Name Of School: The George Washington University
Author: Aneri Patel
Instructor: Reza Jafari
Date: 26 April 2024
Course: DATS 6401 Visualization Of Complex Data

Table of Contents:

1. Abstract 6
2. Introduction 6
  2.1 Project Background 6
  2.2 Objectives of the Project 7
3. Description of the Dataset 8
4. Pre-processing Dataset 9
  4.1 Objective 9
  4.2 Feature Engineering 9
  4.3 Data Transformation for PCA 9
5. Outlier Detection & Removal 11
  5.1 Implementation 11
  5.2 Observations 12
6. Principal Component Analysis (PCA) 13
  6.1 Methods and Results 13
  6.2 Observations 14
7. Normality Test 15
  7.1 Methods and Results 15
  7.2 Observations 16
8. Heatmap & Pearson Correlation Coefficient Matrix 22
  8.1 Heatmap Observations 22
  8.2 Scatter Plot Matrix Observations 24
9. Statistics 24
10. Data Analysis and Visualization Insights 36

11. Dashboard 61
11.1 Home Page 61
11.2 Market Trend Analysis 62
11.3 Model Performance Analysis 63
11.4 Car Industry Statistical Analysis 64
12. Conclusion 65
12.1 Learnings from Various Created Graphs 65
12.2 Utility of Python Dashboard for Users 65
12.3 User-friendliness of the Created App 65
12.4 Functionality of the Created App 66
13. Appendix 67
13.1 Static Plots Code..... 67
13.2 Interactive Dashboard Code 93
14. References 125

Table of Figures and Tables:

No.	Type of Plot	Title	Page No.
1	Line Plot	Average Mileage Across Top 10 Years by Sales Volume	36
2	Stacked Bar Plot	Average Price by Manufacturer Country and Fuel Type	37
3	Grouped Bar Plot	Average Car Price by Transmission Type Across Manufacturer Countries	38
4	Count Plot	Number of Cars by Manufacturer	39
5	Pie Chart	Distribution of Cars by Manufacturer Country	40
6	Distribution Plot	Mileage Distribution by Transmission Type	41
7	Pair Plot	Numerical Columns Pairplot	42
8	Heatmap	Count of Cars by Transmission and Fuel Type	43
9	Histogram with KDE	Distribution of Prices for Cars	44
10	QQ-Plot	Normal QQ-Plot of Car Miles Per Gallon	45
11	KDE Plot	Price Distribution by Category	46
12	Im or Regression Plot	Comparative Price vs. Mileage across Different Countries	47
13	Multivariate Box Plot	Mileage Distribution by Manufacturer	48
14	Boxen Plot	Price Distribution by Fuel Type	49
15	Area Plot	Comparative Analysis of Average Car Tax by Category Across Manufacturers	50
16	Violin Plot	Mileage Distribution by Fuel Type	51
17	Joint Plot with KDE and Scatter	Price vs. Tax: KDE and Scatter Representation	52
18	Rug Plot	Mileage Per Gallon Distribution of Vehicles	53
19	3D and Contour Plot	Price vs. Transformed Engine Size	54

20	Cluster Map	Correlation Matrix Cluster Map	55
21	Hexbin Plot	Engine Size vs Tax	56
22	Strip Plot	Tax Distribution of Top 6 Years with Highest Car Taxes	57
23	Swarm Plot	Mileage Distribution by Fuel Type for Top 4 Manufacturers	58
24	Table	Information of dataset	59
25	3D Contour Plot	3D Contour Plot Price v/s Mileage	60

1. Abstract :

This project aims to explore the used car market by analyzing a comprehensive dataset featuring variables such as selling price, mileage, and country of origin. We developed both static plots and an interactive dashboard using advanced data visualization techniques and the Dash framework to provide insightful analytics on used car sales trends. Key visualizations include price distributions, mileage analysis, and geographic sales patterns, enabling users to effectively discern market dynamics. The dashboard and static visualizations were successfully deployed on Google Cloud Platform (GCP), enhancing accessibility and user interaction. This tool aids potential buyers and sellers in making informed decisions and serves as a model for market analysis projects in various sectors.

2. Introduction:

The used car market is an important part of the automotive industry. It provides a cheaper option for buyers and shows how the economy is doing. Used cars play a big role in shaping market trends and how people behave when buying cars.

2.1 Project Background:

The used car market is a crucial sector of the automotive industry, offering valuable insights into economic trends and consumer behaviors. As new car prices continue to escalate, a growing number of consumers are opting for used vehicles, making this market a significant area of economic activity. This trend is not only a reflection of consumer preference for more cost-effective solutions but also of the evolving landscape of automotive ownership, where sustainability and economic viability play increasingly important roles.

In recent years, the proliferation of digital platforms has dramatically reshaped the used car market. These platforms have improved market transparency and operational efficiency by facilitating easier access to vehicle history, valuation data, and a broader national market reach. This digital shift is particularly relevant as it intersects with a rising consumer demand for more sustainable and fuel-efficient vehicles. Despite these advancements, the market still faces challenges such as price variability, the reliability of vehicle condition assessments, and the integration of these platforms with traditional sales channels.

Studying the used car market is essential for understanding these dynamics and their implications on both macroeconomic scales and individual consumer levels. Insights derived from this sector can help stakeholders—ranging from policymakers to private enterprises—develop strategies

that align with shifting economic conditions and consumer expectations. By delving into the intricacies of this market, this project aims to harness advanced data visualization and analytics to unearth patterns that could lead to more informed decisions and strategic adjustments in the automotive industry.

2.2 Objectives of the Project:

The primary aim of this project is to enhance the decision-making process for potential used car buyers by providing comprehensive and intuitive data visualizations and an interactive dashboard. The objectives are detailed as follows:

2.2.1 Development of Static Plots:

To create a series of static plots that analyze various aspects of the used car market. These visualizations focus on key metrics such as price distributions, mileage, and comparisons of vehicle conditions across different manufacturers. The goal is to offer users straightforward, digestible insights that can inform their understanding of market trends and pricing structures.

2.2.2 Interactive Dashboard Creation:

To develop an interactive dashboard that serves as a user-friendly tool for potential car buyers. The dashboard is designed to assist users in determining the type of car that best suits their needs based on a range of selectable criteria, including price, mileage, fuel efficiency, and other relevant parameters. This tool aims to simplify the data analysis process, making it accessible even to those with no technical background in data analytics.

2.2.3 Deployment on Google Cloud Platform (GCP):

To deploy the interactive dashboard on Google Cloud Platform, ensuring reliable access and scalability. Hosting the dashboard on GCP allows for high availability and easy accessibility for users from various geographic locations, enhancing user experience and engagement.

2.2.4 User Empowerment through Data:

To empower users by providing them with the tools and information necessary to make well-informed decisions. By democratizing access to data analytics through user-friendly interfaces, the project aims to reduce the complexity and opacity that often characterizes the used car market.

3. Description of the Dataset

This dataset encompasses an extensive collection of data from over 90,000 used cars, spanning a period from 1970 to 2024. It provides a detailed view of the automotive market across five decades, making it an essential resource for analyzing trends in car features, pricing, and consumer preferences.

Dataset Features:

- **Model:** Specifies the model name of the car. This attribute helps in identifying the design and functional characteristics specific to each model.
- **Year:** Indicates the year the car was manufactured. It is crucial for assessing the age of the vehicle and its relevance to contemporary automotive standards.
- **Price:** Represents the selling price of the car. This is a key variable for economic analysis and understanding market valuation of used cars.
- **Transmission:** Describes the type of transmission the car uses, such as manual or automatic. This factor can significantly influence a car's desirability and price.
- **Mileage:** The total miles the car has been driven. Higher mileage can affect a car's condition and resale value.
- **Fuel Type:** Identifies what type of fuel the car uses (e.g., gasoline, diesel, electric). This information is important for environmental and economic studies.
- Tax: Lists the annual tax rate applicable to the car, which impacts the total cost of ownership.
- **MPG:** Stands for miles per gallon, measuring the car’s fuel efficiency. This is particularly valuable for consumers concerned with fuel economy.
- **Engine Size:** The volume of the car's engine, usually in liters, which affects performance and fuel consumption.
- **Manufacturer:** The company that produced the car. This attribute allows analysis by brand, comparing market strategies and consumer loyalty.

Relevance of the Dataset:

With its comprehensive scope and detailed attributes, this dataset offers a robust foundation for analyzing various aspects of the automotive industry. It enables the exploration of how different factors such as model, engine size, and fuel type influence a car's market price and consumer demand. This dataset is particularly useful for tracking how technological advancements and economic factors have shaped vehicle sales and preferences over time.

4. Pre-processing dataset:

4.1 Objective:

The preprocessing phase aimed to enhance the dataset by adding new features that could provide additional insights during the analysis. Although the dataset was initially clean with no missing values, several steps were taken to refine the dataset further and prepare it for Principal Component Analysis (PCA) and other analytical methods.

4.2 Feature Engineering:

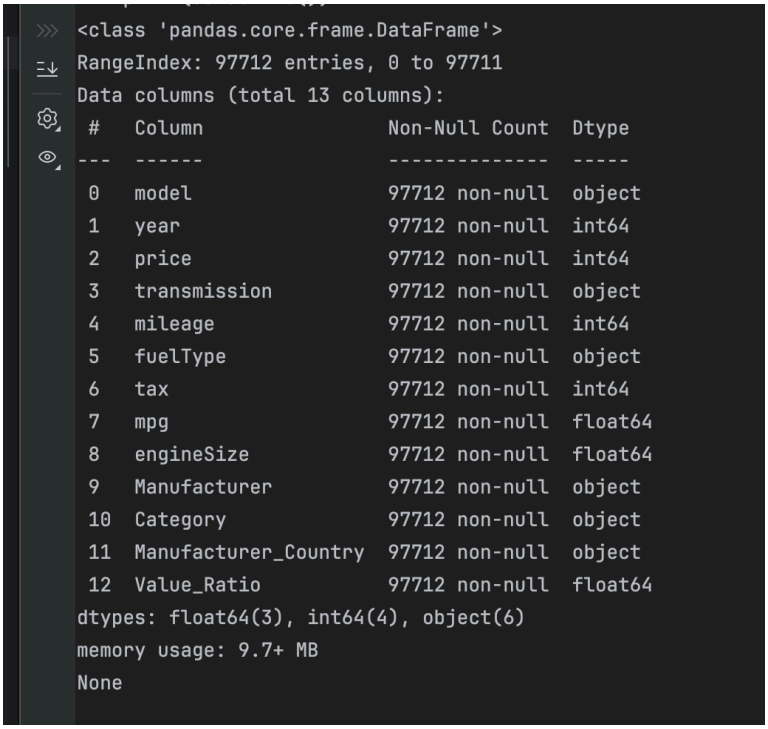
1. **Manufacturer Country:**
    - Created a new column, 'Manufacturer Country,' to identify the country of origin for each car manufacturer. This involved the creation of a dictionary mapping each manufacturer to its respective country. The dictionary was then applied to the dataset to populate the new column. This feature helps in understanding geographical trends in car manufacturing and sales.
2. **Category Based on Price:**
    - Introduced a new column, 'Category,' to classify cars based on their price into different ranges: Low-range, Mid-range, Luxury, Exotic, and Premium. This categorization was determined by predefined price thresholds which grouped cars into segments reflecting their market positioning and consumer perception. This feature is particularly useful for segment-specific analysis in the used car market.
3. **Value Ratio:**
    - Calculated a new feature 'Value Ratio' by dividing the price of the car by its mileage. This ratio provides a normalized measure of cost-effectiveness, offering a direct comparison of price per mile among cars, which can be an indicator of value for money in the used car market.

4.3 Data Transformation for PCA:

- To facilitate PCA, which requires numerical input for its calculations, label encoding was applied to categorical variables. This transformation converted text data in columns like 'Transmission' and 'Fuel Type' into numerical codes, enabling their inclusion in the PCA analysis.

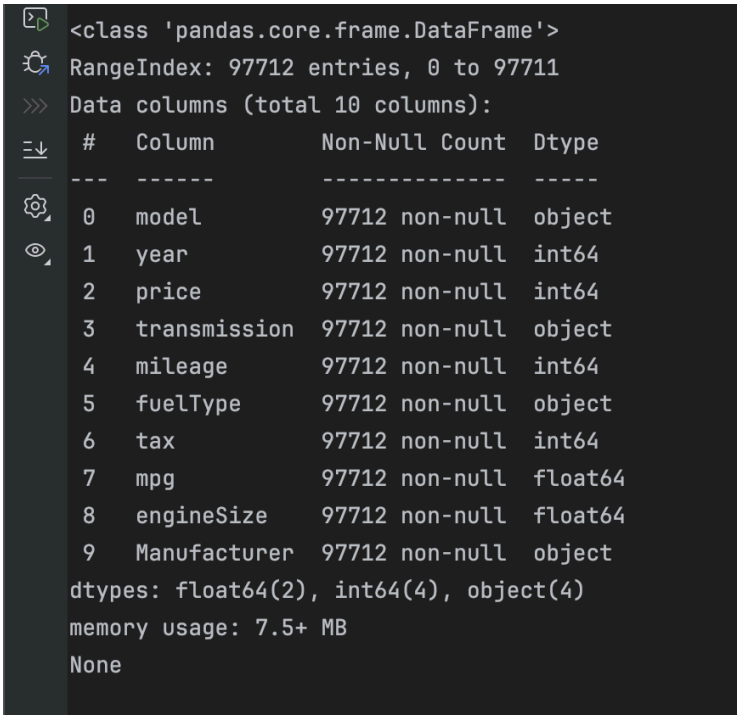
Below is a screenshot of the data information before and after the pre-processing of data:

Before Data Pre-processing:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97712 entries, 0 to 97711
Data columns (total 13 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   model                97712 non-null  object
 1   year                 97712 non-null  int64
 2   price                97712 non-null  int64
 3   transmission         97712 non-null  object
 4   mileage              97712 non-null  int64
 5   fuelType             97712 non-null  object
 6   tax                  97712 non-null  int64
 7   mpg                  97712 non-null  float64
 8   engineSize           97712 non-null  float64
 9   Manufacturer         97712 non-null  object
 10  Category             97712 non-null  object
 11  Manufacturer_Country  97712 non-null  object
 12  Value_Ratio          97712 non-null  float64
dtypes: float64(3), int64(4), object(6)
memory usage: 9.7+ MB
None
```

After Data Pre-Processing:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97712 entries, 0 to 97711
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   model         97712 non-null  object
 1   year          97712 non-null  int64
 2   price         97712 non-null  int64
 3   transmission  97712 non-null  object
 4   mileage       97712 non-null  int64
 5   fuelType      97712 non-null  object
 6   tax           97712 non-null  int64
 7   mpg           97712 non-null  float64
 8   engineSize    97712 non-null  float64
 9   Manufacturer  97712 non-null  object
dtypes: float64(2), int64(4), object(4)
memory usage: 7.5+ MB
None
```

5. Outlier Detection & Removal:

For outlier detection, the Interquartile Range (IQR) method was employed. This statistical technique is effective for identifying outliers by focusing on the distribution's quartiles. The IQR is calculated as the difference between the 75th percentile (Q3) and the 25th percentile (Q1) of the data. Outliers are defined as observations that fall below Q1 - 1.5*IQR or above Q3 + 1.5*IQR. This method is robust against non-normal distributions and is widely used due to its simplicity and efficacy.

5.1 Implementation:

Using the remove_outliers_iqr function, that is created to detect and remove outliers. Outliers were identified and removed across several key numerical attributes of the dataset. Below is the range less than or greater than that values would be considered outliers :

- **Price:**
  - Outliers were identified as values below $6,127.50 and above $36,876.50.
  - This range ensures that extremely low-priced or overly expensive vehicles, which could distort average pricing trends, were excluded from analysis.
- **Mileage:**
  - Values below 28,680.25 miles and above 70,225.75 miles were considered outliers.
  - Removing these outliers helps to avoid skewed analysis due to unusually high or low usage vehicles, which are not typical of average market conditions.
- **Tax:**
  - Outliers in tax were values below $95.00 and above $175.00.
  - This filtering helps in focusing on vehicles within the typical tax bracket, excluding those with anomalous tax rates.
- **MPG (Miles Per Gallon):**
  - Values below 27.40 MPG and above 77.80 MPG were removed.
  - This step ensures the exclusion of extreme fuel efficiency values, which might represent special cases like hybrid or poorly maintained vehicles.
- **Engine Size:**
  - Outliers were considered as engine sizes below 0.0 liters and above 3.2 liters.
  - This criterion excludes unusually small or large engines, focusing analysis on typical consumer vehicles.
- **Value Ratio:**
  - Values below 4.00 and above 8.19 were identified as outliers.

- This metric, representing the price per mile, was cleansed of extreme values to provide a more accurate depiction of car value relative to its usage.

5.2 Observations:

- **Price and Mileage:** The defined thresholds for outliers in price and mileage were crucial for removing extreme values that could skew the analysis. This adjustment was particularly important for ensuring that the dataset accurately reflects the general market without extreme premium or poorly conditioned vehicles.
- **Tax:** The detection of outliers in the tax category revealed a uniform value across the dataset, suggesting a potential issue with data collection or entry. Further investigation or confirmation with data providers might be necessary.
- **MPG and Engine Size:** The boundaries set for MPG and engine size helped in filtering out unusually efficient or large-engine vehicles, which are less common and could misrepresent the typical consumer vehicle.

Below is a screenshot that will give information about dataset after removing outliers:

```
<class 'pandas.core.frame.DataFrame'>
Index: 55413 entries, 0 to 97711
Data columns (total 13 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   model                55413 non-null  object 
 1   year                 55413 non-null  int64  
 2   price                55413 non-null  int64  
 3   transmission         55413 non-null  object 
 4   mileage              55413 non-null  int64  
 5   fuelType             55413 non-null  object 
 6   tax                  55413 non-null  int64  
 7   mpg                  55413 non-null  float64
 8   engineSize           55413 non-null  float64
 9   Manufacturer         55413 non-null  object 
 10  Category             55413 non-null  object 
 11  Manufacturer_Country  55413 non-null  object 
 12  Value_Ratio          55413 non-null  float64
dtypes: float64(3), int64(4), object(6)
memory usage: 5.9+ MB
None
```

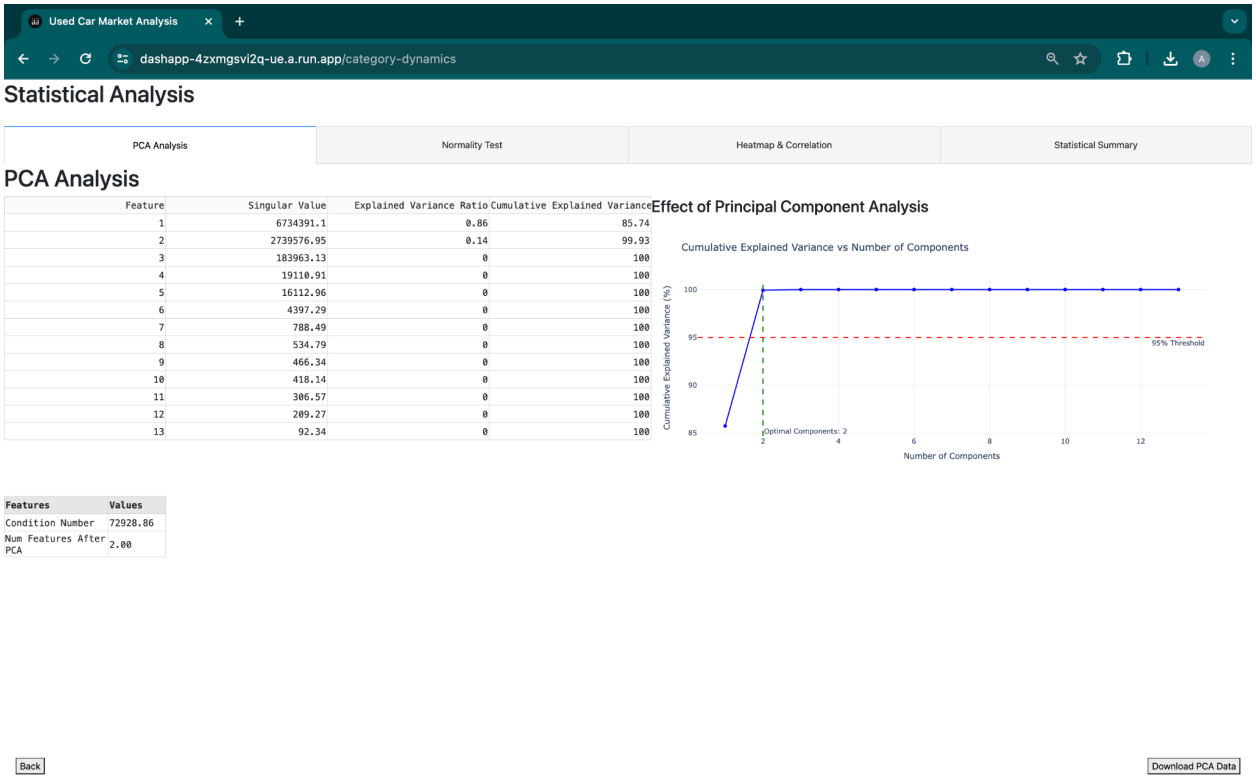
6. Principal Component Analysis :

Principal Component Analysis (PCA) serves as a powerful tool for feature dimensionality reduction, particularly in datasets where multicollinearity or overfitting might be concerns. By transforming the data into a set of orthogonal components that capture the most variance, we can often simplify complex datasets without losing significant information.

6.1 Methods and Results:

For the PCA applied to the cleaned dataset, categorical columns were first encoded numerically to ensure the PCA could process them effectively. Post-encoding, PCA was performed to determine the singular values and the explained variance for each principal component.

- Singular Values and Explained Variance Ratio: The singular values were extracted, and the explained variance ratio was calculated for each principal component, providing insights into the importance of each component in capturing the data's variance.
- Cumulative Explained Variance: By calculating the cumulative explained variance, we determined the minimum number of components required to represent the data effectively. The objective was to find the point where the cumulative explained variance crosses the desired threshold, typically 95%, to ensure we retain most of the information while reducing dimensionality.
- Condition Number: The condition number was calculated to assess the stability and multicollinearity within the dataset. A high condition number suggests potential numerical instability, which PCA helps to mitigate by transforming the data into a space where the variables are uncorrelated.



6.2 Observations

- The first two principal components account for nearly all the variance within the dataset, with the first component alone explaining 85.74% and the second one bringing the cumulative explained variance to 99.93%. This substantial reduction indicates that most of the dataset's information can be encapsulated in just two dimensions, which can significantly simplify further analysis without considerable loss of data integrity.

- The condition number is high at 72928.86, suggesting potential numerical instability or multicollinearity within the original features. This high condition number justifies the use of PCA in this context to improve the numerical robustness of subsequent analyses. After the PCA transformation, the dataset is reduced to just two features, highlighting the effectiveness of PCA in dimensionality reduction while retaining critical information.

7. Normality Tests:

The aim of this part was to determine if the key numerical variables in the dataset, such as price, mileage, tax, mpg, engine size, and value ratio, follow a Gaussian distribution. This is important for choosing appropriate statistical methods for further analysis, as many parametric tests assume normality of the data.

7.1 Methods and Results:

Two tests were applied to assess normality: the Kolmogorov-Smirnov (KS) test and D'Agostino's K^2 Test. Both are robust statistical tests designed to test the hypothesis that a sample comes from a normally distributed population.

- **KS Test:** This non-parametric test compares the sample with a normal distribution in terms of absolute differences between their cumulative distribution functions (CDFs).
- **D'Agostino's K^2 Test:** This test assesses the skewness and kurtosis of the data distribution and compares it to that of a normal distribution.

The tests were conducted using the scipy.stats module, and the results were as follows:

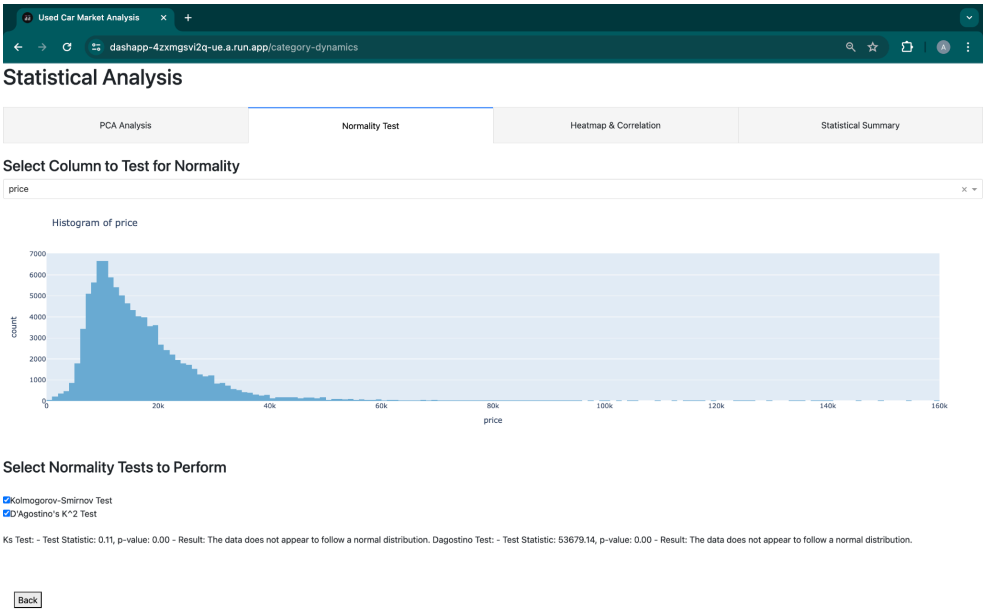
- For each variable tested (price, mileage, tax, mpg, engine size, and value ratio), both the KS and D'Agostino's tests yielded significant p-values ($p < 0.05$), indicating a departure from normality

.

7.2 Observations:

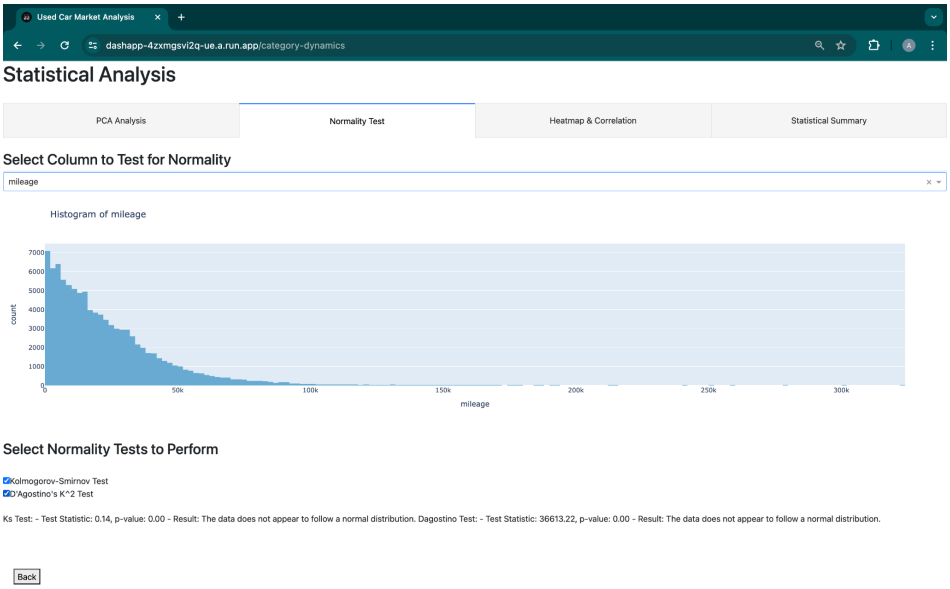
The histogram plots and the test statistics collectively suggest that none of the variables tested are normally distributed. This has several implications:

- **Price:**



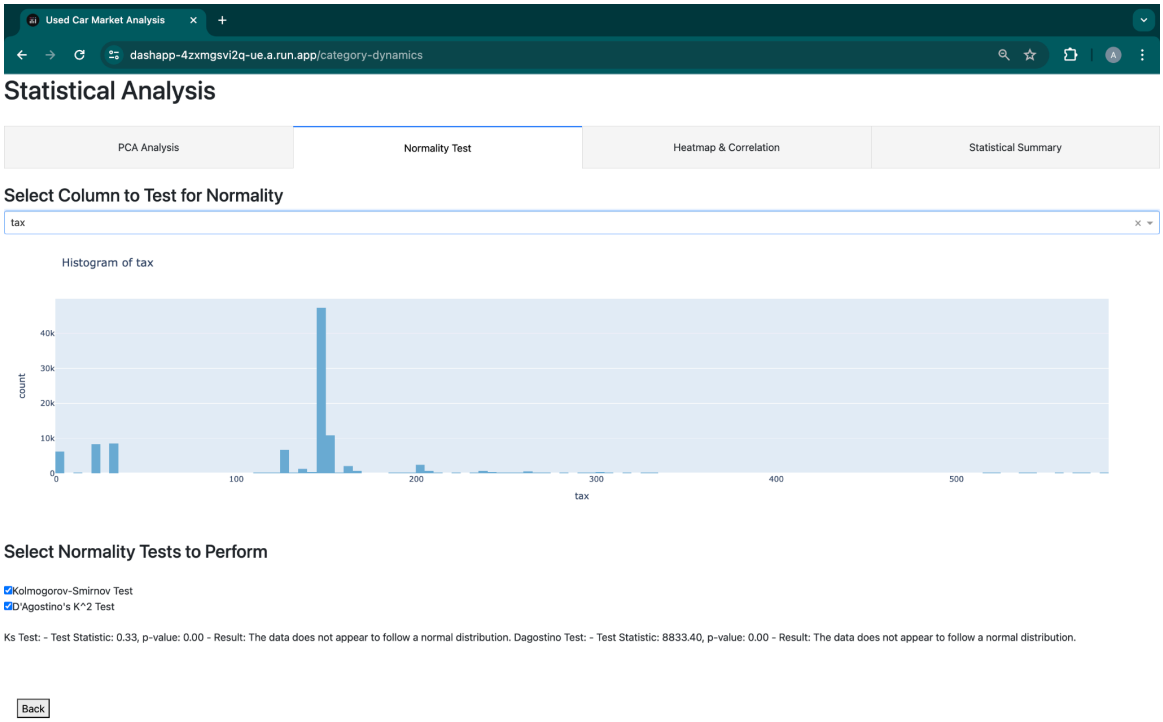
The distribution is skewed with a long tail, indicating a concentration of cars at lower prices and fewer cars at higher prices, which is common in the used car market.

- **Mileage:**



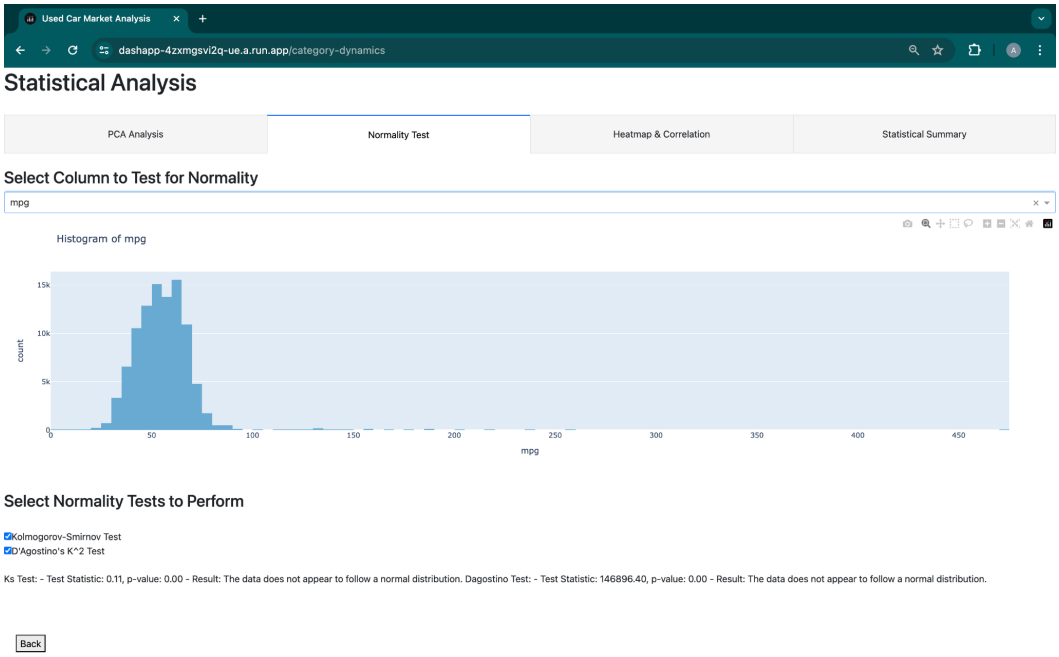
A similar skew is observed, with most cars having lower mileage, and a tail representing vehicles with exceptionally high mileage.

- **Tax:**



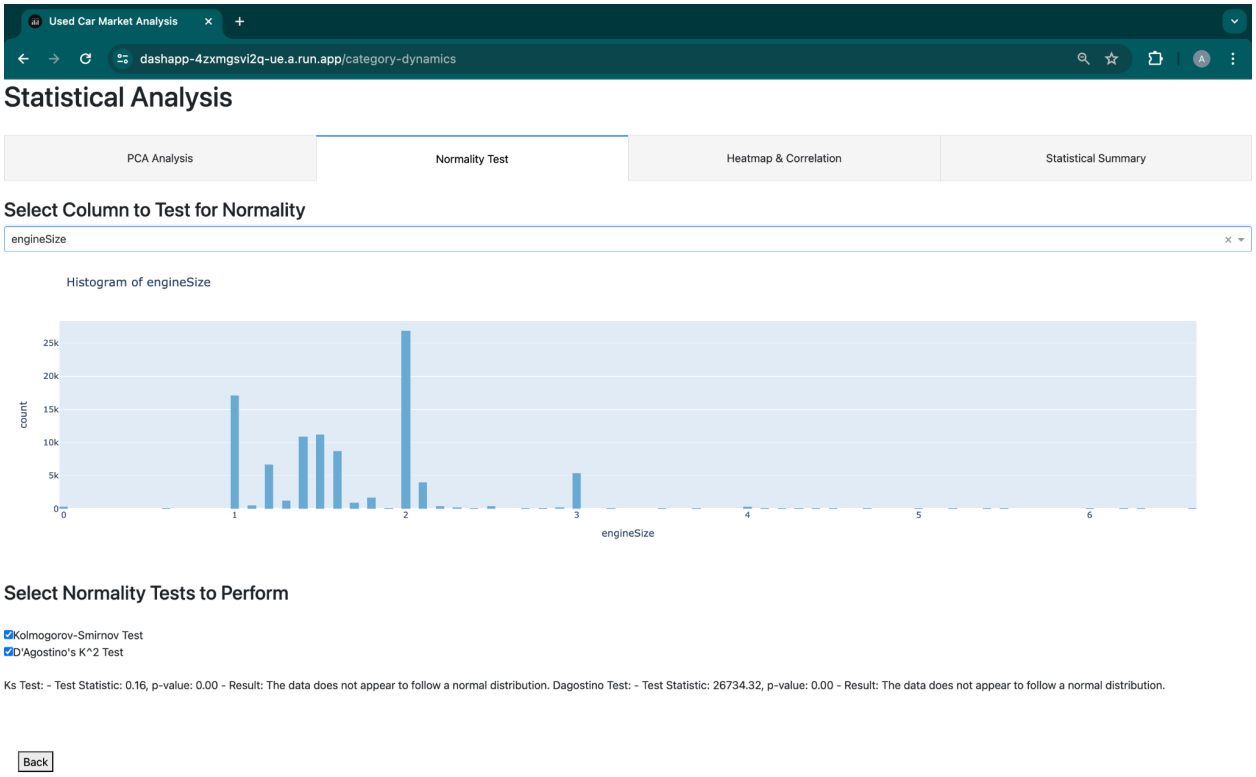
The distribution likely reflects the standardized rates of vehicle tax with deviations representing atypical cases or specific vehicle categories.

- **MPG:**



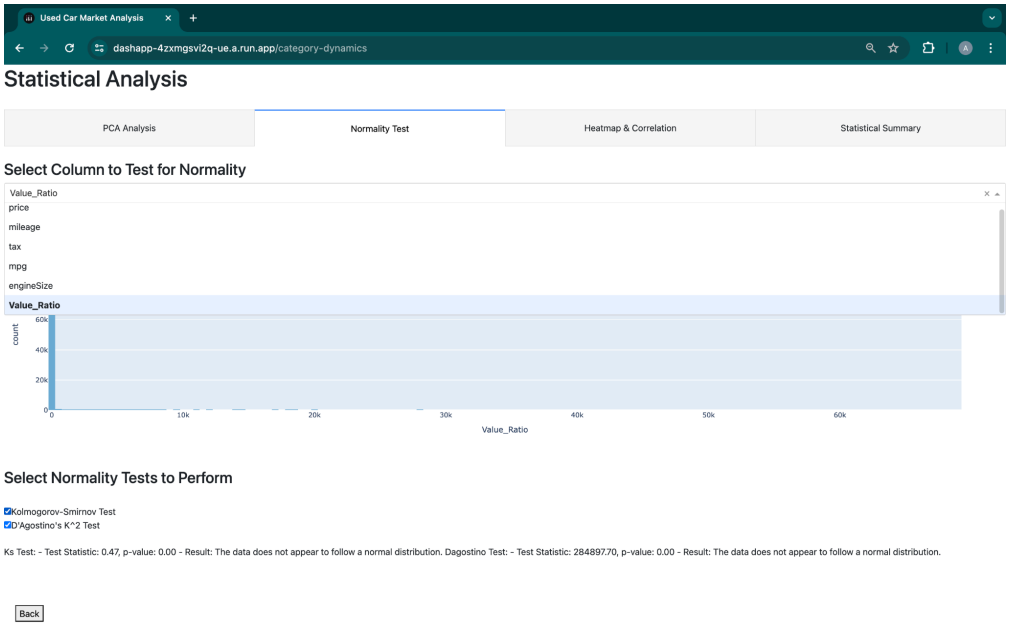
The skew in mpg values may reflect advancements in car fuel efficiency over time or a prevalence of certain types of cars (like SUVs versus compact cars) within the dataset.

- **Engine Size:**



The presence of a skewed distribution suggests a concentration of cars with smaller engines and fewer with larger engines.

- **Value Ratio:**



The skewness indicates that most cars offer a similar price-to-mileage ratio, with fewer cars presenting exceptionally high or low value for money.

8. Heatmap & Pearson correlation coefficient matrix:

The heatmap below illustrates the Pearson correlation coefficients, providing a visual representation of the linear relationships between the variables within the used car dataset. Each cell reflects the correlation between two variables: a value of 1 signifies a perfect positive linear relationship, -1 indicates a perfect negative linear relationship, and 0 implies no linear relationship. The color intensity represents the strength of the correlation, with darker shades signifying stronger relationships.

8.1 Heatmap Observations:

The heatmap provides a visual summary of the Pearson correlation coefficients between several key variables in the used car market dataset. Notable observations include:

- **Price and Engine Size (0.64):** There's a significant positive correlation between the engine size of a car and its price, suggesting that as engine size increases, the price tends to rise as well. This could be attributed to larger engines generally being associated with higher performance or luxury vehicles, which are typically more expensive.

- **Price and Mileage (-0.42):** This moderate negative correlation indicates that cars with higher mileage tend to be less expensive. This relationship is intuitive, as wear and tear, along with higher mileage, usually decrease a car's value.
- **Tax and MPG (-0.45):** A moderate negative correlation suggests that cars with higher fuel efficiency usually have lower tax rates, which may reflect tax incentives for more economical or environmentally friendly vehicles.
- **MPG and Engine Size (-0.25):** There is a negative correlation between engine size and MPG, showing that cars with larger engines tend to have lower fuel efficiency. This is an expected trend since larger engines typically consume more fuel.
- **Price and Tax (0.31):** A mild positive correlation suggests that more expensive cars have slightly higher tax rates. While the correlation is not strong, it might indicate that higher-priced cars fall into higher tax bands or that luxury cars are subject to higher taxes.
- **Engine Size and Value_Ratio (0.28):** A mild positive correlation is observed, implying that vehicles with larger engines may have a higher price per unit of mileage. This could reflect the premium placed on performance characteristics that are often associated with larger engines.
- **Other Variables:** The correlations among the other variables, such as 'Value_Ratio' with 'Mileage' and 'Price,' are relatively weak, indicating no strong linear relationship in the dataset.

The scatter plot matrix presented below offers a visual exploration of the relationships between key variables in our used car dataset. Each plot pairs two different variables, allowing us to observe potential trends and correlations directly. These patterns provide initial insights that guide further statistical analysis and help us understand the factors influencing used car values.

8.2 Scatter Plot Matrix Observations:

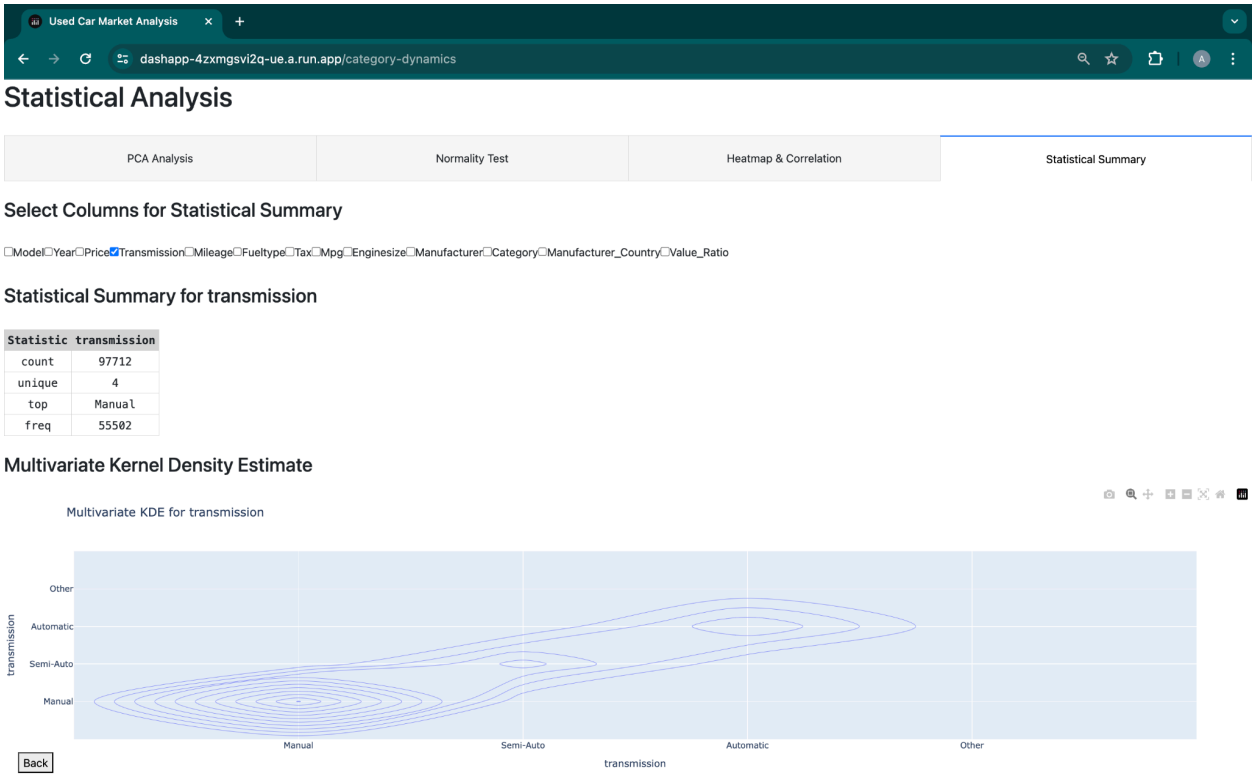
- **Price versus Mileage:** The scatter plot shows an expected downward trend, indicating that cars with higher mileage generally have lower prices. This pattern supports the concept of vehicle depreciation over time and use.
- **Price versus Engine Size:** There is a trend showing that cars with larger engines tend to have higher prices. The scatter plots show a cluster of data points with smaller engines at the lower price range, which spreads out as engine size increases, indicating a variety of prices in the higher engine size category.
- **Mileage versus Tax:** The relationship between mileage and tax doesn't show a clear pattern from the scatter plot, suggesting that there might not be a direct or linear relationship between these variables.
- **Engine Size versus MPG:** The plots reveal an inverse relationship where larger engine sizes are associated with lower MPG, indicating that larger engines are generally less fuel-efficient.
- **Price versus Tax:** The scatter plot does not reveal a strong or clear relationship, suggesting that the sale price of cars does not strongly influence the tax rate, or vice versa.
- **Diagonal Distributions:** The histograms along the diagonal of the matrix provide the distribution of individual variables. It’s apparent that the price, mileage, tax, and engine size do not follow a normal distribution; they are skewed with a concentration of data points toward the lower end of the scale.
- **Value Ratio Scatter Plots:** When plotted against other variables like price and mileage, 'Value Ratio' shows distinct patterns. For instance, 'Value Ratio' versus 'Price' shows a positive trend, indicating that as the price increases, the price per mile can also increase.

9. Statistics:

In this section, we present a detailed statistical summary of the dataset's key attributes, providing essential measures of central tendency and dispersion. This quantitative analysis sets the foundation for interpreting the data's underlying patterns and behaviors. Additionally, we include multivariate kernel density estimates (KDE) for a graphical representation of the data's distribution across multiple dimensions, offering further insight into the complex interrelationships between variables.

The following are the screenshot that shows statistical summary and multivariate KDE plot:

Transmission:



The statistical summary for the 'Transmission' variable presents a breakdown of transmission types within the used car dataset. According to the summary:

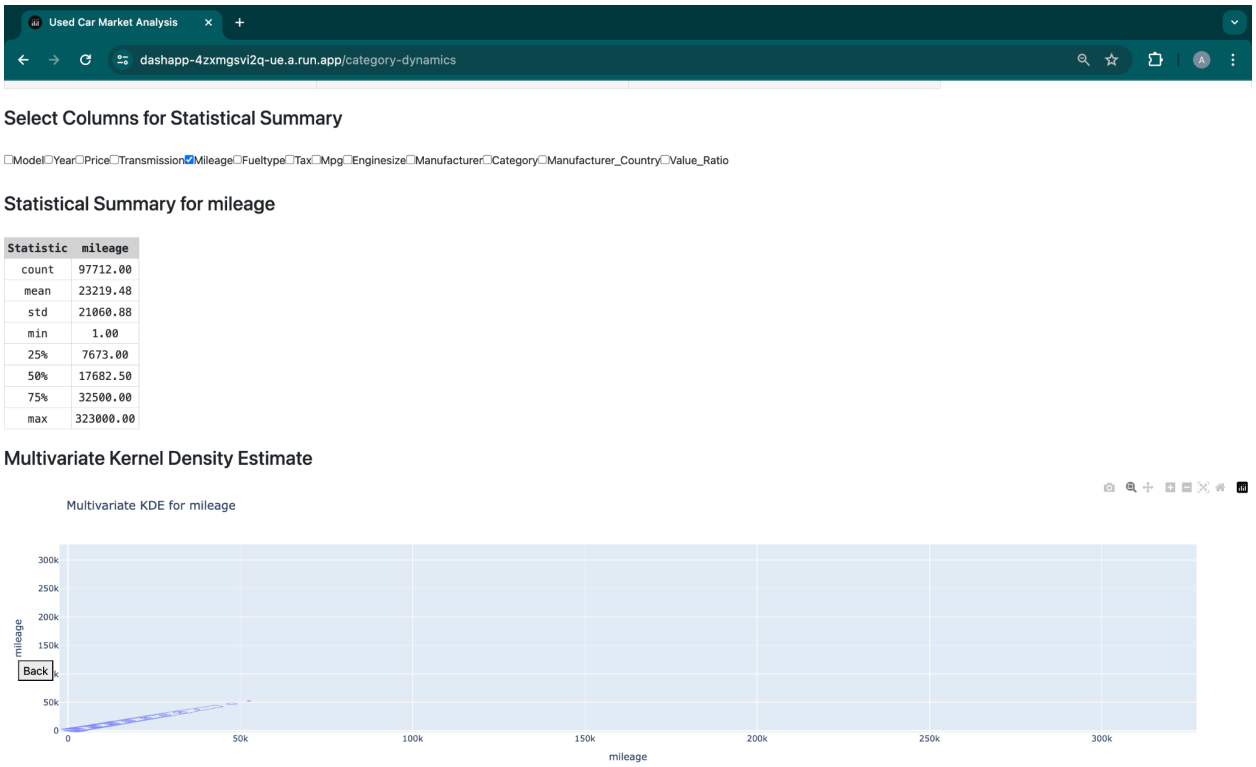
- There are 97,712 entries analyzed for transmission type.
- Four unique types of transmissions have been identified within the dataset.
- The most common transmission type is 'Manual', occurring 55,502 times.

This dominance of manual transmission could reflect a variety of market factors, including manufacturing trends, consumer preferences, or economic considerations.

The multivariate KDE plot for 'Transmission' displays the density of cars across different transmission types. From the KDE contours:

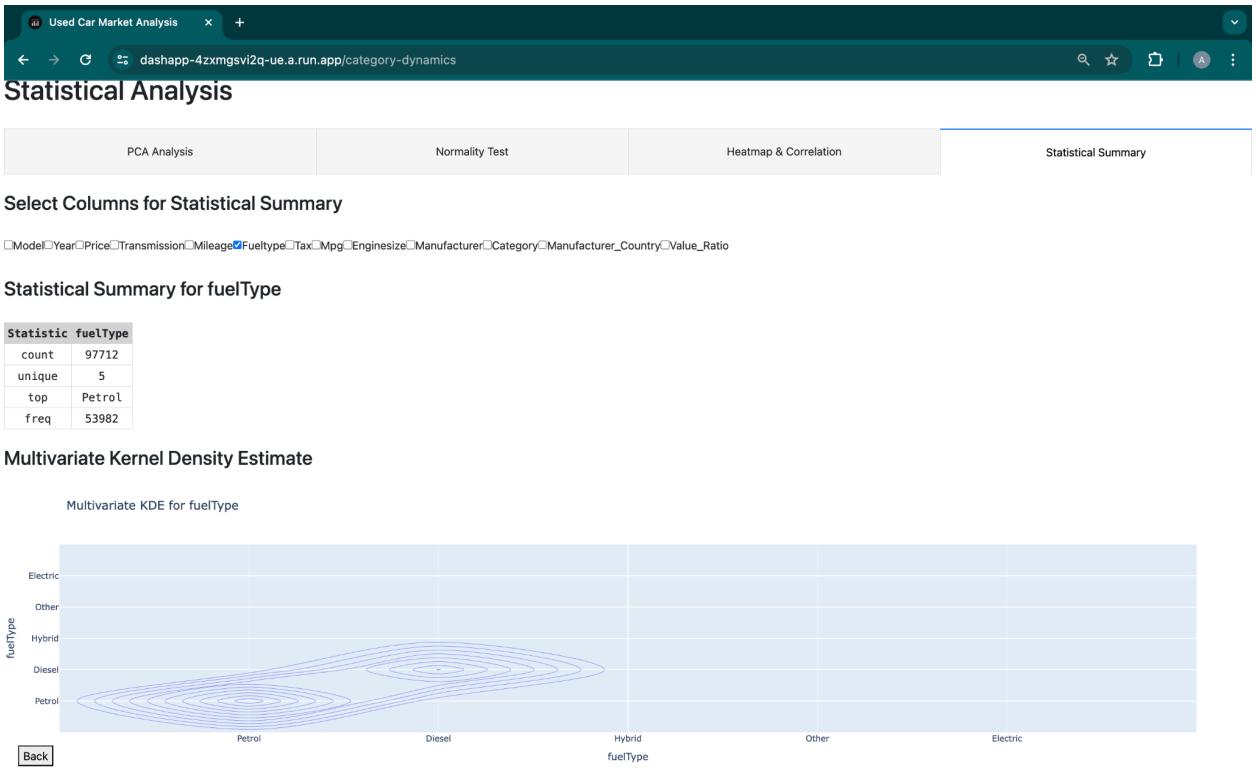
- There is a significant concentration around the 'Manual' transmission, indicating a higher prevalence of manual cars in the dataset.
- Other transmission types like 'Automatic', 'Semi-Auto', and 'Other' show varied densities, which suggests differing levels of popularity or availability in the market.

Mileage:



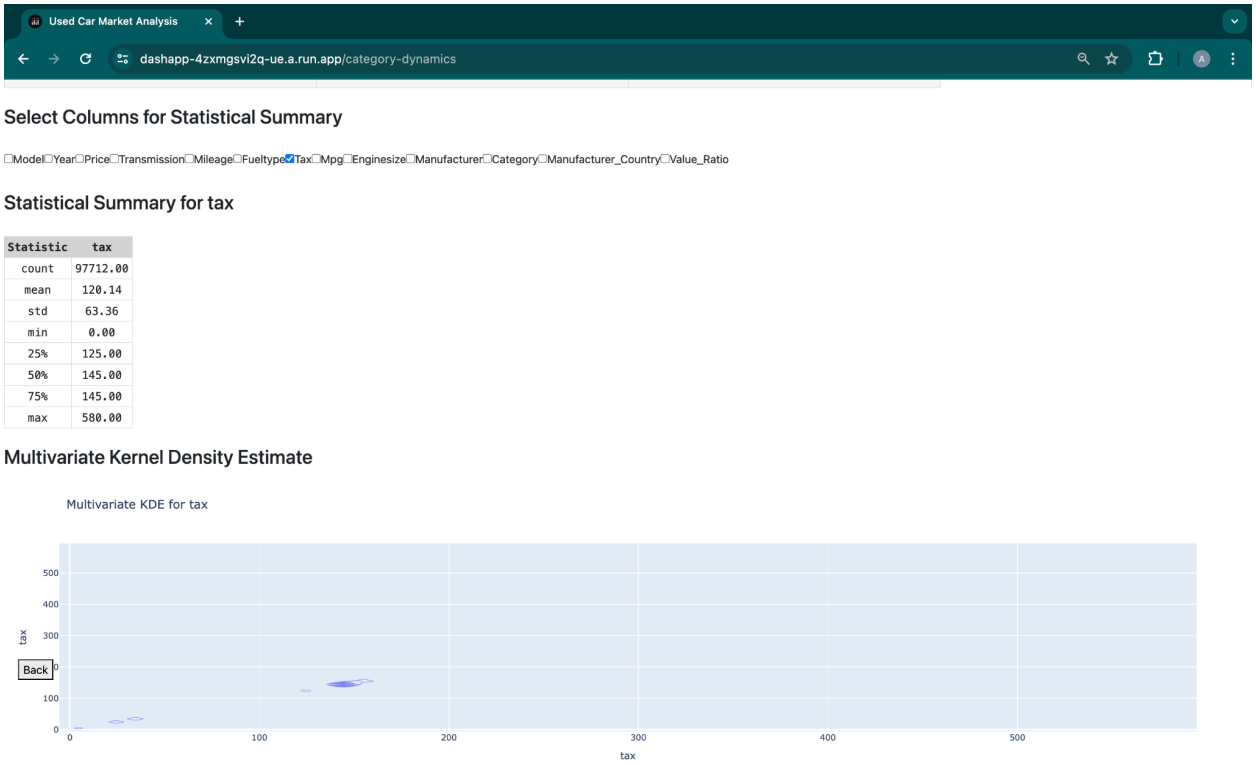
The statistical summary for 'mileage' highlights the distribution of miles driven by the cars in the dataset. The mean mileage is approximately 32,319 miles, with a standard deviation of 21,680 miles, indicating variability in car usage. The median value of 17,682 miles suggests that half of the cars have mileage below this point, which might be due to a mix of nearly-new and older vehicles. The KDE plot for mileage shows the density of cars across different mileage ranges, with a peak in lower mileage zones, reflecting a higher concentration of cars with less usage.

Fuel Type:



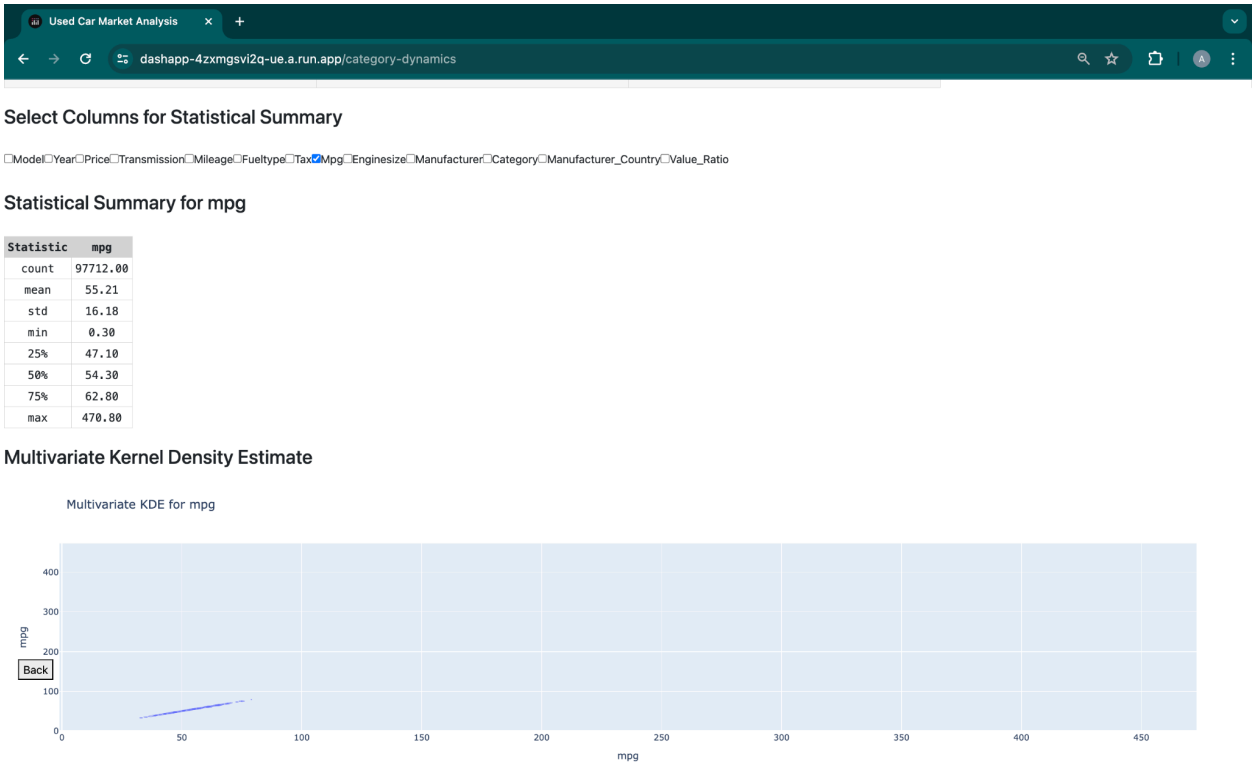
The dataset encompasses 97,712 vehicles, showing a diverse fuel type usage with 'Petrol' being the most common, occurring 53,982 times. The multivariate KDE illustrates the predominance of petrol vehicles, with notable density peaks, and less dense regions for diesel, hybrid, electric, and other fuel types, reflecting the variety and evolving preferences in the car market.

Tax:



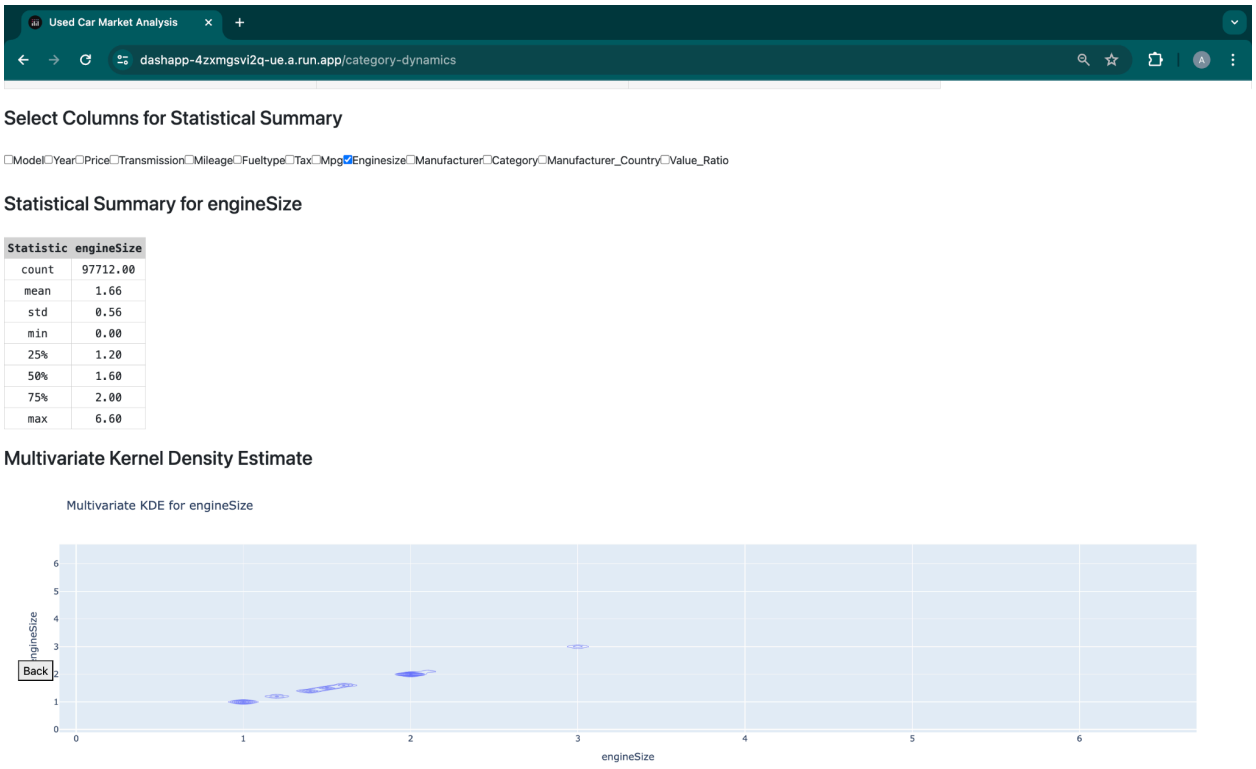
The statistical summary for 'tax' shows a mean value of 120.14 with a standard deviation of 63.36, suggesting variability in the tax rates applied to the vehicles in the dataset. The median and the third quartile (75%) both stand at 145, indicating a concentration of values at this tax bracket. The KDE plot highlights a cluster around this median tax rate, with a tail extending towards higher tax rates, potentially reflecting the range of vehicles from economical to luxury classes that are subject to various tax implications.

MPG:



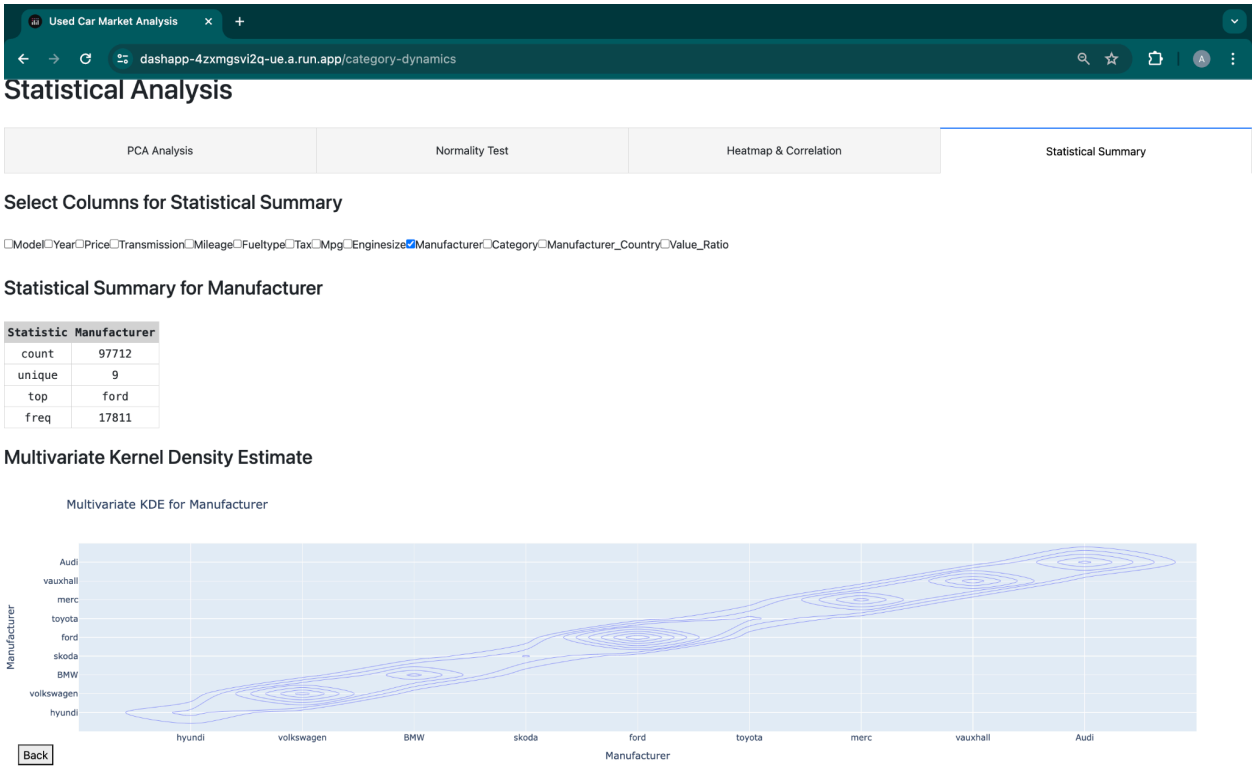
The statistical data for 'mpg' (miles per gallon) reveals an average fuel efficiency across the dataset of 55.21 mpg, with a standard deviation of 16.18 mpg, indicating a moderate spread of values. The median mpg is 54.30, slightly less than the mean, suggesting a slight skew towards lower mpg values in the dataset. The maximum reported mpg is exceptionally high at 470.80, which might represent electric or hybrid vehicles with combined fuel efficiency measures. The KDE plot likely shows a peak around the median value with a long tail towards the higher mpg values, reflecting the range of vehicle efficiencies from conventional combustion engines to more efficient or alternatively fueled vehicles.

Engine Size:



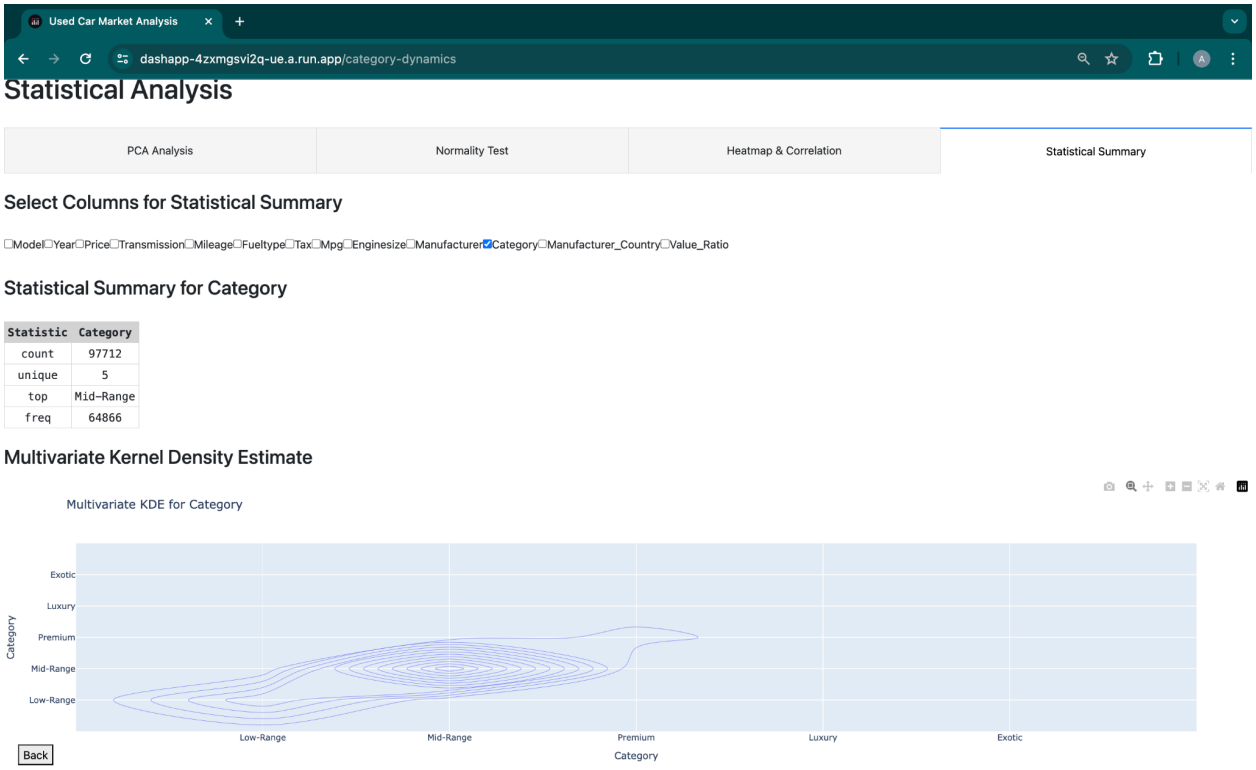
The statistical summary for 'engineSize' shows an average engine displacement of 1.66 liters across the 97,712 vehicles in the dataset, with a standard deviation of 0.56 liters, indicating most engines are close to the average size. The median of 1.60 liters suggests a slight skew toward smaller engine sizes, and the range extends to a maximum of 6.60 liters, which likely includes high-performance or commercial vehicles. The multivariate KDE indicates peaks at common engine sizes, reflecting the market's standard offerings.

Manufacturer:



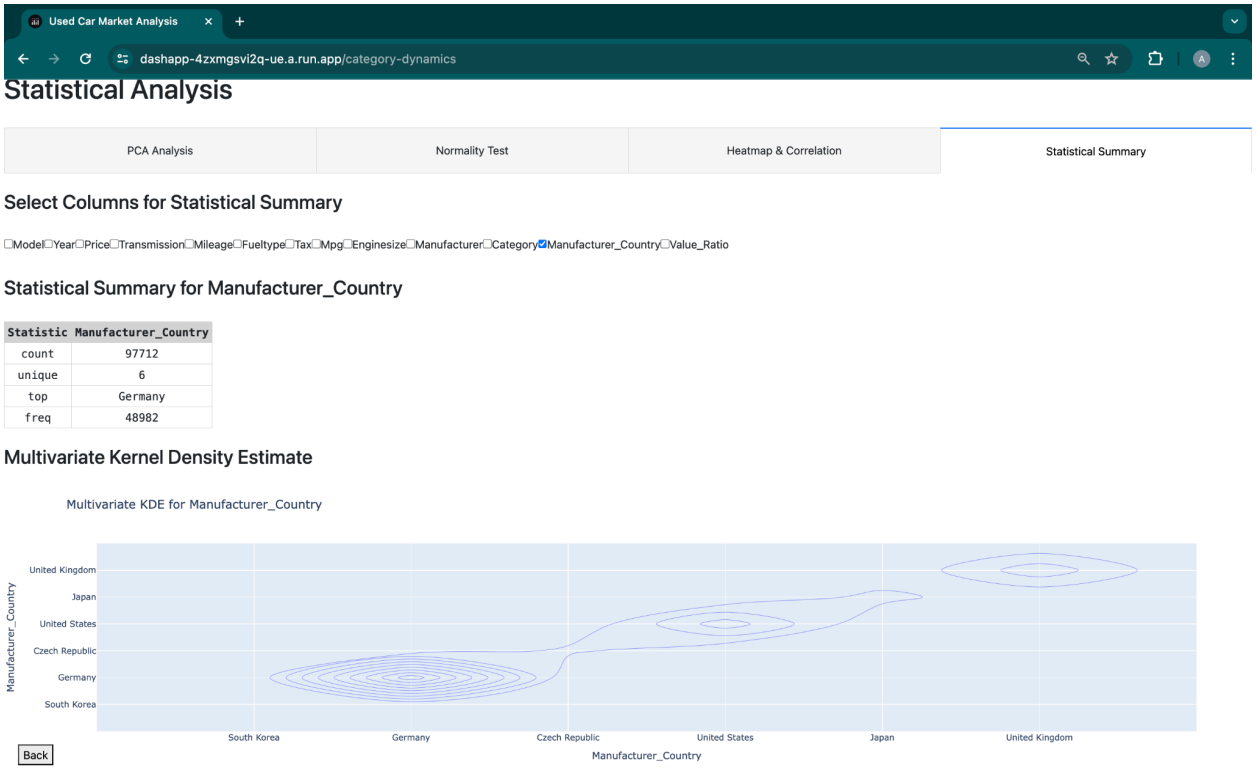
The 'Manufacturer' statistical summary shows diversity among the brands with Ford as the most frequent, underscoring its strong market presence. The multivariate KDE plot reveals concentrated peaks for Ford and a spread for other makes, illustrating the range of consumer choices in the dataset.

Category:



In our car market dataset, 'Mid-Range' is identified as the most common category, dominating the distribution. The KDE reflects this prevalence, with a strong central density for mid-range vehicles, while luxury and exotic categories show less density, indicating their relative scarcity.

Manufacturer Country:



The dataset includes vehicles from six countries, with Germany leading in frequency, reflecting its strong foothold in the car industry. The KDE plot for 'Manufacturer_Country' emphasizes the dominance of German cars, along with significant peaks for South Korean and Japanese manufacturers, suggesting varied consumer preferences. Lesser densities for the United States, United Kingdom, and Czech Republic indicate these cars occupy smaller segments of the market.

Value_Ratio:

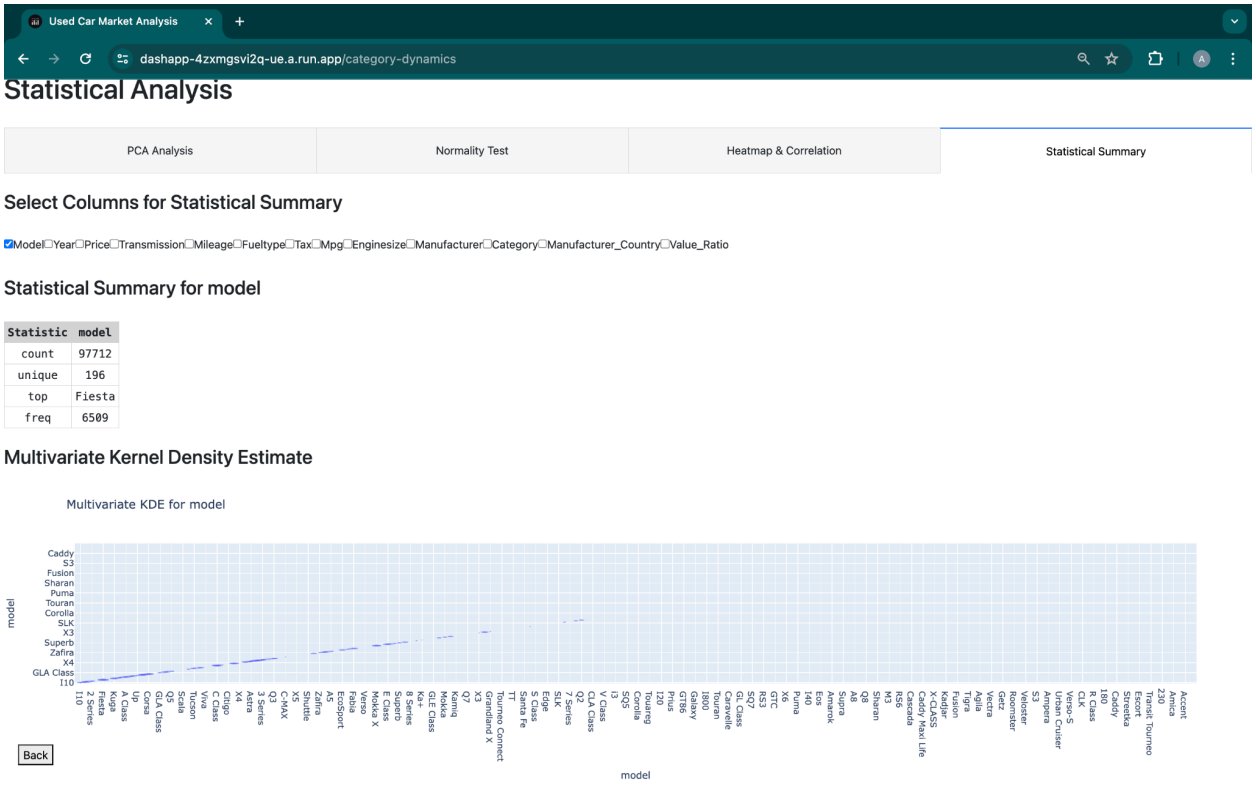
☐Model☐Year☐Price☐Transmission☐Mileage☐Fueltype☐Tax☐Mpg☐Enginesize☐Manufacturer☐Category☐Man

Statistical Summary for Value_Ratio

Statistic	Value_Ratio
count	97712.00
mean	54.73
std	593.58
min	0.00
25%	0.36
50%	0.79
75%	2.50
max	69000.00

The 'Value_Ratio' showcases the cost-to-mileage value across the dataset, with a mean of 54.73. A significant standard deviation of 593.58 suggests a vast range of values, from very economical to potentially overpriced vehicles. The median of 0.79 points to a majority of the cars offering reasonable value for their mileage. However, the maximum ratio of 69,000 indicates extreme outliers, possibly signifying collector's items or rare vehicles with exceptional value propositions.

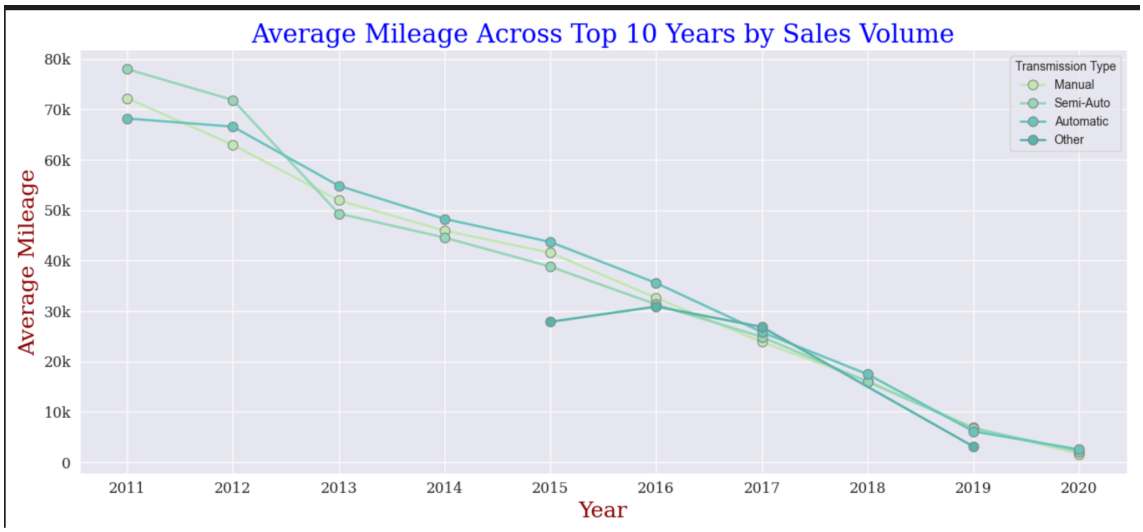
Model:


Used Car Market Analysis
dashapp-4zxmgsvi2q-ue.a.run.app/category-dynamics
Statistical Analysis
PCA Analysis
Normality Test
Heatmap & Correlation
Statistical Summary
Select Columns for Statistical Summary
Model Year Price Transmission Mileage Fueltype Tax Mpg Enginesize Manufacturer Category Manufacturer_Country Value_Ratio
Statistical Summary for model
Statistic model
count 97712
unique 196
top Fiesta
freq 6509
Multivariate Kernel Density Estimate
Multivariate KDE for model
model
Back

10. Data Analysis and Visualization Insights:

In this section, we're going to display a collection of static plots and share their key findings. Every plot gives us a clear picture of specific aspects of our data, and I'll walk you through the main points and their potential implications. Let’s begin exploring these visuals and uncover the insights they offer.

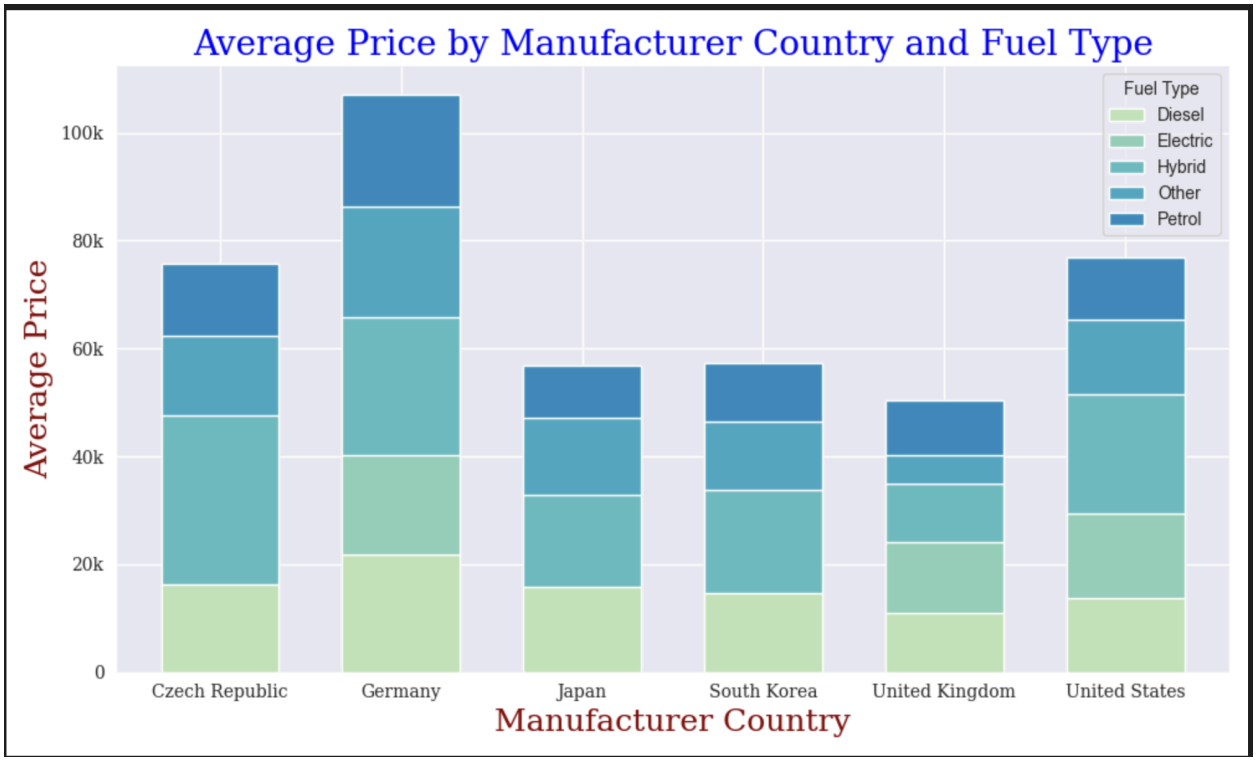
10.1 Line Plot: Average Mileage Across Top 10 Years by Sales Volume



- Over the decade from 2011 to 2020, the average mileage across the top 10 years by sales volume shows a consistent decline for all transmission types, suggesting a potential increase in the sales of newer vehicles with lower mileage or a general trend toward vehicles being driven more during their first few years.

- Vehicles with manual transmissions start off with the highest average mileage in 2011 and maintain a lead until 2017, after which they are overtaken by vehicles with semi-automatic and automatic transmissions, which could indicate a shift in consumer preference or driving habits over the years.

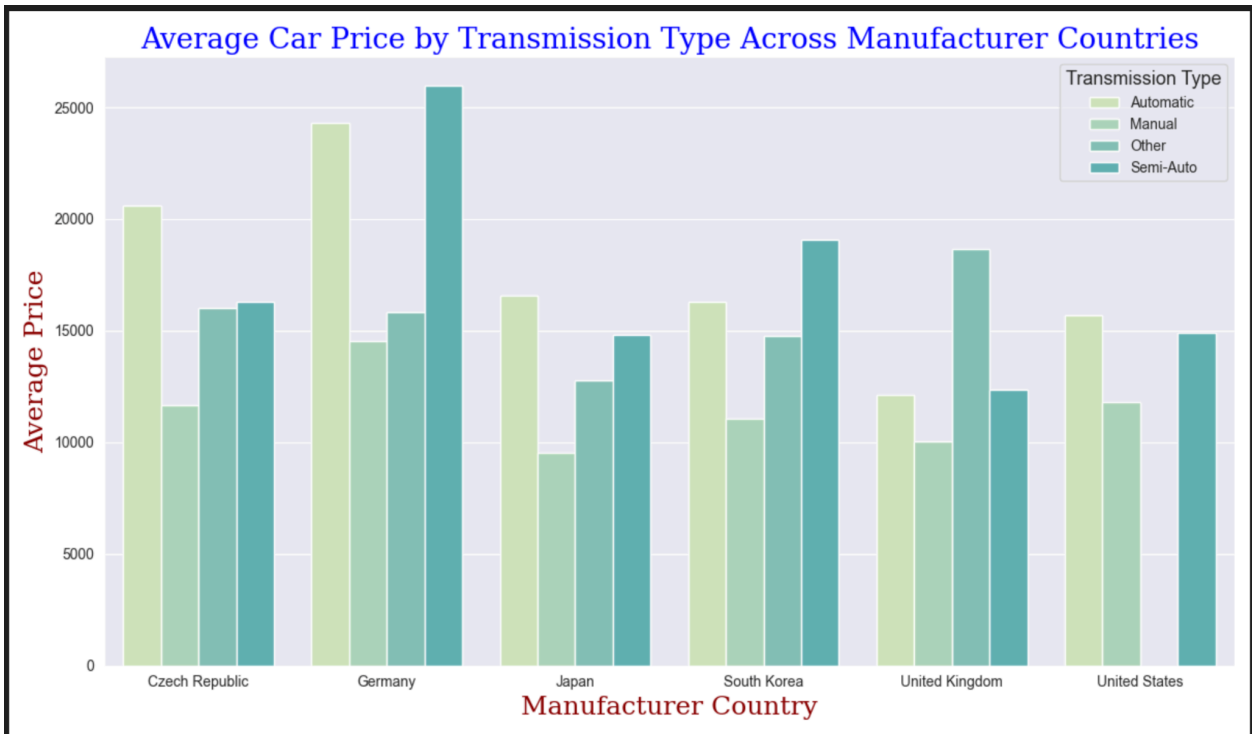
10.2 Stacked Bar Plot: Average Price by Manufacturer Country and Fuel Type



- Vehicles from Germany show the highest average price among all manufacturer countries, with a notable proportion of that average price coming from diesel and petrol vehicles, suggesting that German manufacturers may be positioned at a higher end of the market or that these fuel types are premium offerings in their lineup.

- Electric and hybrid vehicles contribute significantly to the average price in the United States and the United Kingdom, indicating a possibly strong market for alternative fuel vehicles in these countries, which may reflect a higher consumer demand or favorable government policies supporting green energy vehicles in these regions.

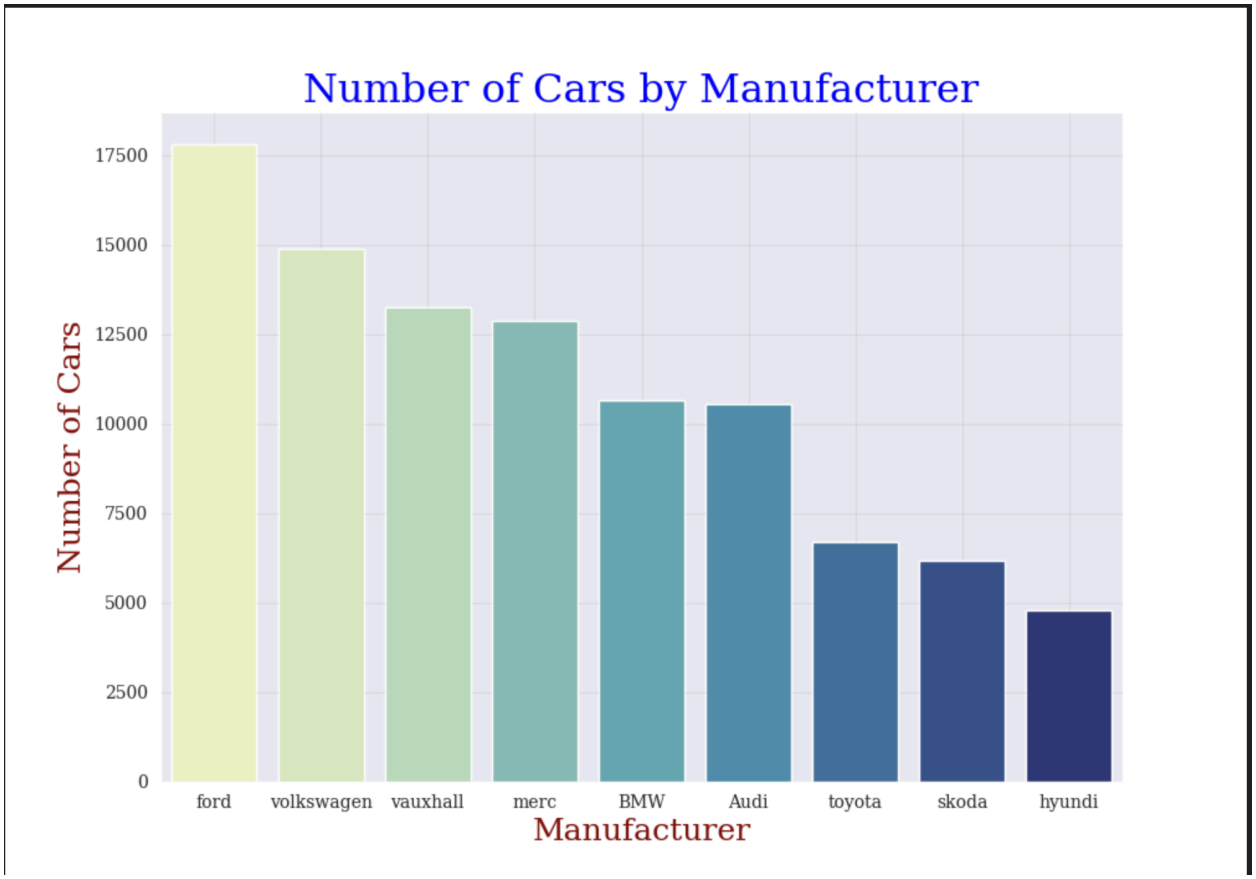
10.3 Grouped Bar Plot: Average Car Price by Transmission Type Across Manufacturer Countries



- German manufacturers command the highest average car prices for both automatic and semi-automatic transmissions when compared to other countries, indicating a possible premium positioning of German-engineered cars or a focus on higher-end market segments.

- The United States shows a notably high average price for cars with 'Other' transmission types, suggesting the presence of niche or specialty vehicles in the market that command premium pricing, or that 'Other' types may include advanced or non-traditional transmission technologies that are more expensive.

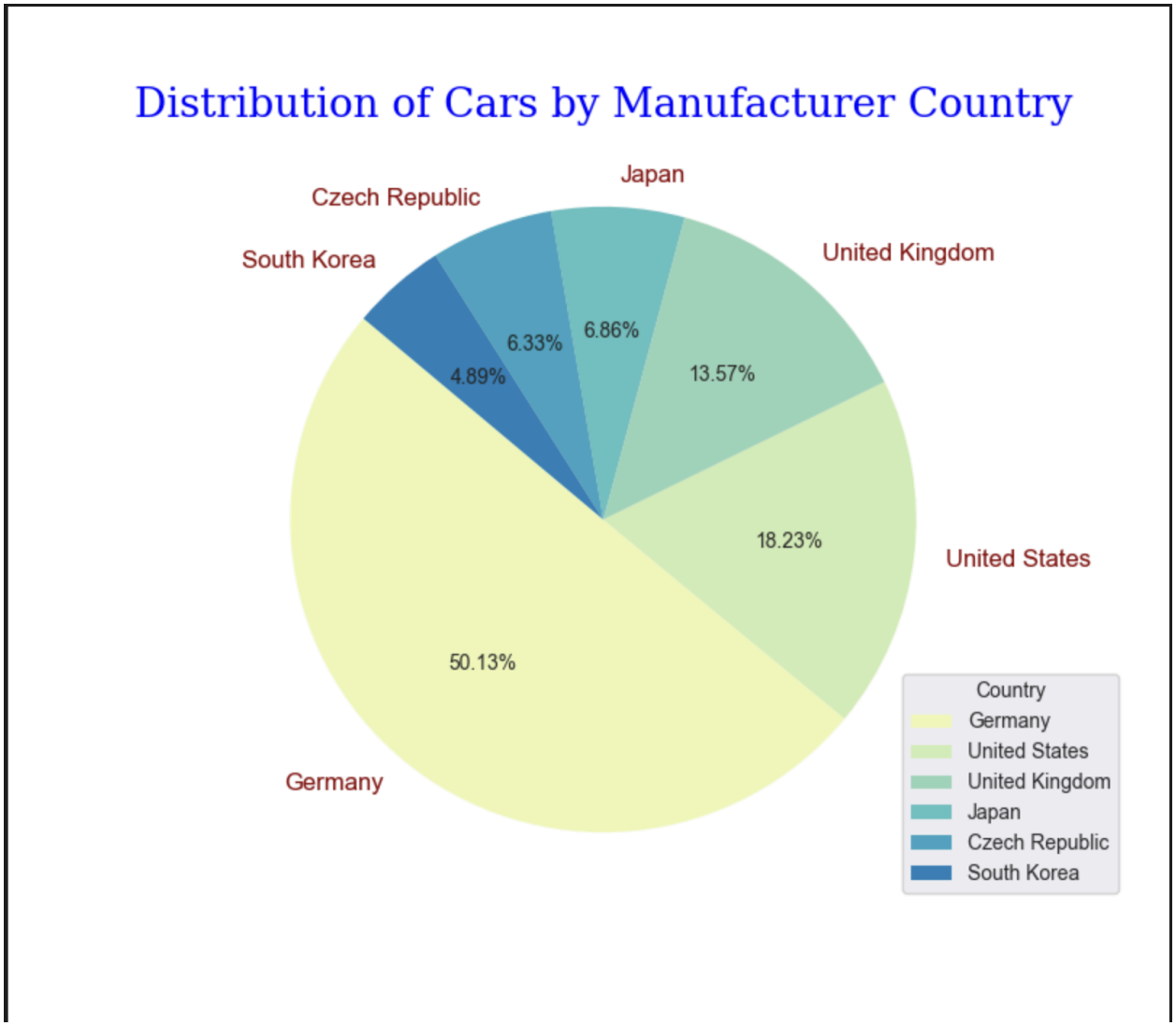
10.4 Count Plot: Number of Cars by Manufacturer



- Ford leads the chart with the highest number of cars by a significant margin compared to other manufacturers listed, indicating a strong market presence or preference for Ford vehicles in the dataset's context.

- While premium brands like BMW and Audi show a substantial number of cars, they are outnumbered by mass-market brands like Volkswagen and Vauxhall, suggesting that volume sales in this dataset are dominated by manufacturers who target a wider customer base.

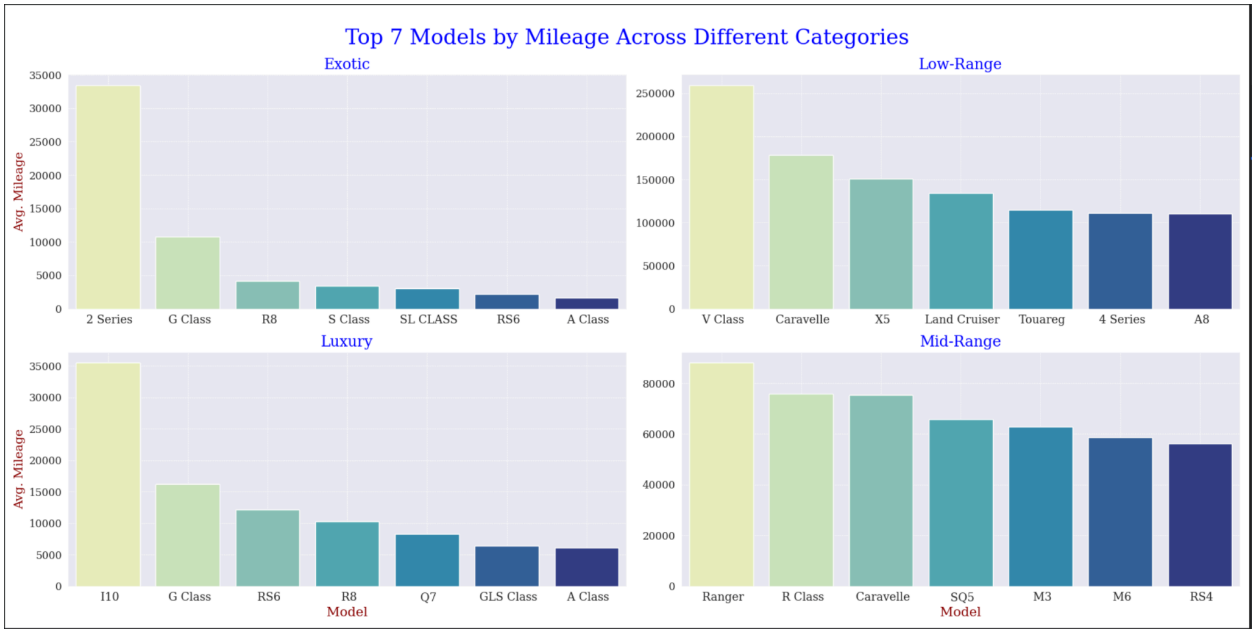
10.5 Pie Chart: Distribution of Cars by Manufacturer Country



- German manufacturers dominate this car distribution, accounting for over half of the cars, which may suggest a strong preference for German engineering or a significant market share of German cars in the dataset.

- The United States and the United Kingdom together make up approximately 31% of the car distribution, indicating these countries also have a significant influence on the car market in the context of the dataset, though not as prominent as Germany.

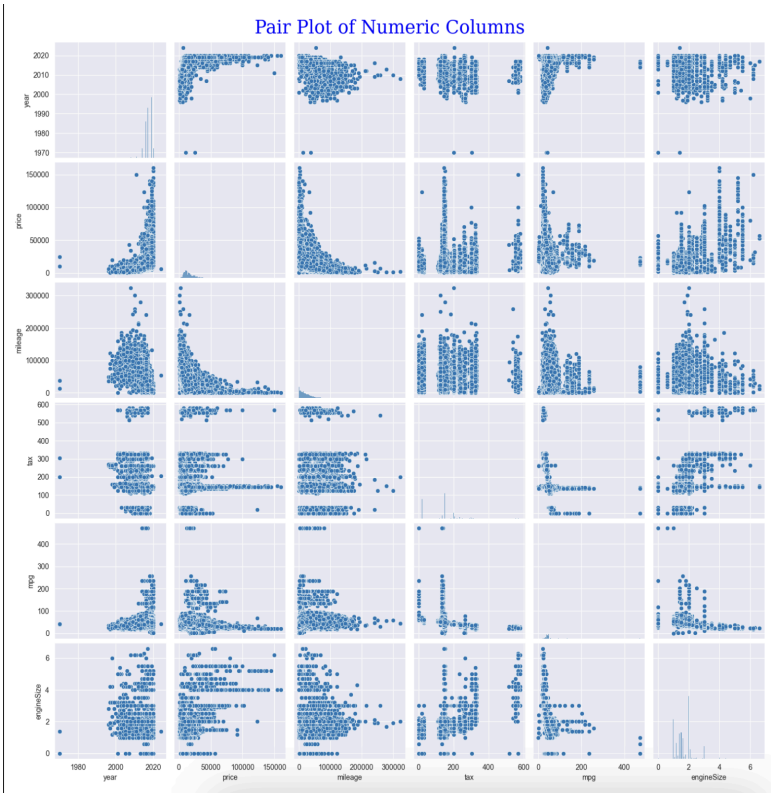
10.6 DistPlot: Mileage Distribution by Transmission Type



- The 2 Series model in the exotic category has the highest average mileage compared to other exotic vehicles, which could imply it's a popular choice for consumers who drive their luxury cars more frequently, or it has been on the market longer allowing for higher mileage accumulation.

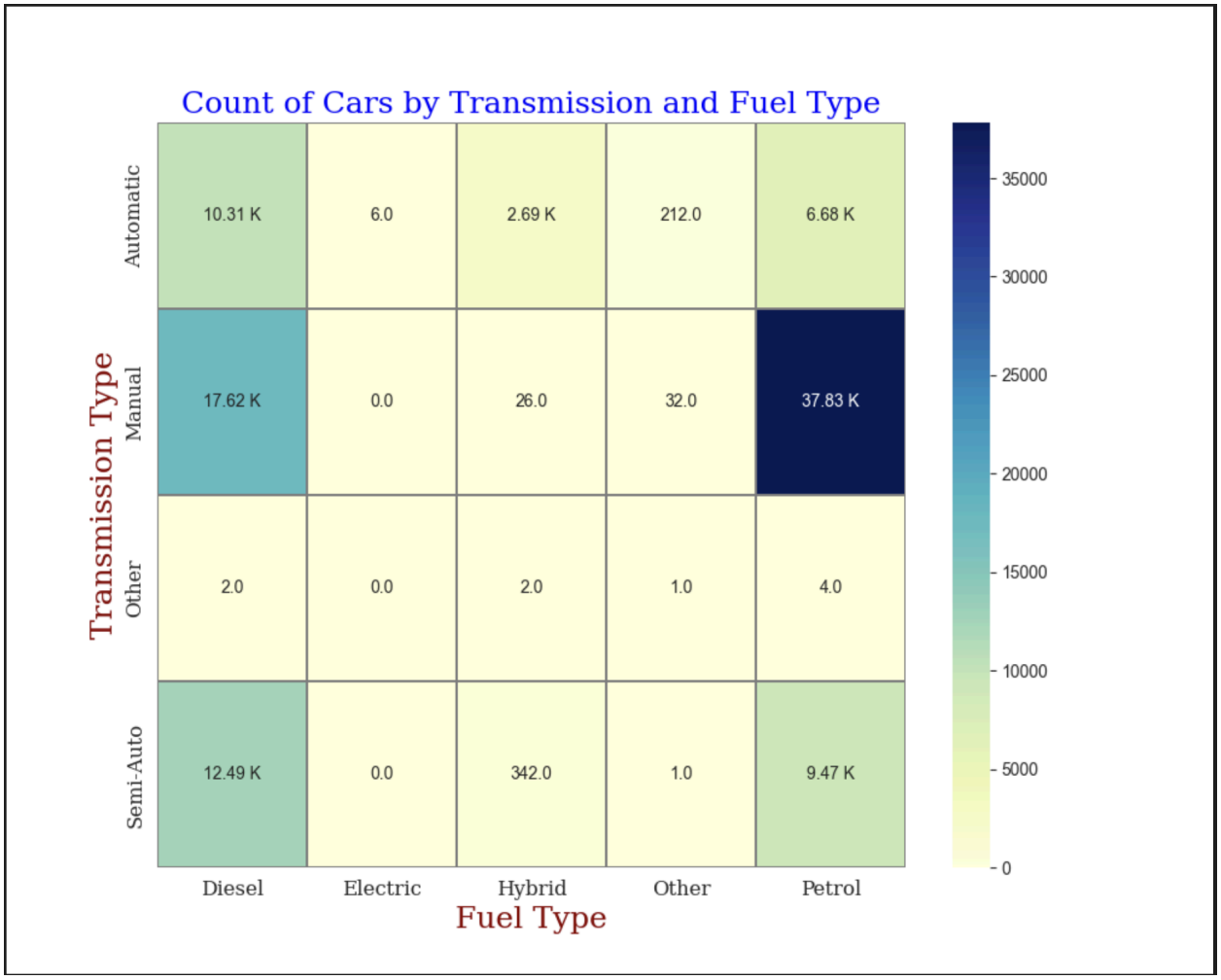
- In the mid-range category, the Ranger has the highest average mileage, suggesting it may be favored for work or utility purposes, which typically involve more frequent use and longer distances, compared to the other mid-range models shown.

10.7 Pair Plot: Pairwise Relationships Between Multiple Variables



- The pair plot reveals strong correlations between certain variables; for instance, there's a notable inverse relationship between the age of a car (year) and its price, suggesting that newer cars tend to be more expensive.
- Mileage appears to have a negative correlation with price, indicating that cars with lower mileage command higher prices, which aligns with market expectations around vehicle wear and tear.
- Engine size shows a varied relationship with other variables; while there seems to be a moderate positive correlation with price, suggesting larger engines may be valued higher, its relation with tax and mpg indicates more complex patterns, possibly due to varying engine types and efficiency levels.

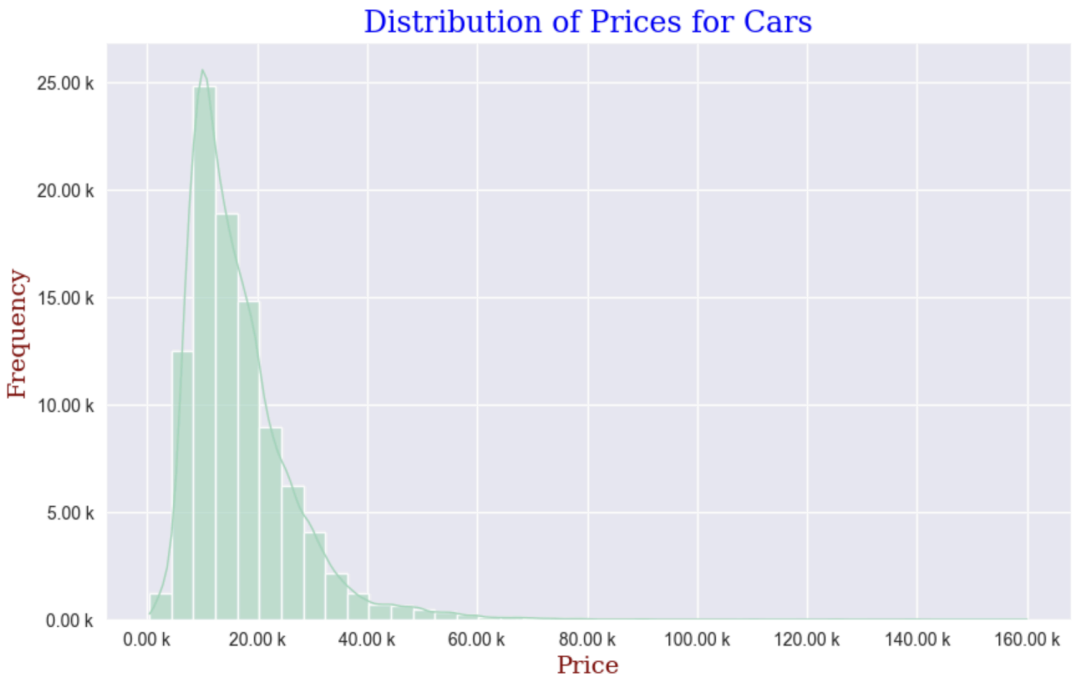
10.8 Heatmap: Count of Cars by Transmission and Fuel Type



- Manual transmission vehicles with petrol fuel type are the most common, as shown by the highest count of 37.83K, indicating a strong market preference or availability for this combination in the dataset.

- Diesel vehicles with automatic transmission follow as the second most prevalent, with a significant count of 10.31K cars, suggesting that consumers also favor the convenience of automatic transmissions in diesel-fueled cars.

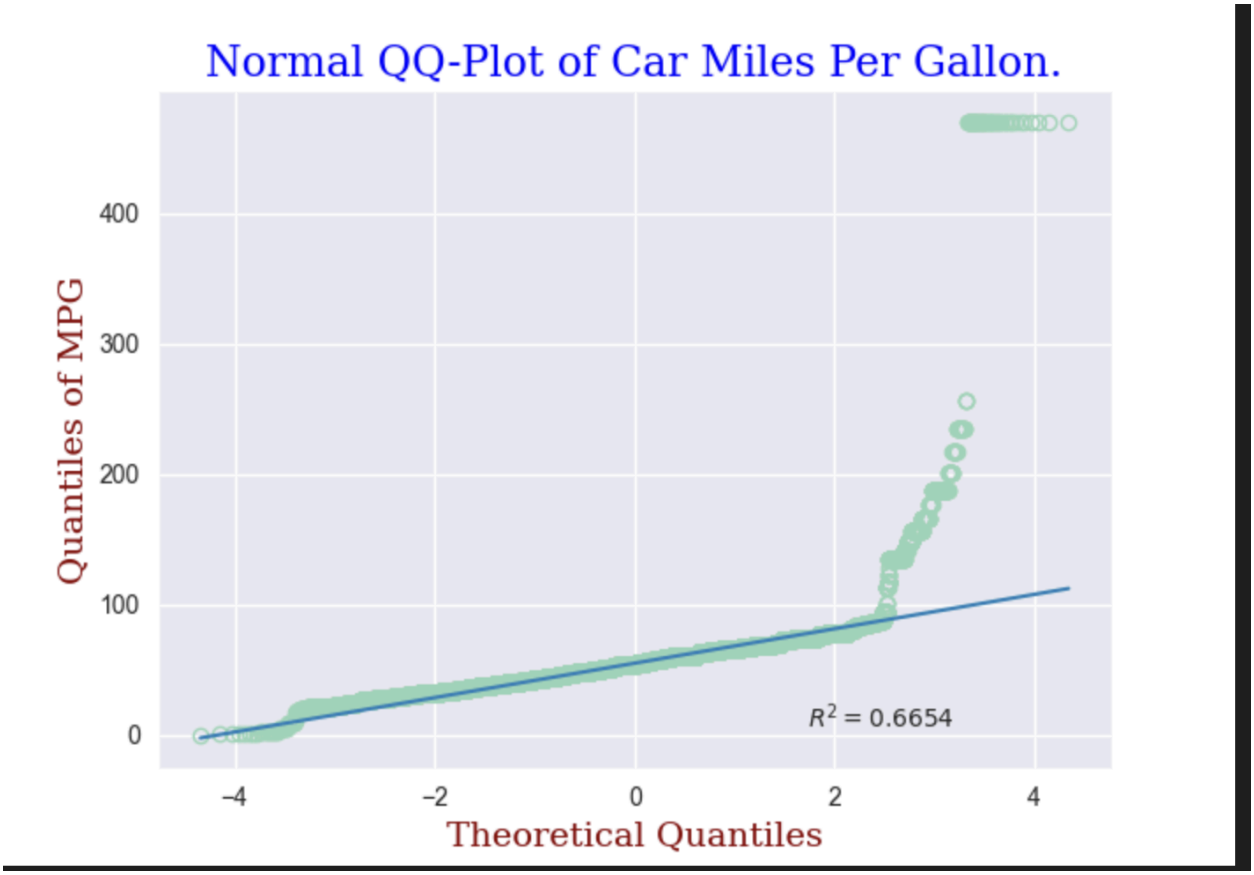
10.9 Histogram with KDE: Distribution of Prices for Cars



- The distribution of car prices is right-skewed, with the majority of cars clustered at lower price points and a tail extending towards higher prices, indicating that affordable cars are more common than expensive ones in this dataset.

- There is a steep drop-off in frequency as car prices increase beyond 20,000, suggesting that higher-priced cars are much rarer and could represent luxury or premium market segments.

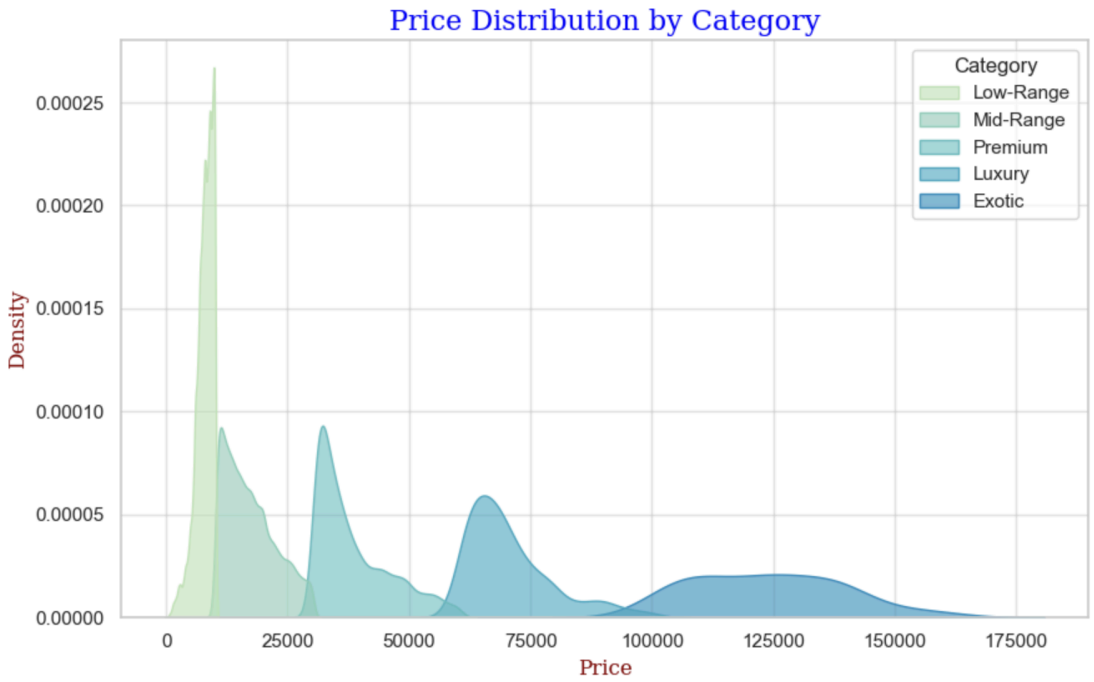
10.10 QQ-Plot: Normal QQ-Plot of Car Miles Per Gallon



- The Normal Q-Q Plot of car miles per gallon (MPG) deviates from the line at both ends, indicating that the distribution of MPG is not normally distributed and has outliers, particularly in the upper range, where cars achieve much higher MPG than what would be expected in a normal distribution.

- With an R^2 value of 0.6654, there is a moderate fit to a normal distribution for the majority of the data, but the noticeable deviation of points at the extremes suggests that the MPG data contains cars with exceptionally high or low efficiency relative to the average.

10.11 KDE Plot: Price Distribution by Category



- The density plot shows a clear peak in the lower price range, suggesting that low-range cars are the most common in this dataset and likely the most accessible to consumers.

- There are distinct peaks for mid-range, premium, luxury, and exotic categories, each with progressively lower density, indicating that as car prices increase, fewer models are available or sold within those higher-priced categories.

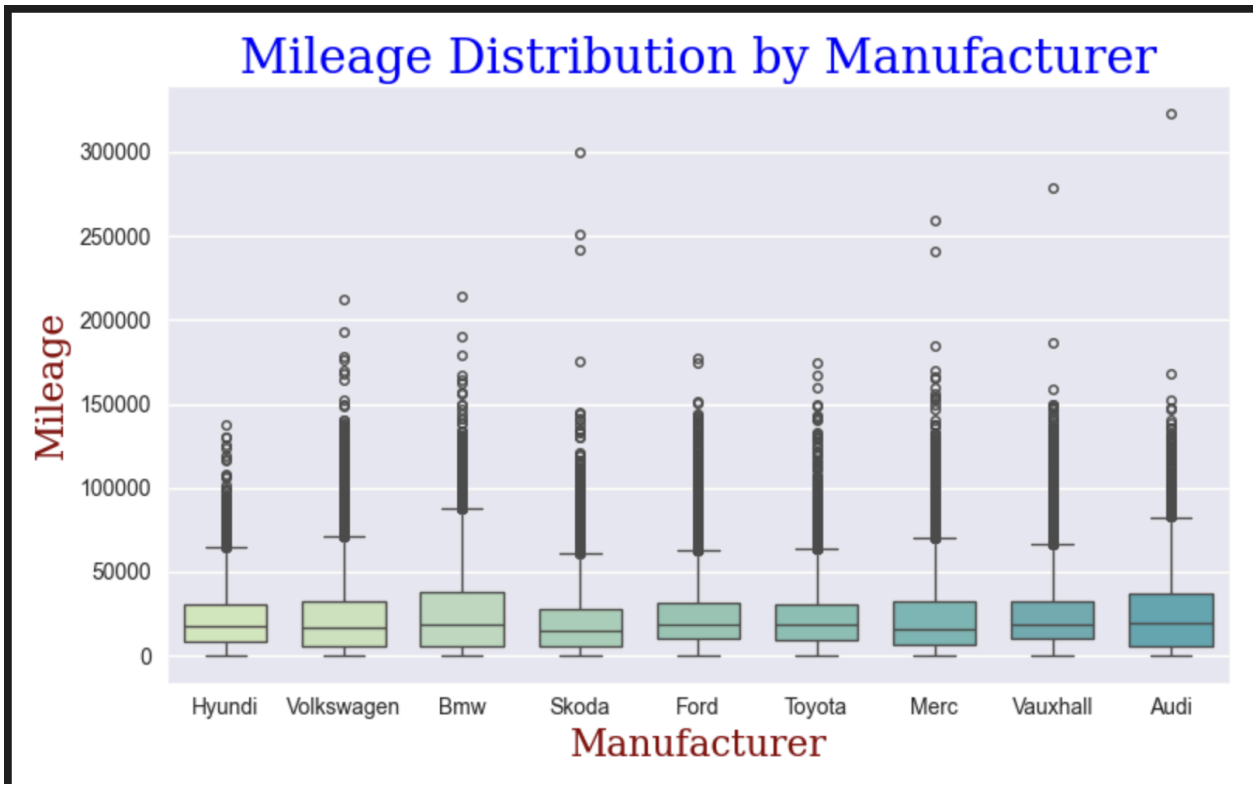
10.12 lm or Regression Plot: Comparative Price vs. Mileage across Different Countries

- Across all countries shown, there is a clear negative relationship between price and mileage, indicating that as a car's mileage increases, its price tends to decrease, which is consistent with the general depreciation pattern of vehicles over time.

- Germany stands out with a wider spread in prices at lower mileage compared to other countries, suggesting a range of car values including high-end models that start at higher prices even when new or with low mileage.

- The United States exhibits a denser clustering of data points at lower price ranges and mileage, potentially indicating a large market for more affordable vehicles and perhaps a quicker depreciation or turnover rate for cars in this market segment.

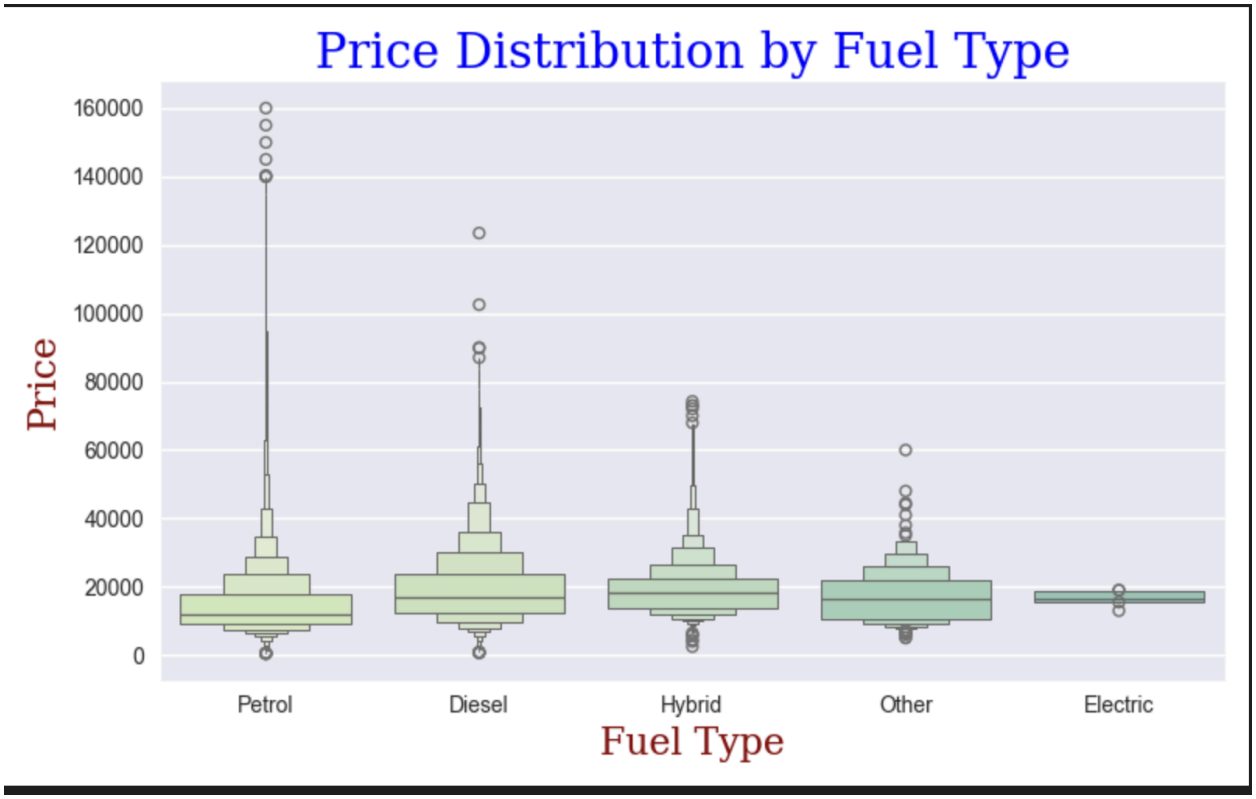
10.13 Multivariate Box Plot: Mileage Distribution by Manufacturer



- The boxplots show that there is a wide range of mileages for cars across all manufacturers, with several outliers indicating that some vehicles have exceptionally high mileage.

- Manufacturers like BMW, Toyota, and Mercedes have a higher spread of mileage (as shown by the longer boxes and whiskers), suggesting a diverse usage of cars from these brands, ranging from lower to very high mileage, whereas brands like Hyundai, Volkswagen, and Audi show a more compact distribution, indicating more consistency in the mileage of their cars.

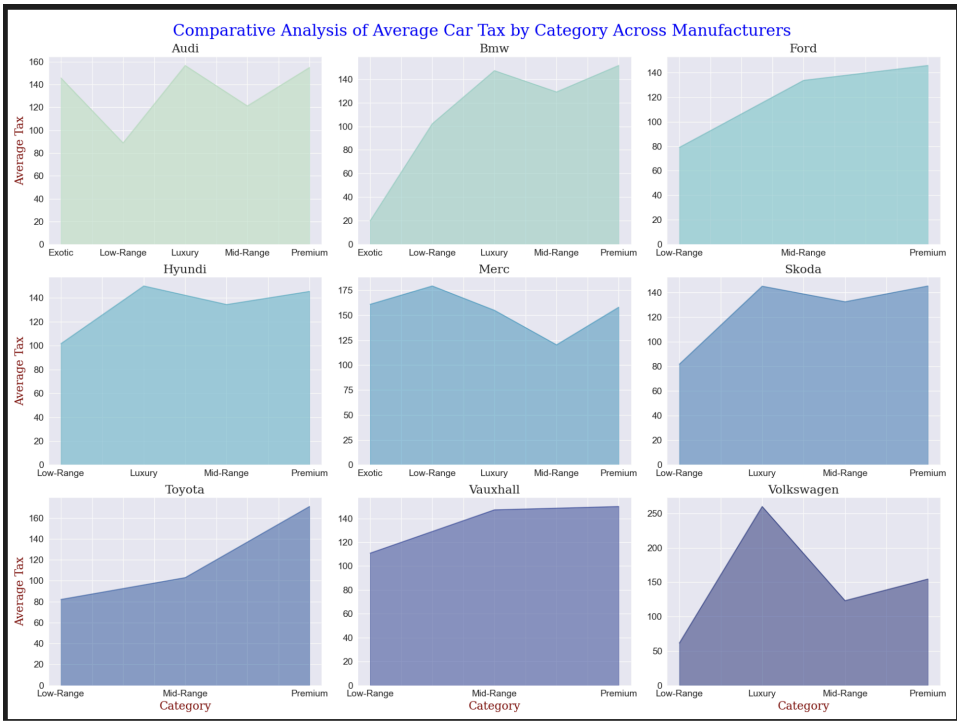
10.14 Boxen Plot: Price Distribution by Fuel Type



- The boxplot shows that diesel and hybrid vehicles tend to have a higher median price compared to petrol and electric vehicles, suggesting these fuel types are associated with more expensive car models or that they typically come with higher-end features.

- Electric vehicles exhibit the narrowest price range and lower median price, which could indicate a more uniform pricing strategy for electric models or reflect incentives and subsidies making them more affordable. Additionally, the fewer outliers for electric vehicles suggest a less varied market or fewer high-priced electric vehicle options.

10.15 Area Plot: Comparative Analysis of Average Car Tax by Category Across Manufacturers

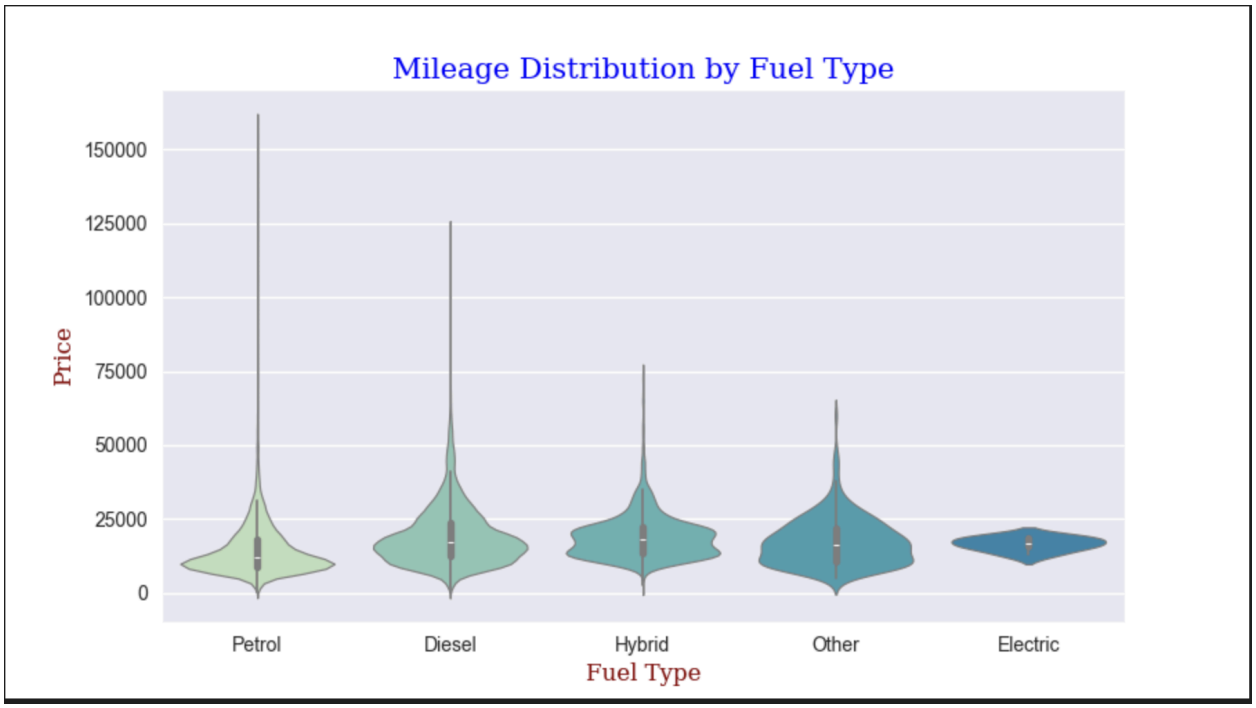


- Volkswagen shows a notable peak in average car tax within the premium category, suggesting that their premium vehicles may have higher emissions or other tax-incurring factors compared to their other categories or the premium offerings from other manufacturers.

- Across most manufacturers, the premium category vehicles tend to have higher average car tax than other categories, indicating that these vehicles are likely in higher tax brackets, possibly due to larger engines or luxury features that may affect their tax liability.

- Ford and Toyota both show an increasing trend in average car tax from low-range to premium vehicles, whereas manufacturers like Audi and BMW show fluctuations across different categories, which could reflect variations in the vehicle specifications that impact tax rates, such as emissions or engine size.

10.16 Violin Plot: Mileage Distribution by Fuel Type

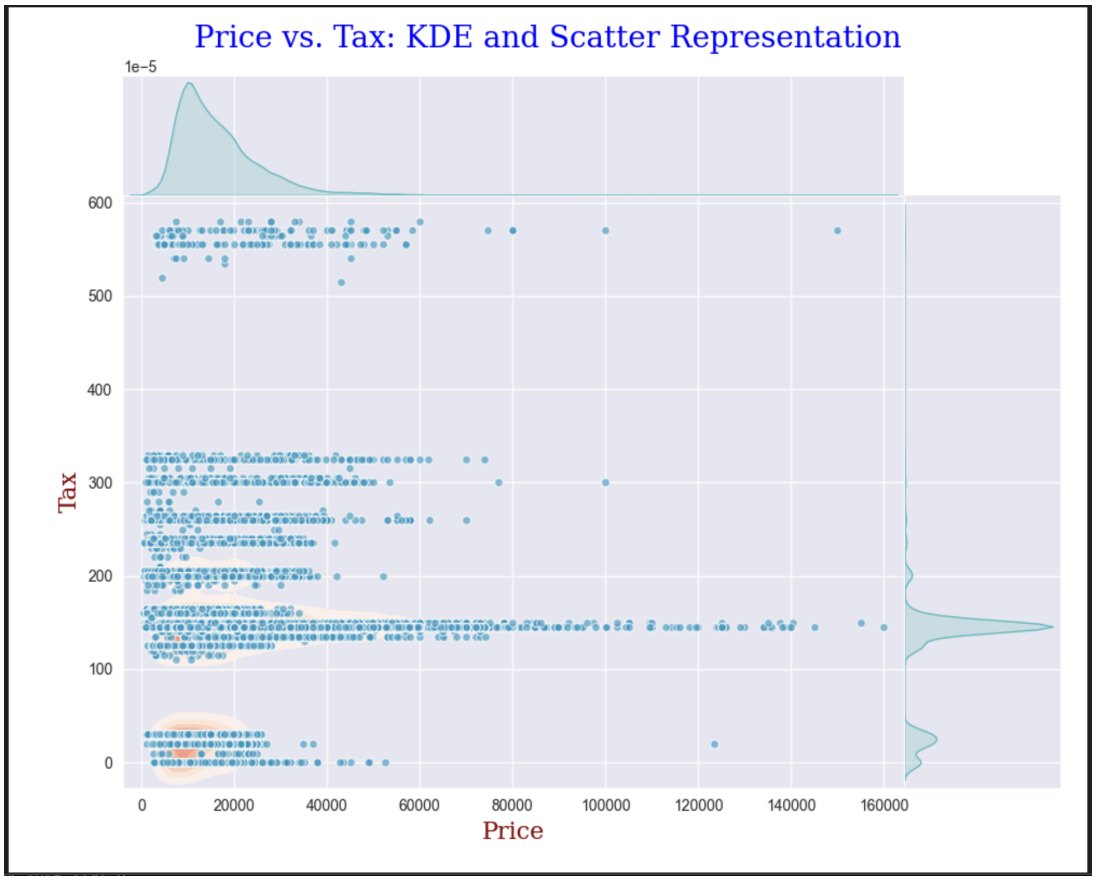


- Diesel vehicles show a wider range of prices than other fuel types, suggesting that there is a diverse market for diesel cars, including both lower-end and higher-end models.

- Electric vehicles have the narrowest distribution in price, which might indicate a more consistent pricing model for electric cars, or that there is less variation in the electric vehicle market in terms of features and specifications that typically affect pricing.

- Hybrid vehicles have a median price that appears to be lower than that of diesel and petrol cars, possibly reflecting the increasing affordability and market competition in hybrid technology. Additionally, the distribution of hybrid prices seems relatively symmetrical, which suggests a balanced market without extreme high or low ends.

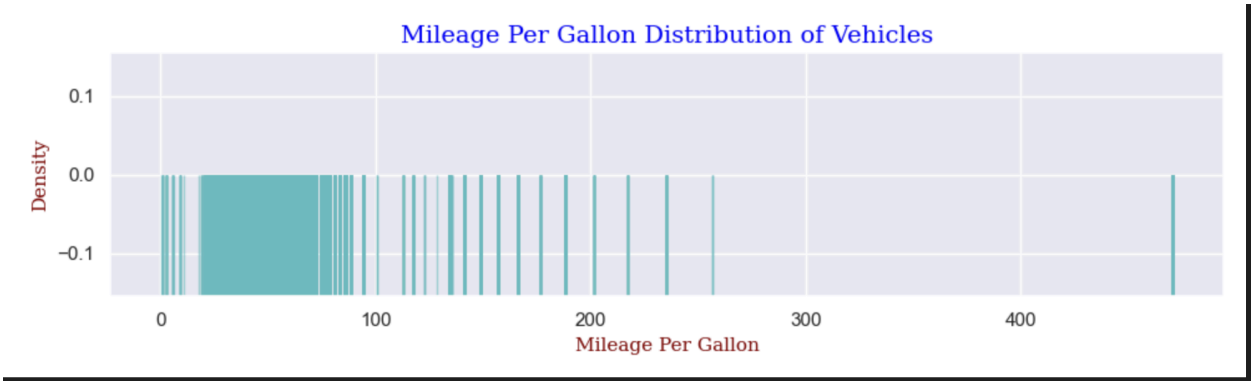
10.17 Joint Plot with KDE and Scatter: Price vs. Tax: KDE and Scatter Representation



- There is a dense concentration of cars with a tax value around 200, regardless of the price, suggesting a common tax band for a large number of vehicles; however, there isn't a clear correlation between price and tax, indicating that car tax may not increase consistently with the car's price.

- The Kernel Density Estimate (KDE) on the right margin shows a significant peak in cars at the lower tax range, while the KDE on the upper margin indicates a wide distribution of car prices, with a peak at the lower end. This pattern may reflect tax policies that do not necessarily penalize higher-priced cars or a prevalence of lower-priced cars within lower tax bands.

10.18 Rug Plot: Mileage Per Gallon Distribution of Vehicles

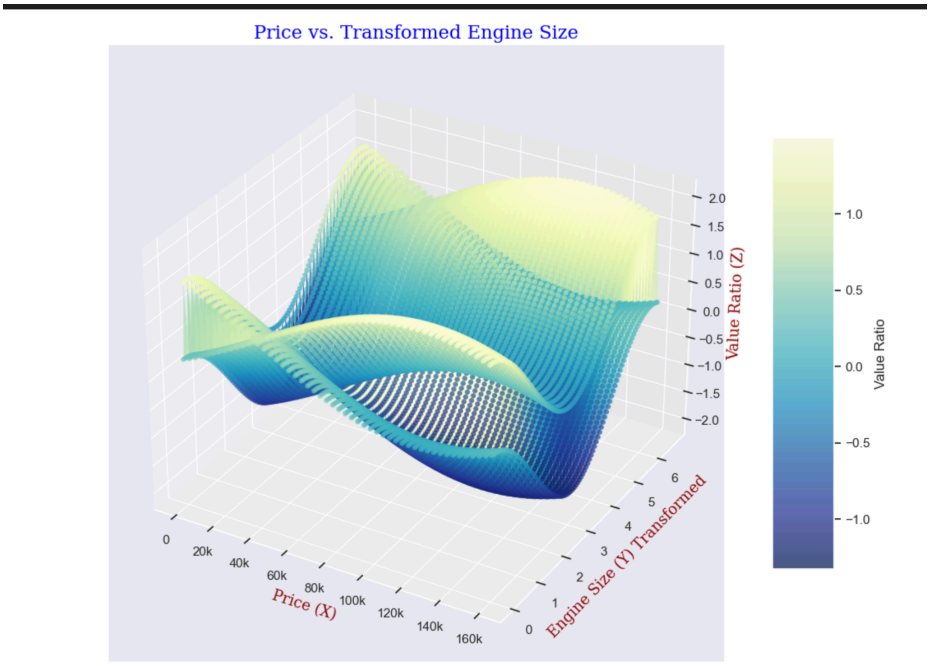


- The distribution of mileage per gallon for vehicles shows distinct peaks at regular intervals, suggesting categorization into specific bands or types of vehicles that have standard fuel efficiency metrics.

- The sharp peak at the higher end of the mileage spectrum indicates a small number of vehicles with exceptionally high miles per gallon, which could be indicative of hybrid or electric vehicles that are capable of achieving greater fuel efficiency compared to traditional internal combustion engines.

- Negative density values, which are not standard in a well-formed distribution, suggest there might be a plotting error or data entry issue with the visualization. Normally, density values should not be negative as they represent the proportion of data points for a given value or range of values.

10.19 3D Plot: Price vs. Transformed Engine Size

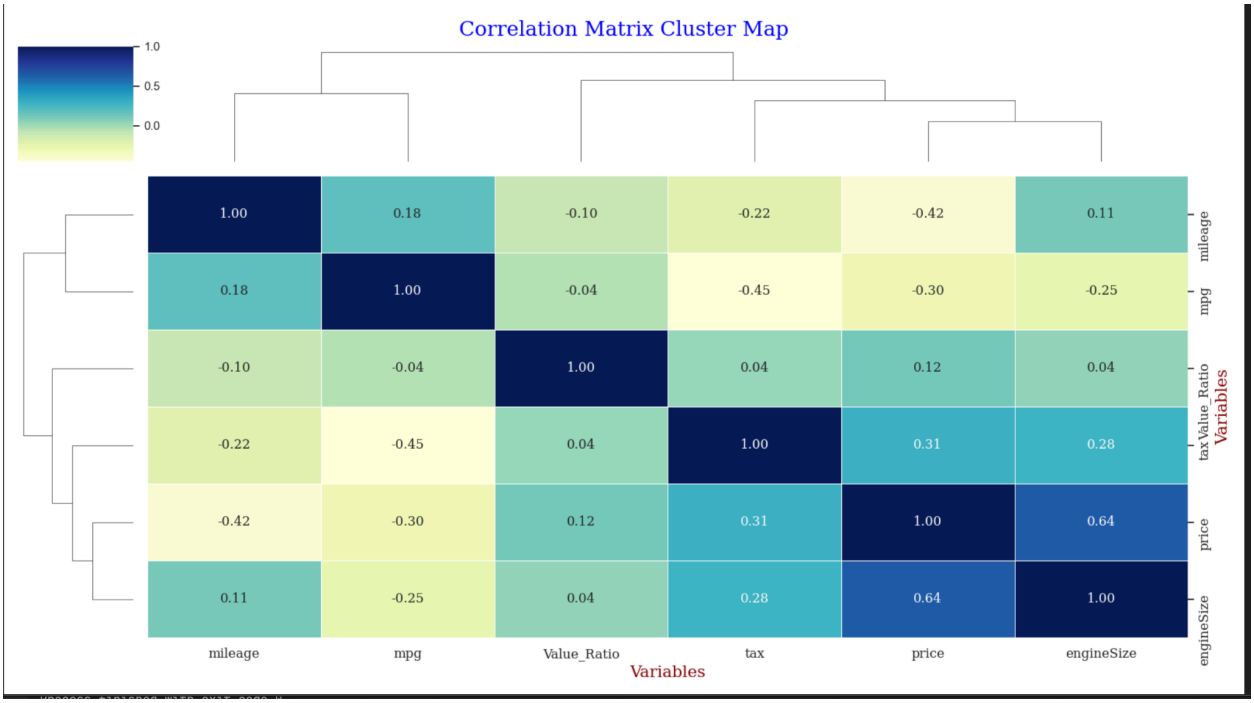


- The 3D surface plot indicates a complex relationship between the price of vehicles and their engine size after a transformation has been applied; there are regions where the value ratio fluctuates significantly, especially at certain levels of transformed engine size, suggesting that engine size alone may not predict price linearly.

- There's a noticeable dip in the value ratio as engine size increases to a certain point, which may suggest that beyond a certain engine size, the increase in price is not as pronounced, potentially indicating diminishing returns on price as engine size increases.

- At the lower end of the engine size scale, there appears to be a peak in the value ratio, indicating that smaller engines may have a proportionally higher price, which could reflect the market for compact or economically designed vehicles that prioritize efficiency over engine power.

10.20 Cluster Map: Correlation Matrix Cluster Map

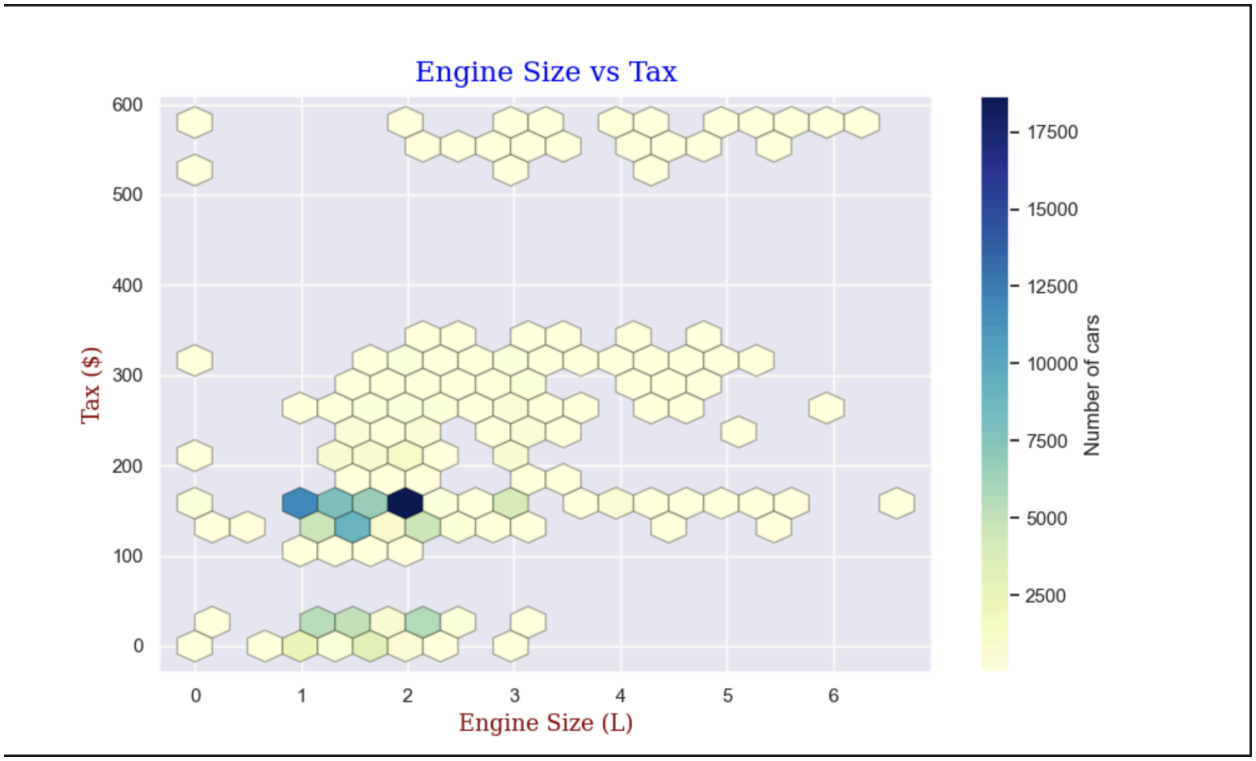


- The correlation matrix's cluster map shows a strong positive correlation (0.64) between tax and price, which suggests that more expensive cars typically have higher tax rates, potentially due to higher emissions or larger engines that are taxed more heavily.

- There is also a moderate positive correlation (0.64) between engine size and price, indicating that cars with larger engines tend to be more expensive, which is a common trend as larger engines are often associated with premium or performance vehicles.

- The heatmap includes dendrograms, which suggest clustering of related variables; however, this specific visualization doesn't provide a strong indication of distinct clusters among the variables, which could mean that there isn't a clear grouping or segmentation in the data based on these factors.

10.21 Hexbin Plot: Engine Size vs Tax



- The hexbin plot suggests that there is a high concentration of cars with engine sizes in the 1-2 liter range, given the darker hexagons representing a higher number of cars, which indicates that smaller engines are more common in the dataset.

- Tax amounts for cars with smaller engines (around 1 liter) tend to be lower, as shown by the clustering of hexagons towards the bottom of the graph; this likely reflects lower tax rates for vehicles with smaller engines.

- There are fewer cars with larger engines (above 2 liters), and these cars tend to incur a higher tax, as can be seen from the hexagons distributed higher on the tax axis. This could indicate that larger engines are less common and possibly subject to higher taxes due to factors like higher emissions.

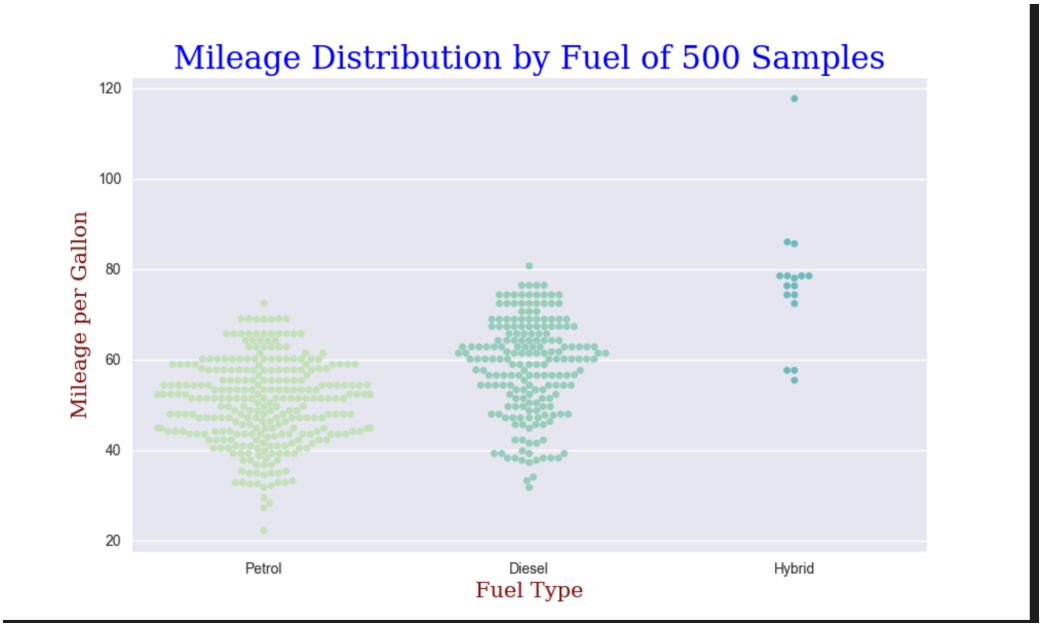
10.22 Strip Plot: Tax Distribution of Top 6 Years with Highest Car Taxes

- There's a visible decrease in tax variation and maximum tax values over the years, from 2015 to 2020, suggesting a possible shift in the types of vehicles being purchased or changes in tax laws affecting vehicle taxes.

- In the year 2015, some cars from manufacturers like BMW and Mercedes had significantly higher taxes, potentially indicating higher emissions or luxury status, but by 2020, the spread and maximum values of taxes for these manufacturers appear to have decreased.

- The scatter density for Audi remains consistently low across the years, indicating a narrower range of tax values for its vehicles, which might reflect a more uniform fleet in terms of emissions or other tax-related characteristics.

10.23 Swarm Plot: Mileage Distribution by Fuel Type for Top 4 Manufacturers



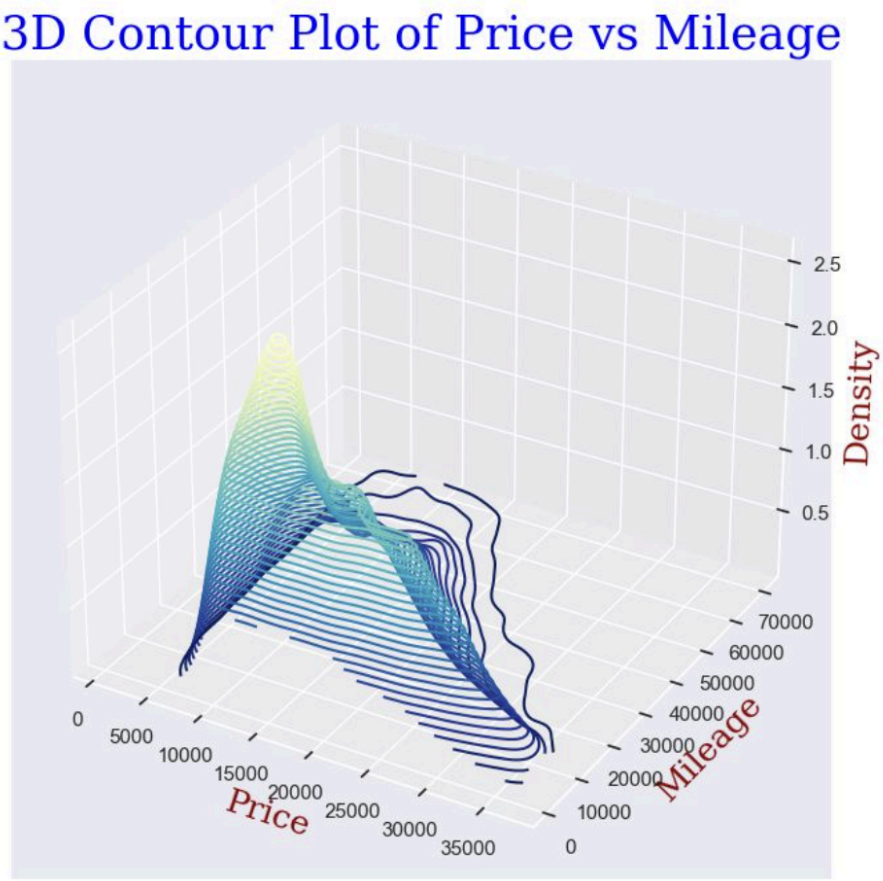
- The swarm plot of a random sample of 500 vehicles reveals that hybrid cars tend to have a significantly higher mileage per gallon than petrol and diesel cars, indicating a trend towards fuel efficiency in hybrid technology.
- To manage the extensive size of the dataset and to enhance the responsiveness of the visualization, only 500 samples were randomly selected for the swarm plot, ensuring a practical and timely analysis while maintaining a snapshot of the overall data trends.

10.24 PrettyTable: Information of DataSet

```
+-----------------------------------------------+
|                     Info                      |
+-----------------------------------------------+
|      <class 'pandas.core.frame.DataFrame'>    |
|      RangeIndex: 97712 entries, 0 to 97711    |
|         Data columns (total 13 columns):      |
|  #   Column               Non-Null Count  Dtype  |
|  ---  ------               --------------  -----  |
|  0   model                97712 non-null  object |
|  1   year                 97712 non-null  object |
|  2   price                97712 non-null  int64  |
|  3   transmission         97712 non-null  object |
|  4   mileage              97712 non-null  int64  |
|  5   fuelType             97712 non-null  object |
|  6   tax                  97712 non-null  int64  |
| 7   mpg                  97712 non-null  float64 |
| 8   engineSize           97712 non-null  float64 |
|  9   Manufacturer         97712 non-null  object |
|  10  Category             97712 non-null  object |
|  11  Manufacturer_Country  97712 non-null  object |
| 12  Value_Ratio          97712 non-null  float64 |
|     dtypes: float64(3), int64(3), object(7)     |
|              memory usage: 9.7+ MB              |
+-----------------------------------------------+
```

- The table displayed in the development environment contains 97,712 records spread across 13 distinct attributes, including categorical data like fuel type and manufacturer, as well as numerical data such as price, mileage, and engine size.
- The table is fully populated with no null values present, indicating a complete dataset that has possibly undergone preprocessing to ensure data quality and readiness for analysis.
- Attributes like 'Manufacturer_Country' and 'Value_Ratio' suggest that the data has been enriched with additional information, potentially for more nuanced analysis like geographic market trends or value assessments.

10.25 3D Contour Plot Price V/S Mileage

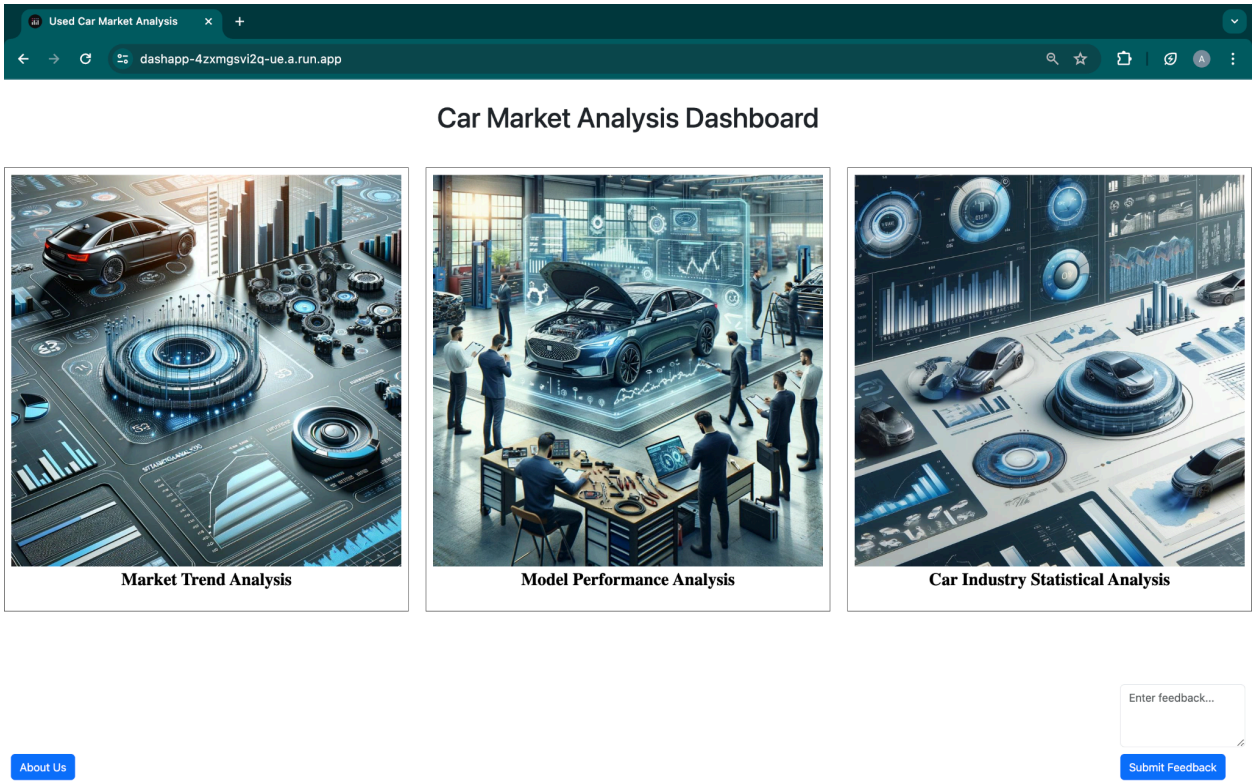


- The 3D contour plot illustrates a distinct peak in vehicle density at lower mileage and price points, indicating a higher concentration of cars with lower usage and cost.
- As mileage increases, the density of cars sharply declines, which could suggest that higher mileage cars are less commonly available or less sought after in the used car market.
- The density of higher-priced cars also diminishes, highlighting a potential buyer preference for more economical options within the dataset's scope.

11. Dashboard

In this section of the report, I will present a comprehensive breakdown of our dashboard's components. Each visual element is designed to distill complex datasets into accessible and actionable insights. Accompanied by screenshots, my explanations will aim to ensure a clear understanding of how each part contributes to the overall analysis.

11.1 Home Page



This dashboard interface serves as a gateway to a rich analysis of the car market. Each image encapsulates a unique segment of the market analysis—clicking on them directs users to detailed explorations of market trends, model performance, and statistical analysis. Hovering over each section reveals tooltips that provide a brief overview of what analysis entails. The user-centric

design includes a 'Submit Feedback' button, encouraging interactive engagement and continuous improvement through user insights.

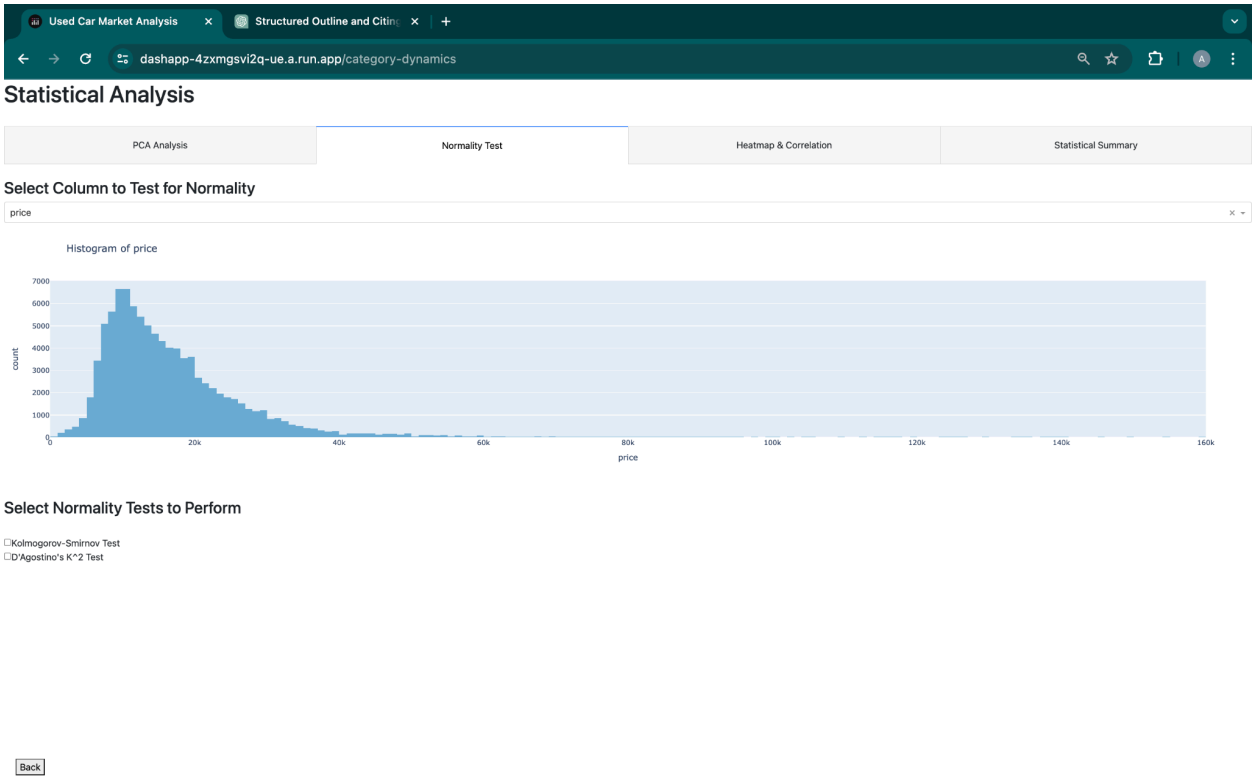
11.2 Market Trend Analysis:

This section of the dashboard offers an interactive experience, where users can tailor the view according to their preferences. By utilizing the radio buttons, you can select a specific manufacturer and instantly observe how the graphs update to reflect the data for that choice. The inclusion of a range slider adds another layer of customization, allowing for a focused analysis on the chosen range of years. These tools work in tandem to ensure that the presented information is both relevant and precise to the user's needs.

11.3 Model Performance Analysis:

The 'Model Performance Analysis' section of our dashboard is a dynamic tool designed for interactive exploration of vehicle data. Users can select a transmission type, such as manual, from a dropdown menu, which will update all corresponding graphs to reflect that selection. Additionally, there is an input field where users can specify the number of top models they wish to analyze, customizing the first plot accordingly. For an even more refined view, click on the 'Outliers ???' button will re-render the box plot without outliers, offering a cleaner representation of the data for focused analysis.

11.4 Car Industry Statistical Analysis



The Statistical Analysis tab of this dashboard is a comprehensive suite for exploratory data analysis, offering diverse analytical approaches such as PCA, normality tests, correlation assessments, and statistical summaries. Within the PCA Analysis tab, users have the option to download PCA results directly, enabling a seamless transition from on-screen analysis to in-depth offline review. The Normality Test tab presents a user-friendly interface, complete with dropdown menus for selecting variables and checkboxes for applying different statistical tests. Moving over to the Heatmap Correlation tab, an interactive slider transitions the view between a heatmap and a scatter plot matrix, offering both overview and detailed exploration of variable relationships. Lastly, the Statistical Summary tab provides a checklist to customize the display of statistical details and multivariate KDE plots, enhancing the understanding of data distribution and underlying patterns.

12. Conclusion

The culmination of this project has not only been about data visualization but also about insight generation and user engagement. The graphical representations crafted in this project serve as a testament to the storytelling power of data, while the interactive dashboard stands as a bridge between raw numbers and strategic insights. Here are the reflective insights drawn from this Project.

12.1 Learnings from Various Created Graphs:

- The array of static plots reinforced the trend of price depreciation in correlation with increased mileage, showcasing the inherent value loss of vehicles over time.
- Manufacturer countries heavily influence average car prices, with German-made cars often positioned at a higher price point, highlighting the premium associated with specific regions.
- The diversity in fuel types and their associated costs revealed a market inclination towards diesel and hybrid vehicles, which could indicate a transition phase in consumer preference towards more sustainable options.

12.2 Utility of Python Dashboard for Users:

- The interactive Python dashboard serves as a crucial tool for users, providing tailored insights into the used car market. Its dynamic nature allows for the interrogation of specific data points based on user-defined parameters such as manufacturer, price range, and year, thereby empowering users with customized data exploration.
- By converting complex datasets into accessible visuals, the dashboard simplifies the decision-making process for potential buyers and sellers in the used car market.

12.3 User-friendliness of the Created App:

- The application's interface is engineered for simplicity and ease of use, with interactive controls such as dropdowns and sliders enabling users to navigate and analyze data

effortlessly, fostering a seamless user experience that accommodates feedback for ongoing enhancements.

12.4 Functionality of the Created App:

- The app's functionality is demonstrated by its capacity to adapt to varied user inputs and provide immediate, relevant visual data. Features like the PCA download capability, adjustable sliders for year range, and the 'Outliers???' button underscore a design philosophy that prioritizes functional depth and user engagement.
- The ability to process and visualize data with such precision and customization points to a high level of technical proficiency in the app's design, making it a potent analytical tool in the hands of its users.

In summary, the various graphs and the dashboard collectively tell a story of the used car market's dynamics, shaped by factors such as depreciation, fuel preferences, and technological advancements. This project not only illustrates data-driven insights but also underscores the significance of interactive tools in making data more comprehensible and actionable.

13. Appendix

The following is a supporting python code that is developed for this project.

13.1 Static Plots Code:

```
# Name: Aneri Patel
# GWID: G40408020

#Project Phase 01:

#%%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as ticker
import scipy.stats as stats
from io import StringIO
import prettytable
from matplotlib.ticker import FuncFormatter
import ssl
ssl._create_default_https_context = ssl._create_unverified_context



sns.set_style("darkgrid")
#%%
pd.set_option('display.max_columns', None)
url                                                                    =
"https://raw.githubusercontent.com/AneriPatel28/Datavizproject/417b8c552dffab76
06ffaf573b94aa26da51d925/CarsData.csv"
data = pd.read_csv(url)

print(data.info())

#%%
print(data['price'].nunique())
print(data['price'].unique())
print(data.shape)
```

```
#%%

print(data['price'].min())
print(data['price'].max())

#%%

def categorize_by_price(price):
    if price <= 10000:
        return 'Low-Range'
    elif price <= 30000:
        return 'Mid-Range'
    elif price <= 60000:
        return 'Premium'
    elif price <= 100000:
        return 'Luxury'
    else:
        return 'Exotic'

data['Category'] = data['price'].apply(categorize_by_price)

data.head()
#%%

data['Category'].value_counts()

#%%
data['Manufacturer'].value_counts()
#%%
manufacturer_country = {
    'ford': 'United States',
    'volkswagen': 'Germany',
    'vauxhall': 'United Kingdom',
    'merc': 'Germany',
    'BMW': 'Germany',
    'Audi': 'Germany',
    'toyota': 'Japan',
    'skoda': 'Czech Republic',
```

```
    'hyundi': 'South Korea'
}
data['Manufacturer_Country'] = data['Manufacturer'].map(manufacturer_country)
print(data['Manufacturer_Country'].value_counts())
#%%

data['Value_Ratio'] = data['price'] / data['mileage']

#%%
print(data.info())
#%%
print(data.head(3))
print(data['engineSize'].value_counts())

#%%

data['year'] = data['year'].astype(object)

print(data.info())
#%%
print(data['year'].value_counts())


buffer = StringIO()
data.info(buf=buffer)
info_str = buffer.getvalue()

# Create a PrettyTable
table = prettytable.PrettyTable()
table.field_names = ["Info"]

# Split the output string into lines for processing
info_lines = info_str.split('\n')

# Add each line as a separate row in the table
for line in info_lines:
    if line:  # avoid adding empty lines
        table.add_row([line.strip()])
```

```
# Print the table
print(table)

#%%



#%%




##############################################################################
##########

#%%
print(data['transmission'].value_counts())
#%%

#### Count Plot
plt.figure(figsize=(10, 7))
# Count plot for the 'model' column
plot=sns.countplot(x='Manufacturer',          data=data,          order          =
data['Manufacturer'].value_counts().index,palette='YlGnBu')

plt.title('Number   of   Cars   by   Manufacturer',   fontdict={'fontsize':   23,
'fontname': 'serif', 'color': 'blue'})
plt.xlabel('Manufacturer',   fontdict={'fontsize':   18,   'fontname':   'serif',
'color': 'darkred'})
plt.ylabel('Number  of  Cars',  fontdict={'fontsize':  18,  'fontname':  'serif',
'color': 'darkred'})
plt.grid(True, color='grey', linestyle='-', linewidth=0.5, alpha=0.25)

plt.xticks(fontname='serif')
plt.yticks(fontname='serif')
plt.show()
```

```
#%%

# Heatmap with cbar

grouped                    =                    data.groupby(['transmission',
'fuelType']).size().reset_index(name='count')

pivot_table    =    grouped.pivot(index='transmission',    columns='fuelType',
values='count')
pivot_table=pivot_table.fillna(0)
def custom_format(x):
   if x >= 1000:
       return '{:.2f} K'.format(x/1000)
   else:
       return str(x)
annot_array = np.vectorize(custom_format)(pivot_table.values)
plt.figure(figsize=(10, 8))
sns.heatmap(pivot_table, annot=annot_array, cmap='YlGnBu', fmt='s', cbar=True,
linewidths=0.03, linecolor='grey')
plt.title('Count of Cars by Transmission and Fuel Type', fontdict={'fontsize':
18, 'fontname': 'serif', 'color': 'blue'})
plt.xlabel('Fuel Type', fontdict={'fontsize': 18, 'fontname': 'serif', 'color':
'darkred'})
plt.ylabel('Transmission Type', fontdict={'fontsize': 18, 'fontname': 'serif',
'color': 'darkred'})
plt.xticks(fontname='serif', fontsize=12)
plt.yticks(fontname='serif', fontsize=12)
plt.show()


#%%



#%%

# BarPlot with subplots:
```

```
data_group = data.groupby(['Category', 'model']).agg({'mileage': 'mean'}).reset_index()

top_models_by_category = data_group.groupby('Category').apply(lambda x: x.nlargest(7, 'mileage')).reset_index(drop=True)

categories = top_models_by_category['Category'].unique()
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
axs = axs.flatten()

bottom_row_indices = [len(axs) - 2, len(axs) - 1]
first_column_indices = [0, len(axs) // 2]

for i, (ax, category) in enumerate(zip(axs, categories)):
    category_data = top_models_by_category[top_models_by_category['Category'] == category]
          sns.barplot(data=category_data, x='model', y='mileage', ax=ax, palette='YlGnBu')
      ax.set_title(category, fontdict={'fontsize': 17, 'fontname': 'serif', 'color': 'blue'})

   if i in bottom_row_indices:
         ax.set_xlabel('Model', fontdict={'fontsize': 15, 'fontname': 'serif', 'color': 'darkred'})
   else:
       ax.set_xlabel('')

   if i in first_column_indices:
          ax.set_ylabel('Avg. Mileage', fontdict={'fontsize': 14, 'fontname': 'serif', 'color': 'darkred'})
   else:
       ax.set_ylabel('')

   ax.set_xticklabels(ax.get_xticklabels(), fontname='serif', fontsize=13)
   ax.set_yticklabels(ax.get_yticklabels(), fontname='serif', fontsize=12)
   ax.grid(True, which='both', linestyle='--', linewidth=0.5)
```

```
plt.tight_layout(rect=[0, 0, 1, 0.93])
fig.suptitle('Top 7 Models by Mileage Across Different Categories',
fontsize=24, fontname='serif', color='blue',
            va='bottom', y=0.93)
plt.show()

#%%

# Pie Charts
country_counts = data['Manufacturer_Country'].value_counts()
print(country_counts)

num_shades = 8
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()
print(colors)
plt.figure(figsize=(8,7))
pies, texts, autotexts = plt.pie(country_counts, labels=country_counts.index,
autopct='%1.2f%%', startangle=140, colors=colors)
plt.title('Distribution of Cars by Manufacturer Country', fontdict={'fontsize':
20, 'fontname': 'serif', 'color': 'blue'})
for text in texts:
   text.set_color('darkred')
   text.set_fontsize(12)
for pie in pies:
   pie.set_linewidth(0)

plt.legend(title='Country', title_fontsize='10', fontsize='10', loc='lower
left', bbox_to_anchor=(0.87, 0.01))
plt.show()


#%%

# Histplot with KDE
plt.figure(figsize=(10, 6))
sns.histplot(data['price'], kde=True, color='#97d6b9', alpha=0.6, bins=40,
line_kws={'linewidth': 1})
```

```
plt.title('Distribution of Prices for Cars', fontdict={'fontsize': 17,
'fontname': 'serif', 'color': 'blue'})
plt.xlabel('Price', fontdict={'fontsize': 14, 'fontname': 'serif', 'color':
'darkred'})
plt.ylabel('Frequency', fontdict={'fontsize': 14, 'fontname': 'serif', 'color':
'darkred'})
plt.gca().xaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos:
'{:,.0f}.00 k'.format(x/1000)))
plt.gca().yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos:
'{:,.0f}.00 k'.format(x/1000)))
plt.show()

#%%

#QQ-plot

fig = plt.figure(figsize=(7,5))
ax = fig.add_subplot(111)
res = stats.probplot(data['mpg'], dist="norm", plot=ax, rvalue=True)
ax.get_lines()[0].set_markerfacecolor('none')  # Makes the markers hollow
ax.get_lines()[0].set_markeredgecolor('#97d6b9')  # Sets the color of the ring
ax.get_lines()[0].set_marker('o')  # Sets the marker to a circle
ax.get_lines()[1].set_color('#1f80b8')
ax.get_lines()[1].set_linestyle('-')
ax.set_title('Normal QQ-Plot of Car Miles Per Gallon.',fontdict={'fontsize':
17, 'fontname': 'serif', 'color': 'blue'})
ax.set_xlabel('Theoretical Quantiles', fontdict={'fontsize': 14, 'fontname':
'serif', 'color': 'darkred'})
ax.set_ylabel('Quantiles of MPG', fontdict={'fontsize': 14, 'fontname':
'serif', 'color': 'darkred'})
plt.show()


#%%

avg_tax = data.groupby(['Manufacturer',
'Category'])['tax'].mean().reset_index(name='Average Tax')
```

```
#%%

# Area Plot:

num_shades = 12
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()

manufacturers = avg_tax['Manufacturer'].unique()
fig, axs = plt.subplots(3, 3, figsize=(20, 15), constrained_layout=True)
fig.subplots_adjust(hspace=0.4, wspace=0.4)

for i, ax in enumerate(axs.flat):
   if i < len(manufacturers):
       manufacturer = manufacturers[i]
       manufacturer_data = avg_tax[avg_tax['Manufacturer'] == manufacturer]
       single_color_palette = [colors[i + 3]]
                manufacturer_data  =  manufacturer_data.pivot(index='Category',
columns='Manufacturer', values='Average Tax')
       manufacturer_data.plot(kind='area', stacked=False, ax=ax,
                              color=single_color_palette, legend=False)
       if i // 3 == 2:
                ax.set_xlabel('Category',fontdict={'fontsize': 17, 'fontname':
'serif', 'color': 'darkred'})
       else:
           ax.set_xlabel('')
       if i % 3 == 0:
              ax.set_ylabel('Average Tax',fontdict={'fontsize': 17, 'fontname':
'serif', 'color': 'darkred'})
       else:
           ax.set_ylabel('')

            ax.set_title(manufacturer.capitalize(),  fontdict={'fontsize': 18,
'fontname': 'serif'})
       ax.tick_params(axis='x', labelsize=14)
       ax.tick_params(axis='y', labelsize=14)
   else:
```

```
        break

for j in range(i + 1, 9):
    fig.delaxes(axs.flat[j])
fig.suptitle('Comparative Analysis of Average Car Tax by Category Across
Manufacturers', fontsize=24, fontname='serif',
             color='blue',y=0.98)
plt.tight_layout(rect=[0, 0, 1, 0.993])
plt.show()


#%%
num_shades = 27
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()

#%%
# Box plot
plt.figure(figsize=(8,5))
ax = sns.boxplot(x='Manufacturer', y='mileage', data=data, palette=colors[5:],
fliersize=4, showfliers=True, width=0.7, linewidth=1)
ax.set_title('Mileage Distribution by Manufacturer', fontsize=22,
fontname='serif', color='blue')
ax.set_xlabel('Manufacturer', fontdict={'fontsize': 17, 'fontname': 'serif',
'color': 'darkred'})
ax.set_ylabel('Mileage', fontdict={'fontsize': 17, 'fontname': 'serif',
'color': 'darkred'})
ax.set_xticklabels([tick.get_text().capitalize() for tick in
ax.get_xticklabels()])

plt.tight_layout()
plt.show()


#%%
#boxen plot
num_shades = 27
palette = sns.color_palette("YlGnBu", num_shades)
```

```
colors = palette.as_hex()
plt.figure(figsize=(8,5))
ax = sns.boxenplot(x='fuelType', y='price', data=data, palette=colors[5:])
ax.set_title('Price Distribution by Fuel Type', fontsize=22, fontname='serif',
color='blue')
ax.set_xlabel('Fuel Type', fontdict={'fontsize': 17, 'fontname': 'serif',
'color': 'darkred'})
ax.set_ylabel('Price', fontdict={'fontsize': 17, 'fontname': 'serif', 'color':
'darkred'})
ax.set_xticklabels([tick.get_text().capitalize() for tick in
ax.get_xticklabels()])

plt.tight_layout()
plt.show()


#%%









#%%

#%%
def thousands_formatter(x, pos):
   return f'{int(x/1000)}k' if x >= 1000 else str(int(x))

#%%

manufacturer_countries = list(data['Manufacturer_Country'].unique())
```

```
import matplotlib.ticker as mticker

def slice_by_manufacturer_country(df, country):
   return df[df["Manufacturer_Country"] == country]




num_shades = 16
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()[4:]

fig, axs = plt.subplots(2, 3, figsize=(19, 13))

for idx, country in enumerate(manufacturer_countries):
   ax = axs[idx // 3, idx % 3]
   country_data = slice_by_manufacturer_country(data, country)
   sns.regplot(x='mileage', y='price', data=country_data, ax=ax,
                     scatter_kws={'s': 30, 'alpha': 0.6, 'edgecolor': 'white',
'color': colors[idx]},
               line_kws={'color': colors[idx]})
   ax.set_ylim(bottom=0)
   ax.set_title(f'{country}', fontsize=15, fontname='serif', )

   ax.xaxis.set_major_formatter(mticker.FuncFormatter(thousands_formatter))
   ax.yaxis.set_major_formatter(mticker.FuncFormatter(thousands_formatter))
   for label in ax.get_xticklabels():
       label.set_fontsize(14)
       label.set_fontname('serif')
   for label in ax.get_yticklabels():
       label.set_fontsize(14)
       label.set_fontname('serif')

   if idx // 3 == 1:
        ax.set_xlabel('Mileage', fontdict={'fontsize': 16, 'fontname': 'serif',
'color': 'darkred'})
   else:
       ax.set_xlabel('')
```

```
   if idx % 3 == 0:
          ax.set_ylabel('Price', fontdict={'fontsize': 16, 'fontname': 'serif',
'color': 'darkred'})
   else:
       ax.set_ylabel('')

fig.suptitle('Comparative Price vs. Mileage across Different Countries',
fontsize=27, fontname='serif', color='blue')
plt.tight_layout(rect=[0, 0, 1, 0.98])
plt.show()
#%%




#######


#%%


#%%



colors= ['#c4e8b4', '#97d6b9', '#6dc6be', '#5db5ad']
top_years = data['year'].value_counts().nlargest(10).index
top_years = data['year'].value_counts().nlargest(10).index.sort_values()
df_top_years = data[data['year'].isin(top_years)]

plt.figure(figsize=(13,6))

ax= sns.lineplot(data=df_top_years, x='year', y='mileage', hue='transmission',
estimator='mean',               ci=None,               marker='o',markersize=8,
markeredgecolor='grey',palette=colors,linewidth=2)

ax.yaxis.set_major_formatter(mticker.FuncFormatter(thousands_formatter))
```

```
for label in ax.get_xticklabels():
   label.set_fontsize(12)
   label.set_fontname('serif')

for label in ax.get_yticklabels():
   label.set_fontsize(12)
   label.set_fontname('serif')
plt.xticks(top_years)
plt.xlabel('Year',fontdict={'fontsize':  18,  'fontname':  'serif',  'color':
'darkred'})
plt.ylabel('Average  Mileage',fontdict={'fontsize':  18,  'fontname':  'serif',
'color': 'darkred'})
plt.title('Average Mileage Across Top 10 Years by Sales Volume', fontsize=21,
fontname='serif', color='blue')
plt.legend(title='Transmission Type')
plt.grid(True)
plt.tight_layout()
plt.show()



#%%

import random

num_shades = 50
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()
selected_colors = random.sample(colors, 4)

print(selected_colors)

#%%
```

```
#%%

# Stack barplot

num_shades = 10
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()[2:]

fig, ax = plt.subplots(figsize=(10, 6))  # Reduced width from 13 to 10
average_price_by_country_fuel = data.groupby(['Manufacturer_Country',
'fuelType'])['price'].mean().unstack(fill_value=0)
average_price_by_country_fuel.plot(kind='bar', stacked=True, color=colors,
ax=ax, width=0.65)
ax.set_title('Average Price by Manufacturer Country and Fuel Type',
fontsize=20, fontname='serif', color='blue')
ax.set_xlabel('Manufacturer Country', fontdict={'fontsize': 18, 'fontname':
'serif', 'color': 'darkred'})
ax.set_ylabel('Average Price', fontdict={'fontsize': 18, 'fontname': 'serif',
'color': 'darkred'})
ax.yaxis.set_major_formatter(mticker.FuncFormatter(thousands_formatter))
ax.set_xticklabels(ax.get_xticklabels(), rotation=360)
ax.legend(title='Fuel Type')
for label in ax.get_xticklabels():
   label.set_fontsize(10)
   label.set_fontname('serif')
for label in ax.get_yticklabels():
   label.set_fontsize(10)
   label.set_fontname('serif')
plt.tight_layout()
plt.show()




#%%
```

```
grouped_data = data.groupby(['Manufacturer_Country', 'transmission'])['price'].mean().reset_index()




#%%


# Group barplot

plt.figure(figsize=(12, 7))
num_shades = 12
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()[2:]

sns.barplot(x='Manufacturer_Country', y='price', hue='transmission', data=grouped_data, palette=colors)
plt.title('Average Car Price by Transmission Type Across Manufacturer Countries',fontsize=20, fontname='serif', color='blue')
plt.xlabel('Manufacturer Country',fontdict={'fontsize': 18, 'fontname': 'serif', 'color': 'darkred'})
plt.ylabel('Average Price',fontdict={'fontsize': 18, 'fontname': 'serif', 'color': 'darkred'})
plt.legend(title='Transmission Type', loc='upper right', title_fontsize='13', fontsize='10')
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.tight_layout()
plt.show()


#%%

#%%

# Violin plot draft 01:
```

```
plt.figure(figsize=(9,5))
num_shades = 10
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()[2:]
sns.violinplot(x='fuelType',                                    y='price',
data=data,linewidth=0.85,palette=colors,edgecolor='grey')
plt.title('Mileage  Distribution  by  Fuel  Type',fontsize=15,  fontname='serif',
color='blue')
plt.xlabel('Fuel  Type',fontdict={'fontsize':  12,  'fontname':  'serif',  'color':
'darkred'})
plt.ylabel('Price',fontdict={'fontsize':   12,   'fontname':   'serif',   'color':
'darkred'})
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()

#%%

data.head()
#%%

# Joint Plot with KDE:

plt.figure(figsize=(8,9))

num_shades = 12
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()[2:]

joint_kde_scatter  =  sns.jointplot(x='price',  y='tax',  data=data,  kind='kde',
fill=True,color=colors[3],space=0, cmap='Reds')
joint_kde_scatter.plot_joint(sns.scatterplot,      s=30,      edgecolor='white',
linewidth=1, alpha=0.6,color=colors[5])

joint_kde_scatter.set_axis_labels('Price',   'Tax',   fontdict={'fontsize':   16,
'fontname': 'serif', 'color': 'darkred'})
```

```
joint_kde_scatter.fig.suptitle('Price vs. Tax: KDE and Scatter Representation',
fontsize=20, fontname='serif', color='blue')

joint_kde_scatter.fig.subplots_adjust(top=0.92)

joint_kde_scatter.fig.set_size_inches(10, 8)

plt.show()



#%%

sns.set(style="whitegrid")

#%%

# KDE Plot

num_shades = 10
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()[2:]

plt.figure(figsize=(10, 6))
sns.kdeplot(data=data, x='price', hue='Category', fill=True, common_norm=False,
palette=colors, alpha=0.6, linewidth =1)
plt.title('Price Distribution by Category', fontsize=16, fontname='serif',
color='blue')
plt.xlabel('Price', fontdict={'fontsize': 12, 'fontname': 'serif', 'color':
'darkred'})
plt.ylabel('Density', fontdict={'fontsize': 12, 'fontname': 'serif', 'color':
'darkred'})
plt.grid(True,alpha=0.6)
plt.show()

#%%

sns.set(style="darkgrid")
```

```
# Cluster map:

numeric_data = data.select_dtypes(include=['number'])

corr = numeric_data.corr()
cluster_map    =    sns.clustermap(corr,    cmap="YlGnBu",    figsize=(16,    9),
linewidths=.5, annot=True, fmt=".2f",
                           annot_kws={'size': 12, 'fontname': 'serif'},
                           dendrogram_ratio=(.1, .2))

cluster_map.fig.suptitle('Correlation   Matrix   Cluster   Map',   fontsize=19,
fontname='serif', color='blue', va='center',y=0.95)

cluster_map.ax_heatmap.set_xlabel('Variables',     fontdict={'fontsize':     15,
'fontname': 'serif', 'color': 'darkred'})
cluster_map.ax_heatmap.set_ylabel('Variables',     fontdict={'fontsize':     15,
'fontname': 'serif', 'color': 'darkred'})

plt.setp(cluster_map.ax_heatmap.get_xticklabels(),                  fontsize=12,
fontname='serif')
plt.setp(cluster_map.ax_heatmap.get_yticklabels(),                  fontsize=12,
fontname='serif')

plt.tight_layout()
plt.show()


#%%

# Hexbin:


sns.set(style="darkgrid")
plt.figure(figsize=(10, 6))
edge_color_with_alpha = (0, 0, 0, 0.3)  # Black with 50% transparency
plt.hexbin(data['engineSize'],    data['tax'],    gridsize=20,    cmap='YlGnBu',
mincnt=1,
          edgecolors=edge_color_with_alpha)
```

```
plt.colorbar(label='Number of cars')
plt.xlabel('Engine Size (L)',fontdict={'fontsize': 13, 'fontname': 'serif',
'color': 'darkred'})
plt.ylabel('Tax ($)',fontdict={'fontsize': 13, 'fontname': 'serif', 'color':
'darkred'})
plt.grid(True)
plt.title('Engine Size vs Tax',fontsize=16, fontname='serif', color='blue',
va='center',y =1.02)
plt.show()


#%%

# Strip plot

num_shades = 12
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()[:]

total_tax_by_year = data.groupby('year')['tax'].sum().reset_index()
#%%
#%%
top_years_by_tax = total_tax_by_year.sort_values('tax',
ascending=False).head(6)['year']

filtered_data = data[data['year'].isin(top_years_by_tax)]

manufacturer_order = ['Ford', 'Vauxhall', 'Skoda', 'Hyundi', 'Toyota',
'Mercedes', 'Bmw', 'Volkswagen', 'Audi']
#%%

num_shades = 7
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()[1:]



#%%
```

```
fig, axs = plt.subplots(2, 3, figsize=(20, 12))
axs = axs.flatten()

for i, year in enumerate(sorted(top_years_by_tax)):
   year_data = filtered_data[filtered_data['year'] == year]

       sns.stripplot(x='Manufacturer',  y='tax',  data=year_data  ,jitter=True,
ax=axs[i],color='green',alpha=0.6)
   axs[i].set_title(f' {year}',fontsize=16, fontname='serif', color='navy')
   axs[i].set_xticklabels([text.capitalize() for text in manufacturer_order])

   if i in [3, 4,5]:
         axs[i].set_xlabel('Manufacturer',fontdict={'fontsize': 15, 'fontname':
'serif', 'color': 'darkred'})
   else:
       axs[i].set_xlabel('')

   if i in [0, 3]:
           axs[i].set_ylabel('Tax  ($)',fontdict={'fontsize':  15,  'fontname':
'serif', 'color': 'darkred'})
   else:
       axs[i].set_ylabel('')

fig.suptitle('Tax  Distribution  of  Top  6  Years  with  Highest  Car  Taxes',
fontsize=22, fontname='serif', color='blue')

plt.tight_layout(rect=[0, 0.03, 1, 0.95])

plt.tight_layout()
plt.show()



#%%

# Swarm plot:
```

```
num_shades = 10
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()[2:]
top_manufacturers = data['Manufacturer'].value_counts().index.tolist()

# Filter data for top manufacturers
top_manufacturers_data = data[data['Manufacturer'].isin(top_manufacturers)]


sampled_data = top_manufacturers_data.sample(n=500, random_state=42)

# Create the swarm plot
plt.figure(figsize=(10, 6))
swarm_plot = sns.swarmplot(
   x='fuelType',
   y='mpg',
   data=sampled_data,
   order=sampled_data['fuelType'].value_counts().index,
   palette=colors)



# Set title and labels with formatting
swarm_plot.set_title('Mileage Distribution by Fuel of 500 Samples',
fontsize=22, fontname='serif', color='blue')
swarm_plot.set_xlabel('Fuel Type', fontdict={'fontsize': 15, 'fontname':
'serif', 'color': 'darkred'})
swarm_plot.set_ylabel('Mileage per Gallon', fontdict={'fontsize': 15,
'fontname': 'serif', 'color': 'darkred'})

plt.show()

#%%

# Rugplot

num_shades = 10
palette = sns.color_palette("YlGnBu", num_shades)
```

```
colors = palette.as_hex()[2:]



plt.figure(figsize=(10, 3))
sns.rugplot(data=data, x='mpg',height=0.5, color=colors[2],alpha=0.6)
plt.title('Mileage Per Gallon Distribution of Vehicles',fontsize=14,
fontname='serif', color='blue')
plt.xlabel('Mileage Per Gallon',fontdict={'fontsize': 11, 'fontname': 'serif',
'color': 'darkred'})
plt.ylabel('Density',fontdict={'fontsize': 11, 'fontname': 'serif', 'color':
'darkred'})
plt.tight_layout()
plt.show()



#%%



# 3D plot
#%%
#%%

x = np.linspace(data['price'].min(), data['price'].max(), 100)  # Assuming
'price' ranges from 1 to 10 for demonstration
y = np.linspace(data['engineSize'].min(), data['engineSize'].max(), 100)  #
Assuming 'engineSize' ranges from 1 to 3 for demonstration
X, Y = np.meshgrid(x, y)
def some_function(X, Y):
   return np.sin(X) + np.cos(Y)

Z = some_function(X, Y)



fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(X, Y, Z, cmap='YlGnBu_r', edgecolor='none', alpha=0.7)

k=ax.scatter(X, Y, Z, c=Z, cmap='YlGnBu_r', edgecolor='none', alpha=0.7)
```

```
ax.set_xlabel('Price    (X)',fontdict={'fontsize':    13,    'fontname':    'serif',
'color': 'darkred'})
ax.set_ylabel('Engine    Size    (Y)    Transformed',fontdict={'fontsize':    13,
'fontname': 'serif', 'color': 'darkred'})
ax.set_zlabel('Value Ratio (Z)',fontdict={'fontsize': 13, 'fontname': 'serif',
'color': 'darkred'})
ax.set_title('Price vs. Transformed Engine Size',fontsize=16, fontname='serif',
color='blue')
ax.xaxis.set_major_formatter(FuncFormatter(thousands_formatter))
fig.colorbar(surf, shrink=0.7, aspect=7, label='Value Ratio')
plt.tight_layout()
plt.show()


#%%



#%%

# Distplot




fig, axs = plt.subplots(2, 2, figsize=(14, 10))
fig.subplots_adjust(hspace=0.4, wspace=0.4)


transmission_types = ['Manual', 'Semi-Auto', 'Automatic', 'Other']
num_shades = 10
palette = sns.color_palette("YlGnBu", num_shades)
colors = palette.as_hex()[2:]
i=0
for idx, (ax, transmission) in enumerate(zip(axs.flat, transmission_types)):
   subset = data[data['transmission'] == transmission]
```

```
    sns.distplot(subset['mileage'],  bins=30, ax=ax, color=colors[i])
            ax.set_title(f'Transmission:    {transmission}',    fontsize=13.5,
fontname='serif', color='navy')
    ax.xaxis.set_major_formatter(FuncFormatter(thousands_formatter))
    ax.yaxis.set_major_formatter(FuncFormatter(thousands_formatter))

    if idx < 2:
        ax.set_xlabel('')
    else:
         ax.set_xlabel('Mileage', fontdict={'fontsize': 13, 'fontname': 'serif',
'color': 'darkred'})

    if idx % 2 == 0:
         ax.set_ylabel('Density', fontdict={'fontsize': 13, 'fontname': 'serif',
'color': 'darkred'})
    else:
        ax.set_ylabel('')

    i += 1
fig.suptitle('Mileage  Distribution  by  Transmission  Type',  fontsize=20,
fontname='serif', color='blue')
plt.tight_layout()
plt.show()

#%%
# Outlier Detection and removal:
def remove_outliers_iqr(data):
    for column in data.select_dtypes(include=['number']).columns:
        Q1 = data[column].quantile(0.25)
        Q3 = data[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - (1.5 * IQR)
        upper_bound = Q3 + (1.5 * IQR)

        print(f"Column: {column}")
         print(f"Values below {abs(lower_bound):.2f} and above {upper_bound:.2f}
are considered outliers for this column.\n")
```

```
            data = data[(data[column] >= lower_bound) & (data[column] <=
upper_bound)]

    return data
data=remove_outliers_iqr(data)

print(data.info())



############
# PAIRPLOT #
############

numeric_columns = data.select_dtypes(include=['number'])
#%%
plt.figure(figsize=(15, 10))
pair_plot = sns.pairplot(numeric_columns)
plt.subplots_adjust(top=0.95)
plt.suptitle('Pair Plot of Numeric Columns', fontsize=25, fontname='serif',
color='blue', y=0.98)
plt.show()



#%%



#  Contour plot:
from scipy.stats import gaussian_kde



# Select the 'price' and 'mileage' columns for the contour plot
x = data['price']
y = data['mileage']

# Generate a meshgrid for the contour plot
x_range = np.linspace(x.min(), x.max(), 100)
y_range = np.linspace(y.min(), y.max(), 100)
X, Y = np.meshgrid(x_range, y_range)
```

```
# Perform a kernel density estimate on the data
kde = gaussian_kde([x, y])
Z = kde(np.vstack([X.ravel(), Y.ravel()]))

# Plot the contour in 3D
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.contour3D(X, Y, Z.reshape(X.shape), 50, cmap='YlGnBu_r')
ax.set_xlabel('Price', fontsize=18, fontname='serif', color='darkred')
ax.set_ylabel('Mileage', fontsize=18, fontname='serif', color='darkred')
ax.set_zlabel('Density', fontsize=18, fontname='serif', color='darkred')
ax.set_title('3D Contour Plot of Price vs Mileage', fontsize=25,
fontname='serif', color='blue')
plt.show()
```

13.2 Interactive Dashboard Code:

```
import dash
from dash import dcc, html, callback
from dash.dependencies import Input, Output, State
import plotly.express as px
import dash_bootstrap_components as dbc
from dash.dash_table.Format import Group
import pandas as pd
import plotly.graph_objs as go
import numpy as np
import plotly.figure_factory as ff
import scipy.stats as stats
import statsmodels.api as sm
import dash_bootstrap_components as dbc
import dash_table
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP])

app.title = "Used Car Market Analysis"
url                                                                    =
"https://raw.githubusercontent.com/AneriPatel28/Datavizproject/417b8c552dffab76
06ffaf573b94aa26da51d925/CarsData.csv"
data = pd.read_csv(url)

def categorize_by_price(price):
    if price <= 10000:
        return 'Low-Range'
    elif price <= 30000:
        return 'Mid-Range'
    elif price <= 60000:
        return 'Premium'
    elif price <= 100000:
        return 'Luxury'
    else:
        return 'Exotic'

data['Category'] = data['price'].apply(categorize_by_price)

manufacturer_country = {
    'ford': 'United States',
    'volkswagen': 'Germany',
    'vauxhall': 'United Kingdom',
    'merc': 'Germany',
    'BMW': 'Germany',
    'Audi': 'Germany',
    'toyota': 'Japan',
    'skoda': 'Czech Republic',
    'hyundi': 'South Korea'
}
data['Manufacturer_Country'] = data['Manufacturer'].map(manufacturer_country)

data['Value_Ratio'] = data['price'] / data['mileage']
```

```
data['year'] = data['year'].astype(object)
def get_market_trends_layout():
   return html.Div([
       html.H1("Market Trend Analysis"),
       dcc.Loading(
           id="loading-market-trends",
           type="default",
           children=[
               html.Div([
                   dcc.RadioItems(
                       id='manufacturer-radio',
                       options=[
                           {'label': 'Ford', 'value': 'ford'},
                           {'label': 'Volkswagen', 'value': 'volkswagen'},
                           {'label': 'Vauxhall', 'value': 'vauxhall'},
                           {'label': 'Mercedes', 'value': 'merc'},
                           {'label': 'BMW', 'value': 'BMW'},
                           {'label': 'Audi', 'value': 'Audi'},
                           {'label': 'Toyota', 'value': 'toyota'},
                           {'label': 'Skoda', 'value': 'skoda'},
                           {'label': 'Hyundai', 'value': 'hyundi'}
                       ],
                       value='toyota',
                          labelStyle={'display': 'inline-block', 'margin-right':
'20px'},
                       style={
                           'width': '100%',
                           'display': 'inline-flex',
                           'justify-content': 'space-around',
                           'border': '1px solid #ccc',
                           'padding': '10px',
                           'margin-bottom': '10px',
                           'box-shadow': '0 2px 4px rgba(0,0,0,0.1)'
                       }
                   )
               ], style={'order': '1'}),
```

```
                        html.Div(id='graph-container', style={'order': '2',
'margin-bottom': '30px'}),
                html.Div([
                    dcc.RangeSlider(
                        id='year-slider',
                        min=2000,
                        max=2019,
                        value=[2000, 2019],
                             marks={str(year): str(year) for year in range(2000,
2020)},
                        step=1
                    )
                ], style={'order': '3', 'margin-bottom': '10px'}),
                html.Div([
                                html.Button("Back", id="back-button", n_clicks=0,
style={'width': '100px'})
                    ], style={'display': 'flex', 'justify-content': 'flex-start',
'padding': '20px', 'order': '4'}),
            ]
        )
    ], style={'display': 'flex', 'flex-direction': 'column', 'padding': '20px'})



def generate_heatmap(heatmap_data, selected_manufacturer, selected_years):
    # Create the heatmap with Plotly Express
    fig = px.imshow(
        heatmap_data,
        labels=dict(x="Transmission", y="Fuel Type", color="Average Tax"),
        x=heatmap_data.columns,
        y=heatmap_data.index,
                title=f"Average Tax by Fuel Type and Transmission for
{selected_manufacturer} ({selected_years[0]}-{selected_years[1]})",
        aspect="auto",
        color_continuous_scale="Blues_r",
    )

    fig.update_xaxes(showgrid=True, gridcolor='white', side="bottom")
```

```
    fig.update_yaxes(showgrid=True, gridcolor='white')

    annotations = []
    for y_index, fuel in enumerate(heatmap_data.index):
        for x_index, trans in enumerate(heatmap_data.columns):
            tax_value = heatmap_data.loc[fuel, trans]

            if not np.isnan(tax_value):
                annotations.append(
                    dict(
                        x=x_index,
                        y=y_index,
                        text=f"{tax_value:.2f}",
                        xref="x",
                        yref="y",
                        showarrow=False,
                        font=dict(size=12, color='white')
                    )
                )

    fig.update_layout(
        annotations=annotations,
        xaxis_title='Transmission Type',
        yaxis_title='Fuel Type',
        coloraxis_colorbar=dict(
            title='Avg Tax (£)',
            titleside='right'
        ),
        plot_bgcolor='rgba(0,0,0,0)',
    )

    return fig


def plot_yearly_price_distribution(data):
    price_data = data.groupby('year')['price'].mean().reset_index()
```

```
        price_data['formatted_price']   =   price_data['price'].apply(lambda   x:
f"{x/1000:.2f}k")

    color_scale = px.colors.sequential.Blues

    fig = px.bar(
        price_data,
        x='year',
        y='price',
        title='Yearly Price Distribution',
        labels={'price': 'Average Price (£)'},
        text='formatted_price',
        color='price',
        color_continuous_scale=color_scale,
        color_continuous_midpoint=0.5)


    fig.update_traces(textposition='outside')
    fig.update_layout(
        xaxis=dict(
            title='Year',
            tickmode='linear',
            tick0=price_data['year'].min(),
            dtick=1
        ),
        yaxis=dict(
            title='Average Price (£)',
            tickformat='.2f'
        ),
        plot_bgcolor='white',
        margin=dict(l=40, r=40, t=40, b=80)
    )

    return fig



app.layout = get_market_trends_layout
```

```
@app.callback(
    [Output('url', 'pathname'),
     Output('graph-container', 'children')],
    [Input('year-slider', 'value'),
     Input('manufacturer-radio', 'value'),
     Input('back-button', 'n_clicks')]
)
def update_graph(selected_years, selected_manufacturer, n_clicks):

    if n_clicks and n_clicks > 0:
        return '/', dash.no_update



    filtered_data = data[
                    (data['year']  >=  selected_years[0])  &  (data['year']  <=
selected_years[1]) &
        (data['Manufacturer'].str.lower() == selected_manufacturer.lower())
    ]

    if filtered_data.empty:
            return dash.no_update, html.Div("No data available for the selected
criteria", style={'color': 'red', 'fontSize': 18})
    num_shades = 2
    palette = sns.color_palette("Blues_r", num_shades)
    colors = palette.as_hex()[:]
                        trends_data       =       filtered_data.groupby(['year',
'Manufacturer'])['mileage'].mean().reset_index()
    scatter_trace = go.Scatter(x=trends_data['year'],
        y=trends_data['mileage'],
        mode='lines+markers',
        name=selected_manufacturer,
        line=dict(color=colors[1] , width=3),
        marker=dict(
            size=8,
            line=dict(width=2, color=colors[0] ),
            color=colors[0]
        )
    )
```

```
    scatter_layout = go.Layout(
                      title=f'Market   Trend   Analysis:   Average   Mileage   for
{selected_manufacturer}',
        xaxis=dict(
            title='Year',
            showgrid=True,
            gridwidth=1,
            gridcolor='LightBlue',
            tickmode='array',
            tickvals=list(range(selected_years[0], selected_years[1] + 1)),
                      ticktext=[str(year)  for  year  in  range(selected_years[0],
selected_years[1] + 1)]
        ),
        yaxis=dict(
            title='Average Mileage',
            showgrid=True,
            gridwidth=1,
            gridcolor='LightBlue'
        ),
        legend=dict(
            title='Manufacturer',
            x=1,
            y=1,
            xanchor='right',
            yanchor='top',
            orientation='v'
        ),
        plot_bgcolor='white',
        margin=dict(l=20, r=20, t=40, b=40)
    )

    scatter_fig = go.Figure(data=[scatter_trace], layout=scatter_layout)

                                            category_distribution               =
filtered_data['Category'].value_counts().reset_index()
    category_distribution.columns = ['Category', 'Counts']
```

```
    num_shades = 5
    palette = sns.color_palette("Blues_r", num_shades)
    colors = palette.as_hex()[:]


    donut_trace = go.Pie(
        labels=category_distribution['Category'],
        values=category_distribution['Counts'],
        hole=0.5,
        textposition='outside',
        textinfo='percent+label',
        marker=dict(colors=colors),
    )

    donut_layout = go.Layout(
        title=f'Category Distribution for {selected_manufacturer}',
        margin=dict(l=20, r=20, t=40, b=40),
        width=400,
        height=400
    )

    donut_fig = go.Figure(data=[donut_trace], layout=donut_layout)
                    heatmap_data       =       filtered_data.groupby(['fuelType',
'transmission'])['tax'].mean().unstack().fillna(0)
        heatmap_fig   =   generate_heatmap(heatmap_data,   selected_manufacturer,
selected_years)

    price_distribution_fig = plot_yearly_price_distribution(filtered_data)

    graph_layout = html.Div([
        dcc.Loading(
            id="loading-graphs",
            type="circle",
            children=[
                dcc.Graph(figure=scatter_fig, style={'height': '100%'}),
                  dcc.Graph(figure=donut_fig, style={'width': '450px', 'height':
'450px', 'display': 'inline-block'}),
```

```
                dcc.Graph(figure=heatmap_fig, style={'width': '600px', 'height':
'500px', 'display': 'inline-block'}),
                dcc.Graph(figure=price_distribution_fig,
                          style={'width': '830px', 'height': '500px', 'display':
'inline-block'})
            ]
        )
    ])


    return dash.no_update, graph_layout


################################################      MODEL      PERFORMANCE
#########################################################################


def get_model_performance_layout():
    return html.Div([
        html.H1("Model Performance Analysis"),
        html.Br(),
        html.Div([
            dcc.Dropdown(
                id='transmission-dropdown',
                options=[
                    {'label': 'Manual', 'value': 'Manual'},
                    {'label': 'Semi-Auto', 'value': 'Semi-Auto'},
                    {'label': 'Automatic', 'value': 'Automatic'},
                    {'label': 'Other', 'value': 'Other'}
                ],
                value='Manual',
                clearable=False,
                          style={'width': '300px', 'height': '38px', 'display':
'inline-block'}
            ),
            html.Div([
                html.P("Number of Top Models to Display:",
```

```
                    style={'margin': '0 10px 0 0', 'display': 'inline-block',
'vertical-align': 'middle', 'border': '1px solid #ccc', 'padding': '5px'}),
                dcc.Input(
                    id='top-models-input',
                    type='number',
                    value=10,
                    min=1,
                    max=20,
                           style={'width': '100px', 'height': '38px', 'display':
'inline-block'}
                )
            ], style={'display': 'inline-block', 'padding': '5px'}),

            html.Button('Outliers ???', id='toggle-outliers-button', n_clicks=0,
                        style={'height': '38px', 'display': 'inline-block',
'margin-left': '1080px'})
        ], style={'display': 'flex', 'align-items': 'center', 'margin-bottom':
'20px'}),

      dcc.Loading(
          id="loading-all-graphs",
          children=html.Div([
              html.Div([
                    dcc.Graph(id='bar-chart', style={'display': 'inline-block',
'width': '50%'}),
                     dcc.Graph(id='box-plot', style={'display': 'inline-block',
'width': '50%'})
              ], style={'display': 'flex', 'width': '100%'}),

              html.Div([
                              dcc.Graph(id='price-histogram',  style={'display':
'inline-block', 'width': '33%'}),
                    dcc.Graph(id='pie-chart', style={'display': 'inline-block',
'width': '33%'}),
                                  dcc.Graph(id='swarm-chart',  style={'display':
'inline-block', 'width': '33%'})
              ])
          ]),
```

```
            type="default"
        ),
        # Back button
        html.Div([
        html.A(html.Button('Back', id='back-button', n_clicks=0), href='/')
    ], style={'position': 'absolute', 'bottom': '10px', 'left': '10px'})
    ], style={'display': 'flex', 'flex-direction': 'column', 'padding': '20px'})




def remove_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)
    return data[(data[column] >= lower_bound) & (data[column] <= upper_bound)]

def remove_outliers_iqr_grouped(data, column):
    grouped_data = data.groupby('Manufacturer_Country')
      filtered_groups = [remove_outliers_iqr(group, column) for _, group in
grouped_data]

    filtered_data = pd.concat(filtered_groups)

    return filtered_data


def update_box_plot(filtered_data, remove_outliers):
    filtered_data_copy = filtered_data.copy()
     show_points = not remove_outliers   # If we remove outliers, don't show
points
    num_shades = 4
    palette = sns.color_palette("Blues_r", num_shades)
    colors = palette.as_hex()[::2]
```

```
    if remove_outliers:
            filtered_data_copy = remove_outliers_iqr_grouped(filtered_data_copy,
'mileage')

    fig = px.box(filtered_data_copy, x='Manufacturer_Country', y='mileage',
                 title='Mileage Distribution by Manufacturer Country',
                                      points='outliers' if show_points else
False,color_discrete_sequence=colors)
    fig.update_layout(
            title="Mileage Distribution by Manufacturer Country",
            xaxis=dict(
                title='Manufacturer Country',
            ),
            yaxis=dict(
                title='Mileage',
            ),
            plot_bgcolor='rgba(0,0,0,0)',
            paper_bgcolor='rgba(0,0,0,0)',
            margin=dict(l=20, r=20, t=40, b=20),
            template='plotly_white',
        height=400,
        width=800,
        )

    fig.update_xaxes(tickfont=dict(family='serif', size=15, color='black'))

    return fig

def update_manufacturer_swarm_plot(transmission_type):
    if not transmission_type:
        return go.Figure()
    num_shades = 16
    palette = sns.color_palette("Blues", num_shades)
    colors = palette.as_hex()[7::2]
    # Filter the data based on the selected transmission type
    filtered_data = data[data['transmission'] == transmission_type]

    if filtered_data.empty:
```

```
        return go.Figure()

    fig = px.strip(filtered_data,
                   y='mileage',
                   x='fuelType',
                   color='fuelType',
                       title='Distribution of Car Prices by Manufacturer for ' +
str(transmission_type),
                   template='plotly_white',color_discrete_sequence=colors)

    fig.update_traces(jitter=0.7)
    fig.update_layout(xaxis_title='Mileage', yaxis_title='Manufacturer')
    return fig




@app.callback(
    [Output('bar-chart', 'figure'),
     Output('box-plot', 'figure'),
     Output('price-histogram', 'figure'),
     Output('pie-chart', 'figure'),
     Output('swarm-chart', 'figure')],
    [Input('transmission-dropdown', 'value'),
     Input('top-models-input', 'value'),
     Input('toggle-outliers-button', 'n_clicks')]
)
def update_graph(selected_transmission, top_models, n_clicks):
    if not selected_transmission:
        return go.Figure(), go.Figure(), go.Figure()

    filtered_data = data[data['transmission'] == selected_transmission]
    num_shades = 1
    palette = sns.color_palette("Blues_r", num_shades)
    colors = palette.as_hex()[:]
    price_histogram = px.histogram(
        filtered_data,
        x='price',
```

```
        nbins=40,
        histnorm='probability density',
          title=f"Price Distribution for {selected_transmission} Transmission",
color_discrete_sequence=colors )
    price_histogram.update_layout(
        xaxis_title="Price",
        yaxis_title="Density",
        bargap=0.1,
        template='plotly_white'
    )

                    model_counts = filtered_data.groupby(['model',
'Manufacturer']).size().reset_index(name='count')
    model_counts = model_counts.nlargest(top_models, 'count')
    num_shades = 14
    palette = sns.color_palette("Blues_r", num_shades)
    colors = palette.as_hex()[::3]
    bar_data = []
    for manufacturer in model_counts['Manufacturer'].unique():
            manufacturer_data = model_counts[model_counts['Manufacturer'] ==
manufacturer]
        bar_trace = go.Bar(
            x=manufacturer_data['model'],
            y=manufacturer_data['count'],
            name=manufacturer
        )
        bar_data.append(bar_trace)

    bar_layout = go.Layout(
              title=f"Top {top_models} Models for {selected_transmission}
Transmission",
        xaxis=dict(title='Model'),
        yaxis=dict(title='Count'),
        bargap=0.2,
        height=400,
        width=800,
        template='plotly_white',
        barmode='group'
```

```
    )

    bar_fig = go.Figure(data=bar_data, layout=bar_layout)

    for i, trace in enumerate(bar_fig.data):
        trace.marker.color = colors[i]

    bar_fig.update_layout(
        legend_title='Manufacturer',
          legend=dict(orientation='h', yanchor='bottom', y=1.02, xanchor='right',
x=1)
    )
    bar_fig.update_xaxes(categoryorder='total descending')

    remove_outliers = n_clicks % 2 == 1
    box_fig = update_box_plot(filtered_data, remove_outliers)

    country_counts = filtered_data['Manufacturer_Country'].value_counts()
    country_percentages = (country_counts / country_counts.sum()) * 100


     labels_with_percents = {country: f"{country} {percent:.2f}%" for country,
percent in country_percentages.items()}
                                  filtered_data['label_with_percent']           =
filtered_data['Manufacturer_Country'].map(labels_with_percents)
    num_shades = 17
    palette = sns.color_palette("Blues_r", num_shades)
    colors = palette.as_hex()[1:-3:2]
    pie_fig = px.pie(
        filtered_data,
        names='Manufacturer_Country',
        title='Distribution of Cars by Country of Manufacturer',
        color_discrete_sequence=colors
    )

        pie_fig.update_traces(textinfo='percent+label',  textposition='outside',
hoverinfo='label+percent')
```

```
    pie_fig.update_layout(
        uniformtext_minsize=12,
        uniformtext_mode='hide'
    )


    swarm_fig = update_manufacturer_swarm_plot(selected_transmission)

    return bar_fig, box_fig, price_histogram,pie_fig,swarm_fig



############################################        STATISTICAL        ANALYSIS
#########################################################################

def get_numerical_columns_options(df):
    numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns
    options = [{'label': col, 'value': col} for col in numerical_columns]
    return options
def get_category_dynamics_layout():
    dropdown_options = get_numerical_columns_options(data)

    normality_tests_options = [
        {'label': "Kolmogorov-Smirnov Test", 'value': 'ks'},
        {'label': "D'Agostino's K^2 Test", 'value': 'dagostino'}
    ]
    data_copy = data.copy()
    label_encoders = {}

    categorical_cols = data_copy.select_dtypes(include=['object']).columns
    for col in categorical_cols:
        label_encoders[col] = LabelEncoder()
        data_copy[col] = label_encoders[col].fit_transform(data[col])

    pca = PCA()
    pca.fit(data_copy)

    singular_values = pca.singular_values_
```

```
    explained_variance_ratio = pca.explained_variance_ratio_
    cumulative_explained_variance = np.cumsum(explained_variance_ratio) * 100

    components_to_keep = np.argmax(cumulative_explained_variance >= 95) + 1

    n_components_long = range(1, len(cumulative_explained_variance) + 1)


    condition_number = max(singular_values) / min(singular_values)

    rounded_singular_values = np.round(singular_values, 2)
    rounded_explained_variance_ratio = np.round(explained_variance_ratio, 2)
                                rounded_cumulative_explained_variance           =
np.round(cumulative_explained_variance, 2)

     num_features_after_pca = np.count_nonzero(rounded_explained_variance_ratio >
0.0)
    singular_df = pd.DataFrame({
        'Feature': np.arange(1, len(rounded_singular_values) + 1),
        'Singular Value': rounded_singular_values,
        'Explained Variance Ratio': rounded_explained_variance_ratio,
        'Cumulative Explained Variance': rounded_cumulative_explained_variance
    })
     condition_df = pd.DataFrame({'Features': ['Condition Number', 'Num Features
After PCA'],
                                                    'Values': [condition_number,
num_features_after_pca]})

    pca_plot = dcc.Graph(
        id='pca-plot',
        figure={
            'data': [
                        {'x': np.arange(1, len(rounded_singular_values) + 1), 'y':
rounded_singular_values, 'type': 'bar',
                 'name': 'Singular Values'},
                  {'x': np.arange(1, len(rounded_cumulative_explained_variance) +
1),
```

```
                    'y': rounded_cumulative_explained_variance, 'type': 'line',
'name': 'Cumulative Explained Variance',
                'yaxis': 'y2'}
            ],
            'layout': {
                        'title': 'Effect of PCA: Singular Values and Cumulative
Explained Variance',
                'xaxis': {'title': 'Feature'},
                'yaxis': {'title': 'Singular Value'},
                          'yaxis2': {'title': 'Cumulative Explained Variance',
'overlaying': 'y', 'side': 'right'},
                'barmode': 'group'
            }
        }
    )

    pca_table = html.Div([
        dash_table.DataTable(
            id='singular-table',
            columns=[{'name': i, 'id': i} for i in singular_df.columns],
            data=singular_df.to_dict('records'),
            style_table={'overflowX': 'scroll'},
                 style_cell={'minWidth': '280px', 'width': '280px', 'maxWidth':
'280px', 'overflow': 'hidden',
                         'textOverflow': 'ellipsis'}
        )
    ])
     condition_df = pd.DataFrame({'Features': ['Condition Number', 'Num Features
After PCA'],
                                                  'Values': [condition_number,
num_features_after_pca]})

    condition_table = html.Div([
        dash_table.DataTable(
            id='condition-table',
            columns=[
                {'name': 'Features', 'id': 'Features'},
                {'name': 'Values', 'id': 'Values', 'type': 'numeric',
```

```
                'format': {'specifier': '.2f'}}
            ],
            data=condition_df.to_dict('records'),
              style_table={'overflowX': 'scroll', 'width': '100%'},  # Set width
to 50%
             style_header={'backgroundColor': 'rgb(230, 230, 230)', 'fontWeight':
'bold'},
              style_cell={'textAlign': 'left', 'minWidth': '100px', 'maxWidth':
'180px', 'whiteSpace': 'normal',
                          'overflow': 'hidden', 'textOverflow': 'ellipsis'},
        ),
    ], style={'display': 'inline-block', 'verticalAlign': 'top', 'margin-right':
'10px'}) # Set display to inline-block

    download_button= html.Div([
          html.Button("Download PCA Data", id="btn-download-csv"),  # Button to
trigger download
        dcc.Download(id="download-csv")  # Component to handle the download
    ], style={'position': 'absolute', 'bottom': '20px', 'right': '20px'})
    back_button = html.Div([
        html.A(html.Button('Back', id='back-button', n_clicks=0), href='/')
    ], style={'position': 'absolute', 'bottom': '20px', 'left': '20px'})
    pca_analysis_content = html.Div([
        html.H1("PCA Analysis"),
        html.Div([
            pca_table,
            html.Div([
                html.H3("Effect of Principal Component Analysis"),
                html.Div([

                            html.Div(id='pca-graph-container', style={'margin-top':
'10px'})
                ]),
            ], style={'flex': '1'})
        ], style={'display': 'flex'})
    ])

    return html.Div([
```

```
        html.H1("Statistical Analysis"),
        html.Br(),
        dcc.Tabs(id="tabs", children=[
            dcc.Tab(label='PCA Analysis', value='tab-pca', children=[
                pca_analysis_content,
                html.Br(),
                condition_table,
                download_button,
                back_button

            ]),
    dcc.Tab(label='Normality Test', value='tab-normality', children=[
                html.Div([
                    html.Br(),
                    html.H3("Select Column to Test for Normality"),

                    dcc.Dropdown(
                        id='normality-test-dropdown',
                        options=dropdown_options,
                        value='price'
                    ),
                    dcc.Graph(id='histogram-plot'),
                    back_button
                ]),
                html.Div([
                    html.Br(),
                    html.H3("Select Normality Tests to Perform"),
                    html.Br(),
                    dcc.Checklist(
                        id='normality-tests-checklist',
                        options=normality_tests_options,
                        value=[]
                    ),
                    html.Br(),
                    html.Div(id='normality-test-results')
                ]),
            ]),
```

```
                dcc.Tab(label='Heatmap & Correlation', value='tab-heatmap',
children=[
            html.Div([
                html.Br(),
                dcc.Slider(
                    id='heatmap-splom-selector',
                    min=0,
                    max=1,
                    step=1,
                    marks={
                        0: {'label': 'Heatmap'},
                        1: {'label': 'Scatter Plot Matrix'}
                    },
                    value=0,
                ),
                html.Br(),
                dcc.Loading(
                    id="loading-1",
                    type="default",
                    children=html.Div(id='heatmap-splom-content')
                ),
                back_button
            ], style={'width': '100%', 'height': '100%'})
        ]),
        dcc.Tab(label='Statistical Summary', value='tab-stats', children=[
            html.Div([
                html.Br(),
                html.H3("Select Columns for Statistical Summary"),
                html.Br(),
                dcc.Checklist(
                    id='statistical-summary-columns',
                     options=[{'label': col.title(), 'value': col} for col in
data.columns],
                    value=[],
                    inline=True,
                    style={'margin-bottom': '10px'}
                ),
                dcc.Loading(
```

```
                        id="loading-statistical-summary",
                        type="default",
                        children=html.Div(id='statistical-summary-results')
                    ),
                    back_button
                ])
            ]),
        ]),
        html.Div(id='tabs-content')
    ])



app.layout = get_category_dynamics_layout()
@app.callback(
    Output("download-csv", "data"),
    Input("btn-download-csv", "n_clicks"),
    prevent_initial_call=True
)
def generate_csv(n_clicks):
    data_copy = data.copy()
    label_encoders = {}

    categorical_cols = data_copy.select_dtypes(include=['object']).columns
    for col in categorical_cols:
        label_encoders[col] = LabelEncoder()
        data_copy[col] = label_encoders[col].fit_transform(data[col])

    pca = PCA()
    pca.fit(data_copy)

    singular_values = pca.singular_values_

    condition_number = max(singular_values) / min(singular_values)

    rounded_singular_values = np.round(singular_values, 2)
```

```
    explained_variance_ratio = pca.explained_variance_ratio_
    rounded_explained_variance_ratio = np.round(explained_variance_ratio, 2)

    num_features_after_pca = np.count_nonzero(rounded_explained_variance_ratio > 0.0)
                singular_df = pd.DataFrame({'Feature': np.arange(1, len(rounded_singular_values) + 1),
                                'Singular Value': rounded_singular_values,
                                                'Explained Variance Ratio': explained_variance_ratio})

    return dcc.send_data_frame(singular_df.to_csv, filename="my_data.csv")

@app.callback(
    Output('pca-graph-container', 'children'),
    [Input('pca-graph-container', 'id')]
)
def generate_pca_graph(n_clicks):
    data_copy = data.copy()
            label_encoders = {col: LabelEncoder() for col in data_copy.select_dtypes(include=['object']).columns}
    for col, le in label_encoders.items():
        data_copy[col] = le.fit_transform(data_copy[col])

    pca = PCA()
    pca.fit(data_copy.select_dtypes(include=[np.number]))

    explained_variance_ratio = pca.explained_variance_ratio_
    cumulative_explained_variance = np.cumsum(explained_variance_ratio) * 100
    components_to_keep = np.argmax(cumulative_explained_variance >= 95) + 1

    fig = go.Figure()

    fig.add_trace(go.Scatter(
        x=np.arange(1, len(explained_variance_ratio) + 1),
        y=cumulative_explained_variance,
        mode='lines+markers',
        name='Cumulative Explained Variance',
```

```
        line=dict(color='blue')
    ))

                fig.add_hline(y=95,     line=dict(color='red',     dash='dash'),
annotation_text="95% Threshold", annotation_position="bottom right")
     fig.add_vline(x=components_to_keep,  line=dict(color='green',  dash='dash'),
annotation_text="Optimal       Components:       {}".format(components_to_keep),
annotation_position="bottom right")

    fig.update_layout(
        title='Cumulative Explained Variance vs Number of Components',
        xaxis_title='Number of Components',
        yaxis_title='Cumulative Explained Variance (%)',
        template='plotly_white'
    )

    return dcc.Graph(figure=fig)

app.layout = get_category_dynamics_layout()

# Layout
app.layout = get_category_dynamics_layout()




@app.callback(
    Output('statistical-summary-results', 'children'),
    [Input('statistical-summary-columns', 'value')]
)
def update_statistical_summary(selected_columns):
    if not selected_columns:
        return html.Div("Select columns to generate statistical summary.")

    summaries = []
    for col in selected_columns:
```

```
                                                    summary          =
data[col].describe().to_frame().reset_index().rename(columns={'index':
'Statistic'})

        if data[col].dtype == 'float64' or data[col].dtype == 'int64':
            # Format numeric columns with .2f precision
            summary[col] = summary[col].apply(lambda x: "{:.2f}".format(x))

        summaries.append(html.Div([
            html.Br(),
            html.H3(f'Statistical Summary for {col} \n'),
            html.Br(),
            dash_table.DataTable(
                columns=[{"name": i, "id": i} for i in summary.columns],
                data=summary.to_dict('records'),
                style_table={'overflowX': 'auto'},
                style_cell={'textAlign': 'center', 'padding': '5px'},
                     style_header={'backgroundColor': 'lightgrey', 'fontWeight':
'bold'},
                style_data={'whiteSpace': 'normal', 'height': 'auto'},
                fill_width=False,
            ),
            html.Br(),
            html.Div([
                html.H3('Multivariate Kernel Density Estimate \n'),
                dcc.Graph(id=f'density-plot-{col}')
            ])
        ]))

                    density_fig   =   px.density_contour(data,   x=col,   y=col,
title=f'Multivariate KDE for {col}')
        summaries[-1].children[-1].children[1].figure = density_fig

    return summaries

num_shades = 1
palette = sns.color_palette("Blues", num_shades)
colors = palette.as_hex()[::]
```

```
# Layout
app.layout = get_category_dynamics_layout()

@app.callback(
   Output('histogram-plot', 'figure'),
   [Input('normality-test-dropdown', 'value')]
)
def update_histogram(selected_column):
    fig = px.histogram(data, x=selected_column,nbins=200 ,title=f'Histogram of
{selected_column}',color_discrete_sequence=colors)
   return fig

@app.callback(
   Output('normality-test-results', 'children'),
   [Input('normality-test-dropdown', 'value'),
    Input('normality-tests-checklist', 'value')]
)
def update_normality_tests(selected_column, selected_tests_input):
   if selected_column is None:
       return "No column selected"

   data_sample = data[selected_column].dropna()

   if isinstance(selected_tests_input, list):
       selected_tests = selected_tests_input
   elif isinstance(selected_tests_input, str):
       selected_tests = selected_tests_input.split(',')
   else:
               raise ValueError("selected_tests_input must be a list or a
comma-separated string")

   results_text = []

   for test in selected_tests:
       test = test.strip().lower()  # Ensure it is stripped and lowercase
       test_result = f"{test.capitalize()} Test:"
       if test == 'ks':
```

```
                        ks_stat, ks_p = stats.kstest(data_sample, 'norm',
args=(np.mean(data_sample), np.std(data_sample)))
            test_result += f" \n - Test Statistic: {ks_stat:.2f},\n p-value:
{ks_p:.2f}"
            test_result += " - \n Result: The data does not appear to follow a
normal distribution." if ks_p < 0.05 else " - Result: The data appears to
follow a normal distribution."
        elif test == 'dagostino':
            dagostino_stat, dagostino_p = stats.normaltest(data_sample)
               test_result += f" \n - Test Statistic: {dagostino_stat:.2f}, \n
p-value: {dagostino_p:.2f}"
            test_result += " \n - Result: The data does not appear to follow a
normal distribution." if dagostino_p < 0.05 else " - \n Result: The data
appears to follow a normal distribution."

        results_text.append(test_result)

    return "\n\n".join(results_text).strip()
@app.callback(
    Output('heatmap-splom-content', 'children'),
     [Input('heatmap-splom-selector', 'value')]   # Slider sends numeric values
now
)
def update_visual(selected_option):
    numerical_data = data.select_dtypes(include=[np.number])

    if selected_option == 0:
        correlation_matrix = numerical_data.corr(method='pearson')
        fig = px.imshow(correlation_matrix, text_auto=".2f", aspect='auto',
                                        labels=dict(x='Variable', y='Variable',
color='Correlation'),
                                                x=correlation_matrix.columns,
y=correlation_matrix.columns,
                        color_continuous_scale='Blues', zmin=-1, zmax=1)
         fig.update_layout(title_text='Pearson Correlation Coefficient Heatmap',
title_x=0.5, autosize=True)
    elif selected_option == 1:
        fig = px.scatter_matrix(numerical_data,
```

```
                                dimensions=numerical_data.columns,
                                title='Scatter Plot Matrix',

color_discrete_sequence=px.colors.qualitative.Plotly)
        fig.update_traces(diagonal_visible=False)
        fig.update_layout(autosize=True)

    return dcc.Graph(figure=fig, style={'width': '100%', 'height': '100vh'})

def get_home_layout():
    return html.Div([
        html.Div(style={'height': '30px'}),
                html.H1("Car  Market  Analysis  Dashboard",  style={'text-align':
'center'}),
        html.Div(style={'height': '40px'}),
        dbc.Row([
            dbc.Col(
                dcc.Link(
                    html.Div([
                                html.Img(src="/assets/tab01.jpg", style={'width':
'100%'}),
                        html.P("Market Trend Analysis", style={
                            'text-align': 'center',
                            'font-weight': 'bold',
                            'font-size': '25px',
                            'color': 'black',
                            'font-family': 'Open Sans'}),
                    ], style={'border': '1px solid black', 'padding': '10px'}),
                    href="/market-trends",
                    style={'text-decoration': 'none', 'color': 'black'}
                )
            ),
            dbc.Col(
                dcc.Link(
                    html.Div([
                                html.Img(src="/assets/tab02.jpg", style={'width':
'100%'}),
                        html.P("Model Performance Analysis", style={
```

```
                        'text-align': 'center',
                        'font-weight': 'bold',
                        'font-size': '25px',
                        'color': 'black',
                        'font-family': 'Open Sans'}),
                ], style={'border': '1px solid black', 'padding': '10px'}),
                href="/model-performance",
                style={'text-decoration': 'none', 'color': 'black'}
            )
        ),
        dbc.Col(
            dcc.Link(
                html.Div([
                            html.Img(src="/assets/tab03.jpg", style={'width':
'100%'}),
                    html.P("Car Industry Statistical Analysis", style={
                        'text-align': 'center',
                        'font-weight': 'bold',
                        'font-size': '25px',
                        'color': 'black',
                        'font-family': 'Open Sans'}),
                ], style={'border': '1px solid black', 'padding': '10px'}),
                href="/category-dynamics",
                style={'text-decoration': 'none', 'color': 'black'}
            )
        ),
    ]),
    html.Div([
        dbc.Button("About Us", id="about-us-button", color="primary",
                   style={'position': 'fixed', 'bottom': '10px', 'left':
'10px'}),
        dbc.Tooltip(
             "We analyzed data on used car sales to gain insights into market
trends and pricing dynamics.",
            target="about-us-button",
            trigger="hover"
        )
    ]),
```

```
        html.Div([
                            dbc.Textarea(id="feedback-textarea",  placeholder="Enter
feedback...", style={
                    'width': '180px',
                    'height': '90px',
                }),
                 dbc.Button("Submit Feedback", id="submit-feedback", color="primary",
className="mr-1", style={
                    'margin-top': '10px',
                }),
            ], style={
                'position': 'fixed',
                'bottom': '10px',
                'right': '10px',
                'z-index': '1000'
            }),
    ])


app.layout = html.Div([
    dcc.Location(id='url', refresh=False),
    html.Div(id='page-content')
])

@app.callback(
    Output('page-content', 'children'),
    Input('url', 'pathname')
)
def display_page(pathname):
    if pathname == '/market-trends':
        return get_market_trends_layout()
    elif pathname == '/model-performance':
        return get_model_performance_layout()
    elif pathname == '/category-dynamics':
        return get_category_dynamics_layout()
    else:
        return get_home_layout()
```

```
@app.callback(
    Output('feedback-textarea', 'value'),
    [Input('submit-feedback', 'n_clicks')],
    [State('feedback-textarea', 'value')],
    prevent_initial_call=True
)
def submit_feedback(n_clicks, feedback):
    if n_clicks and feedback:

        return ""
    else:
        return dash.no_update

app.server.run(
    debug=False,
    port=8080,
    host='0.0.0.0'
)
```

14. References:

Meruvulikith. (2024). *90,000 cars data from 1970 to 2024*. Kaggle. https://www.kaggle.com/datasets/meruvulikith/90000-cars-data-from-1970-to-2024

Lin, Z. (2022). Analysis of the used car market in the United States. University of Michigan. https://doi.org/10.54691/bcpbm.v29i.2193

Plotly Technologies Inc. (2024). Dash User Guide. https://dash.plotly.com