DMBI: EXPERIMENT - 3

Aim:

To perform Exploratory Data Analysis and visualization using python.

Theory:

Exploratory Data Analysis (EDA) is a critical phase in the data science workflow that involves investigating datasets to uncover patterns, spot anomalies, test hypotheses, and check assumptions through visual and statistical methods. Coined by statistician John Tukey in the 1970s, EDA emphasizes an iterative, investigative approach rather than confirmatory analysis. It helps in understanding the underlying structure of the data, identifying missing values, outliers, and relationships between variables, which informs subsequent steps like feature engineering, modeling, or hypothesis testing.

In the context of this experiment, EDA is applied to a car dataset (sourced from Kaggle: "Cars Datasets 2025"), which includes attributes such as company names, fuel types, engine specifications, performance metrics, and prices. The process begins with data loading and cleaning, followed by summary statistics and visualizations to reveal insights like brand popularity, performance distributions, and correlations. Python serves as the primary tool due to its rich ecosystem for data manipulation and visualization, enabling efficient handling of structured data like CSV files.

Data visualization is the graphical representation of information and data using visual elements like charts, graphs, and maps. It transforms complex datasets into intuitive, interpretable formats that highlight trends, patterns, and outliers more effectively than raw numbers or tables. According to visualization expert Edward Tufte, effective visualizations minimize "chart junk" (unnecessary elements) while maximizing the data-ink ratio—the proportion of ink used to present actual data.

The code leverages several Python libraries, each serving a specific purpose in data handling, cleaning, and visualization:

- Pandas: Handles data manipulation (DataFrames, read_csv, describe, value_counts).
- NumPy: Supports numerical operations and NaN handling.
- Matplotlib.pyplot: Provides plotting functionality (figures, labels, show).
- Seaborn: Simplifies statistical visualizations (histograms, bar plots, heatmaps).
- Re: Extracts numbers from strings via regular expressions.

Types of Graphs Used:

- Histogram – Represents frequency distribution of continuous variables by dividing data into bins. In this experiment, histograms were plotted for features like engine size, horsepower, speed, and seats to reveal distributions and spot outliers. Useful for univariate analysis and detecting skewness.
- Bar Graph – Displays categorical comparisons using bars proportional to values. Here, bar plots showed top brands by count and average torque of top brands, making it easy to compare rankings and differences across categories.
- Box Plot – Summarizes data with quartiles, medians, and outliers. Used to compare acceleration times (0–100 km/h) across fuel types, highlighting spreads, variability, and performance differences between categories.
- Heatmap – Visualizes correlations in a matrix with colors indicating strength of relationships. The correlation heatmap revealed strong links, e.g., between horsepower and torque, making it compact for multivariate analysis.

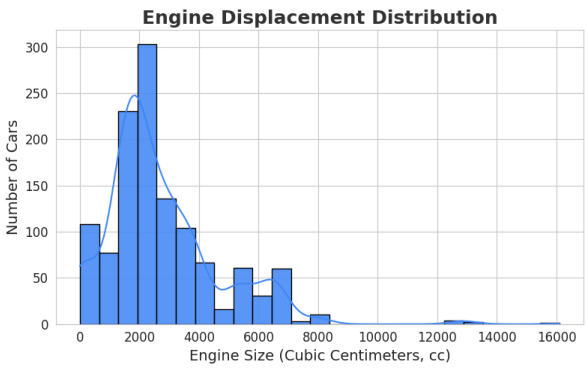
- Scatter Plot – Plots two continuous variables to explore relationships. Horsepower vs Price (log scale) with fuel type as hue revealed clusters, correlations, and outliers, making patterns easier to detect.
- Swarm Plot – Positions data points by categories without overlap, preserving visibility. Applied to engine size vs seats (sample data), it showed how engine capacity varies with seating, useful for small-to-medium datasets.
- Pie Chart – Represents part-to-whole proportions as circular sectors. The top 10 brands plus an "Others" category were plotted, giving a clear picture of brand representation in the dataset.

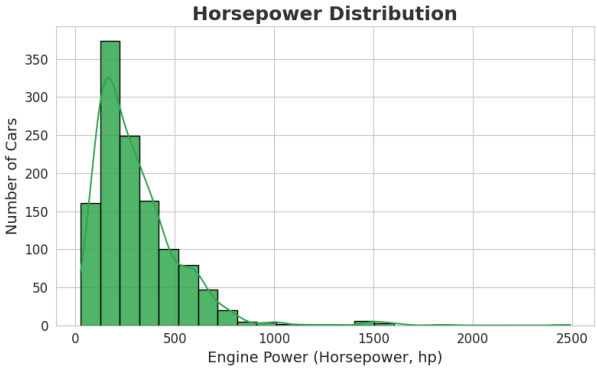
Conclusion:

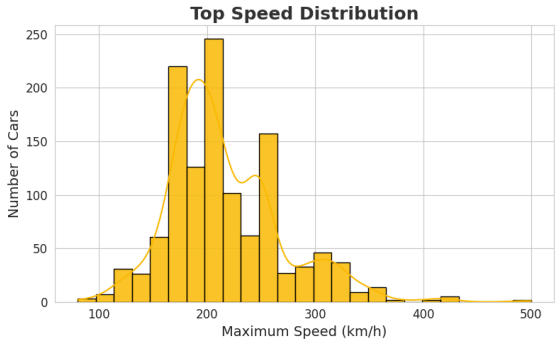
In this experiment, Exploratory Data Analysis and visualization proved highly effective in understanding the car dataset. By applying techniques like histograms, bar charts, box plots, scatter plots, and heatmaps, we were able to identify trends, brand popularity, performance variations, and correlations between attributes such as horsepower, torque, and price. The combination of Python libraries made the process efficient and clear. Overall, EDA not only simplified complex data but also provided meaningful insights that can guide further analysis and decision-making.

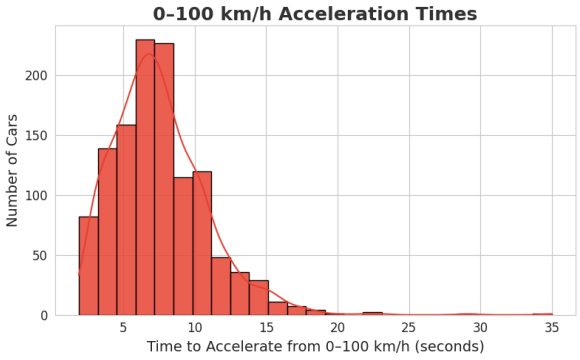
CODE & OUTPUT

```
[ ]  # HISTOGRAM
     hist_colors = ['#4287f5', '#34a853', '#fbbc05', '#ea4335', '#a142f4']


     histogram_features = [
         {
             "col": "Engine_CC",
             "title": "Engine Displacement Distribution",
             "xlabel": "Engine Size (Cubic Centimeters, cc)"
         },
         {
             "col": "HorsePower",
             "title": "Horsepower Distribution",
             "xlabel": "Engine Power (Horsepower, hp)"
         },
         {
             "col": "Max_Speed_kmh",
             "title": "Top Speed Distribution",
             "xlabel": "Maximum Speed (km/h)"
         },
         {
             "col": "Zero_100_Sec",
             "title": "0–100 km/h Acceleration Times",
             "xlabel": "Time to Accelerate from 0–100 km/h (seconds)"
         },
         {
             "col": "Seats",
             "title": "Distribution of Seating Capacity",
             "xlabel": "Number of Seats"
         }
     ]

     sns.set_style("whitegrid")

     for idx, feat in enumerate(histogram_features):
         plt.figure(figsize=(8, 5))
         color = hist_colors[idx % len(hist_colors)]
         sns.histplot(df_clean[feat["col"]], bins=25, kde=True, edgecolor='black', alpha=0.85, color=color)
         plt.title(feat["title"], fontsize=18, fontweight='bold', color='#333333')
         plt.xlabel(feat["xlabel"], fontsize=14)
         plt.ylabel('Number of Cars', fontsize=14)
         plt.xticks(fontsize=12)
         plt.yticks(fontsize=12)
         plt.tight_layout()
         plt.show()
         print()
```

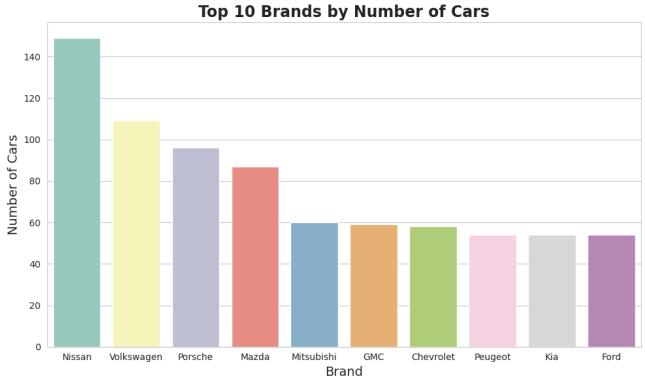


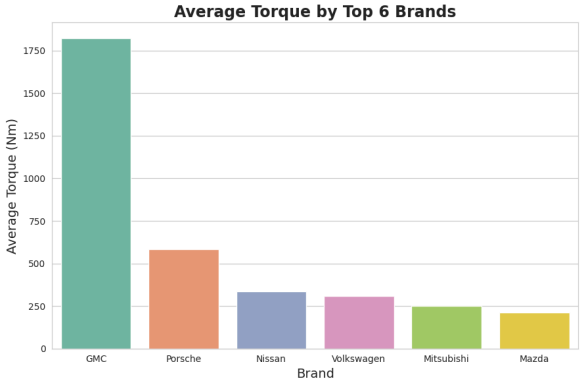




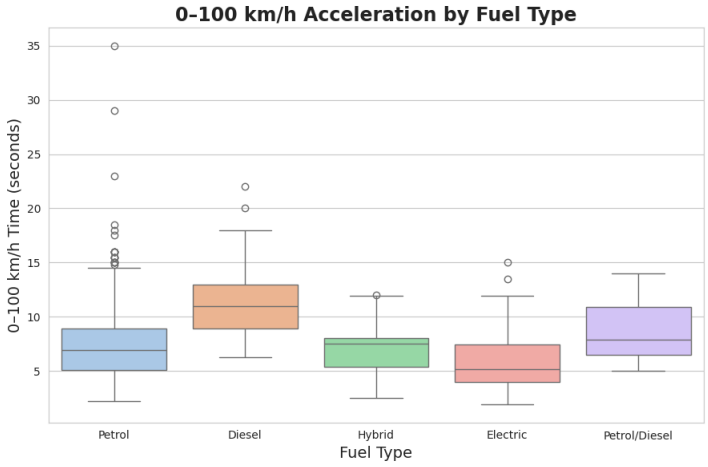


```
[ ]  # BAR GRPAHS
     top_brands = df_clean['Company Names'].value_counts().head(10)
     plt.figure(figsize=(10,6))
     sns.barplot(x=top_brands.index, y=top_brands.values, palette='Set3')
     plt.title('Top 10 Brands by Number of Cars', fontsize=17, fontweight='bold')
     plt.xlabel('Brand', fontsize=14)
     plt.ylabel('Number of Cars', fontsize=14)
     plt.tight_layout()
     plt.show()

     print()

     top6brands = df_clean['Company Names'].value_counts().head(6).index

     avg_torque = df_clean[df_clean['Company Names'].isin(top6brands)].groupby('Company Names')['Torque_Nm'].mean().sort_values(ascending=False)

     plt.figure(figsize=(9,6))
     sns.barplot(x=avg_torque.index, y=avg_torque.values, palette='Set2')
     plt.title('Average Torque by Top 6 Brands', fontsize=17, fontweight='bold')
     plt.xlabel('Brand', fontsize=14)
     plt.ylabel('Average Torque (Nm)', fontsize=14)
     plt.tight_layout()
     plt.show()
```

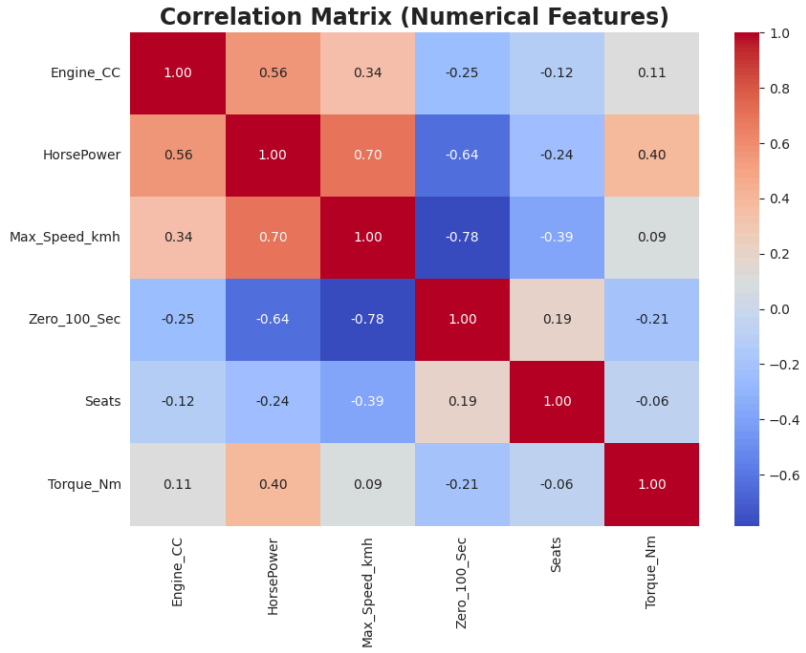




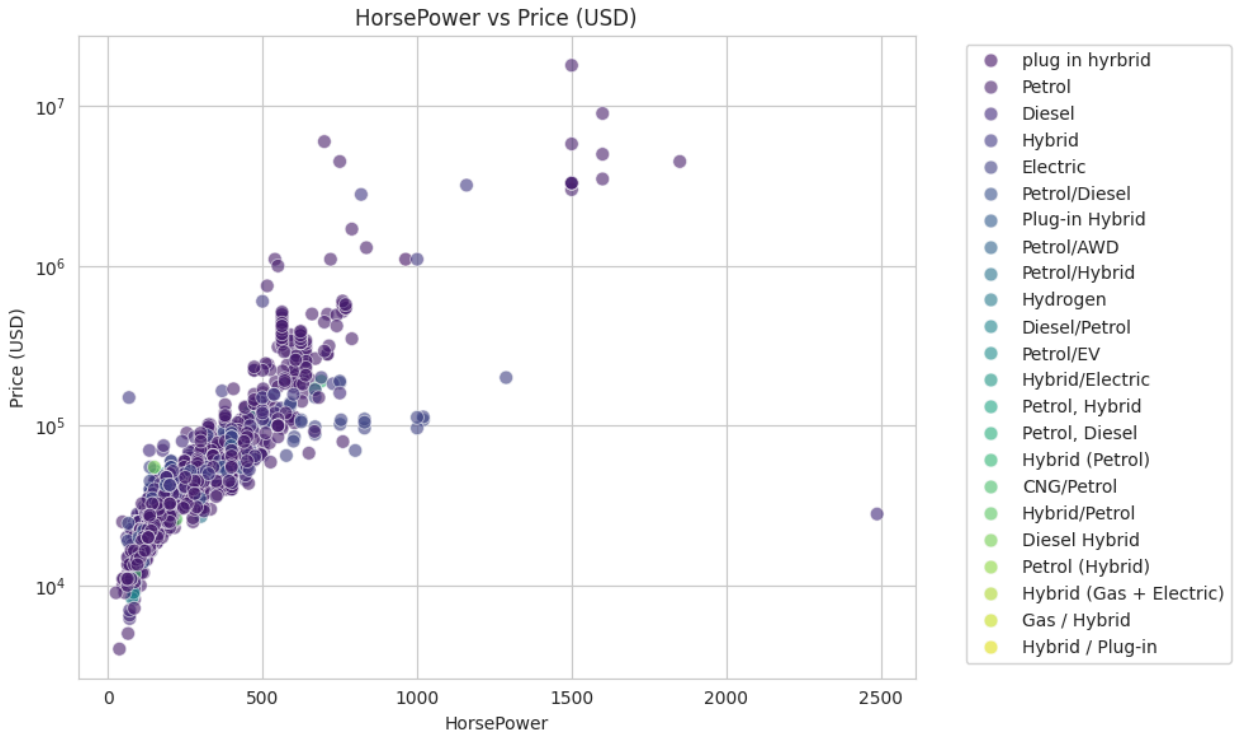

```
[ ]  # BOX PLOT
     top_fuels = df_clean['Fuel Types'].value_counts().head(5).index
     plt.figure(figsize=(9,6))
     sns.boxplot(x='Fuel Types', y='Zero_100_Sec', data=df_clean[df_clean['Fuel Types'].isin(top_fuels)], palette='pastel')
     plt.title('0-100 km/h Acceleration by Fuel Type', fontsize=17, fontweight='bold')
     plt.xlabel('Fuel Type', fontsize=14)
     plt.ylabel('0-100 km/h Time (seconds)', fontsize=14)
     plt.tight_layout()
     plt.show()
```



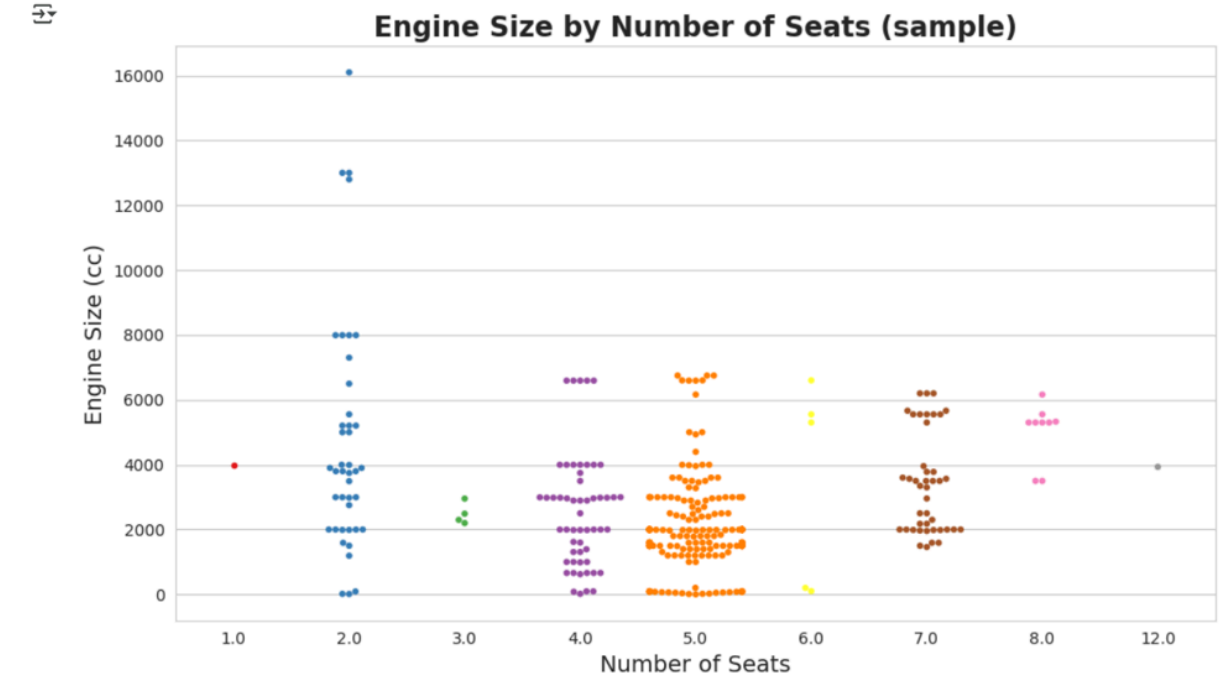
```
[ ]  # HEAT MAP
     plt.figure(figsize=(9,7))
     corr = df_clean[['Engine_CC', 'HorsePower', 'Max_Speed_kmh', 'Zero_100_Sec', 'Seats', 'Torque_Nm']].corr()
     sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
     plt.title('Correlation Matrix (Numerical Features)', fontsize=17, fontweight='bold')
     plt.tight_layout()
     plt.show()
```




```
[ ]  # SCATTER PLOT: HorsePower vs Price_USD
     plt.figure(figsize=(10, 6))
     sns.scatterplot(
         data=df_clean,
         x='HorsePower',
         y='Price_USD',
         alpha=0.6,
         hue='Fuel Types',  # Color by fuel type
         palette='viridis',
         s=60  # Marker size
     )
     plt.title('HorsePower vs Price (USD)')
     plt.xlabel('HorsePower')
     plt.ylabel('Price (USD)')
     plt.yscale('log')  # Log scale for better visualization of price distribution
     plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
     plt.tight_layout()
     plt.show()
```




```
[ ]  # SWARM PLOT
     sample = df_clean.dropna(subset=['Seats','Engine_CC']).sample(n=350, random_state=2)
     plt.figure(figsize=(10,6))
     sns.swarmplot(x='Seats', y='Engine_CC', data=sample, palette='Set1', size=4)
     plt.title('Engine Size by Number of Seats (sample)', fontsize=17, fontweight='bold')
     plt.xlabel('Number of Seats', fontsize=14)
     plt.ylabel('Engine Size (cc)', fontsize=14)
     plt.tight_layout()
     plt.show()
```



```python
# PIE CHART
brand_counts = df['Company Names'].value_counts()
top_brands = brand_counts.head(10)
others_count = brand_counts[10:].sum()

# Combine into a single series for the pie chart
pie_data = top_brands.copy()
if others_count > 0:
    pie_data['Others'] = others_count

# Create the pie chart
plt.figure(figsize=(10, 8))
colors = plt.cm.Set3(np.linspace(0, 1, len(pie_data)))
wedges, texts, autotexts = plt.pie(
    pie_data.values,
    labels=pie_data.index,
    autopct='%1.1f%%',
    startangle=90,
    colors=colors
)

# Improve readability
plt.title('Distribution of Car Brands', fontsize=16, fontweight='bold')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle

# Adjust text properties
for autotext in autotexts:
    autotext.set_color('black')
    autotext.set_fontweight('bold')

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