guide decision-making processes for inventory management, marketing strategies, and customer targeting.