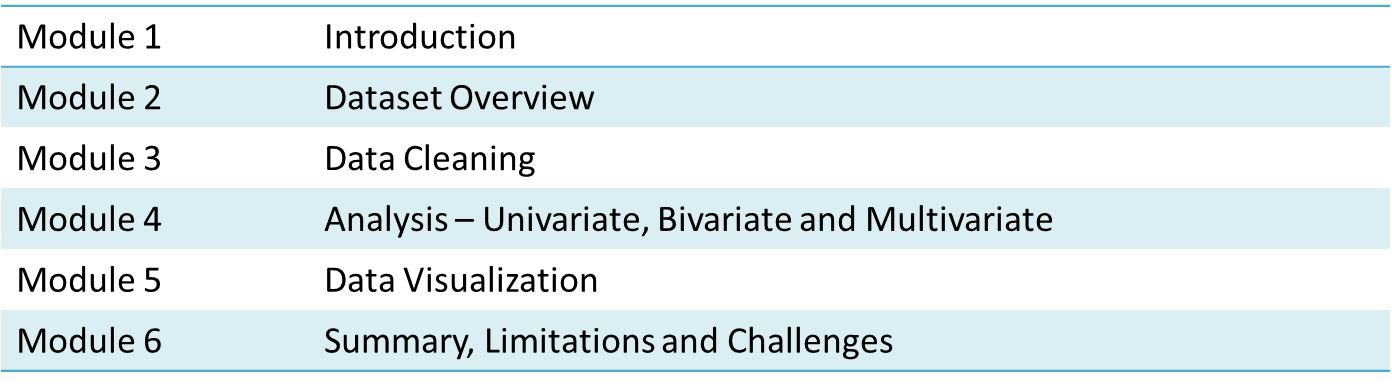
**EXPLORATORY DATA ANALYSIS PROJECT DOCUMENTATION**

**Project Scope:**

Choose a dataset and perform exploratory data analysis to understand the patterns, trends, and relationships within the data.

Use visualizations like histograms, scatter plots, and box plots to represent the data.



**Module 1. Introduction**

**Importance of EDA in the data science workflow.**

**Objectives of your EDA.**

EDA stands for Exploratory Data Analysis. It is an approach to analyzing and visualizing data sets to summarize their key characteristics, often with the help of statistical graphics and other data visualization methods. The primary goal of EDA is to uncover patterns, trends, relationships, and anomalies within the data, helping to generate hypotheses and insights.

Here are some key aspects of Exploratory Data Analysis:

1. Data Summarization: EDA involves summarizing the main characteristics of a dataset, such as its central tendency, spread, and shape of the distribution. Descriptive statistics, like mean, median, and standard deviation, are commonly used.

2. Data Visualization: Graphical representations, such as histograms, box plots, scatter plots, and heatmaps, are employed to visualize the distribution and relationships within the data. Visualization aids in understanding the structure and patterns present in the dataset.

3. Identifying Patterns and Trends: EDA helps in identifying patterns, trends, and relationships between variables. This includes exploring correlations, dependencies, and potential outliers in the data.

4. Handling Missing Data: Exploring the extent of missing data and deciding on strategies to handle missing values is part of EDA. Techniques like imputation or removal of missing values may be considered.

5. Outlier Detection: EDA includes the identification of outliers, which are data points that significantly deviate from the overall pattern. Outliers can impact statistical analyses and may need special consideration.

6. Feature Engineering: Exploring relationships between variables may suggest opportunities for feature engineering, where new variables are created based on existing ones to enhance the predictive power of models.

7. Data Transformation: EDA may involve transforming variables to make them suitable for analysis. Common transformations include log transformations or normalization.

8. Hypothesis Generation: EDA often leads to the generation of hypotheses about the underlying patterns in the data. These hypotheses can then be tested using more formal statistical methods.

9. Iterative Process: EDA is an iterative process. As insights are gained, further questions may arise, prompting additional analysis and exploration.

EDA is a crucial step in the data analysis process, providing a foundation for subsequent modelling and decision-making. It is widely used in various fields, including statistics, data science, and machine learning, to gain a deeper understanding of datasets before applying more advanced techniques.

**Importance and Objectives of EDA in the data science workflow.**

Exploratory Data Analysis (EDA) plays a crucial role in the data science workflow for several reasons:

1. Understanding Data Structure:

- EDA helps data scientists gain an initial understanding of the structure, content, and quality of the dataset. This includes identifying data types, variable distributions, and potential issues with data integrity.

2. Identifying Patterns and Trends:

- EDA allows for the identification of patterns, trends, and relationships within the data. Discovering these insights is essential for formulating hypotheses and guiding subsequent analysis.

3. Data Cleaning and Preprocessing:

- EDA helps in identifying and addressing data quality issues such as missing values, outliers, or inconsistencies. Cleaning and preprocessing the data are crucial steps before building models or conducting further analysis.

4. Feature Engineering:

- Through exploration, potential features for modelling may be uncovered. EDA can inspire the creation of new variables or transformations that enhance the predictive power of machine learning models.

5. Model Selection and Evaluation:

- Understanding the characteristics of the data can guide the selection of appropriate models. EDA insights can also inform model evaluation strategies, helping to set realistic expectations for model performance.

6. Hypothesis Generation and Testing:

- EDA often leads to the generation of hypotheses about the relationships between variables. These hypotheses can be formally tested using statistical methods, contributing to the scientific rigor of the analysis.

7. Communication with Stakeholders:

- EDA results provide a basis for communicating findings with stakeholders who may not have a technical background. Visualizations and summaries from EDA help in conveying complex information in an accessible manner.

8. Risk Mitigation:

- Detecting outliers and understanding data distributions during EDA can help mitigate risks associated with erroneous data points or assumptions. This is especially important in critical decision-making processes.

9. Optimizing Model Performance:

- EDA can reveal insights that lead to optimizations in model performance. By understanding the distribution of target variables and relationships between features, models can be fine-tuned for better accuracy.

10. Iterative Data Exploration:

- EDA is an iterative process that often involves refining hypotheses, revisiting data cleaning steps, and adapting analysis based on new insights. This iterative nature ensures a comprehensive understanding of the data.

11. Data Visualization for Interpretability:

- Visualizations generated during EDA provide a visual representation of data patterns, making it easier to communicate findings and support decision-making.

In summary, EDA is a foundational step in the data science workflow. It not only aids in understanding the data but also influences subsequent stages, contributing to the overall success of data-driven projects.

**Module 2. Dataset Overview**

Summary of the dataset you are working with.

- include information on the size, structure, and basic statistics (mean, median, standard deviation, etc.).

**Dataset File: data.csv (provided in group)**

If you have a dataset in a file (e.g., CSV, Excel), you can use the following code snippet as an example:

import pandas as pd

# Load your dataset

file\_path = 'your\_dataset.csv' # Replace with your actual file path

data = pd.read\_csv(file\_path)

# Display basic information about the dataset

print("Dataset Information:")

print(data.info())

# Display basic statistics

print("\nBasic Statistics:")

print(data.describe())

# Display the first few rows of the dataset

print("\nFirst few rows:")

print(data.head())

This code uses the Pandas library to load a dataset, display general information about its structure, show basic statistics, and print the first few rows.

You need to replace 'your\_dataset.csv' with the actual path to your dataset file.

If your dataset is not in a tabular format or has a different structure, you might need to adjust the code accordingly. Additionally, you can explore other Pandas functions and visualizations to gain deeper insights into your dataset.

For an Exploratory Data Analysis (EDA) project in Python, you'll commonly use a variety of libraries to handle data, perform analysis, and create visualizations. Here's a list of some essential Python libraries for EDA:

1. Pandas:

- Used for data manipulation and analysis. It provides data structures like DataFrame for efficient data handling.

pip install pandas

2. NumPy:

- Used for numerical operations and working with arrays. Pandas relies on NumPy for many of its operations.

pip install numpy

3. Matplotlib:

- A versatile plotting library for creating static, animated, and interactive visualizations.

pip install matplotlib

4. Seaborn:

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

pip install seaborn

5. Scikit-learn:

- A machine learning library that includes tools for data preprocessing, modeling, and evaluation. Useful for advanced analysis beyond basic EDA.

pip install scikit-learn

6. Statsmodels:

- Provides classes and functions for estimating and testing statistical models.

pip install statsmodels

7. Scipy:

- A library for scientific and technical computing. It builds on NumPy and provides additional functionality.

pip install scipy

8. Plotly:

- A library for creating interactive plots and dashboards.

pip install plotly

9. Bokeh:

- Another library for creating interactive and real-time plots.

pip install bokeh

10. Jupyter Notebooks:

- An interactive computing environment that allows you to create and share documents containing live code, equations, visualizations, and narrative text.

pip install jupyter

These libraries provide a solid foundation for conducting Exploratory Data Analysis in Python. Depending on your specific needs, you may explore additional libraries or tools as your project progresses. Make sure to install the libraries in your Python environment using the provided `pip install` commands.

In an Exploratory Data Analysis (EDA) project, you often calculate the mean, median, and standard deviation to summarize the central tendency and spread of your dataset. Here's how you can calculate these statistics using Python and the Pandas library:

Assuming you have a Pandas DataFrame named `df`, and you want to calculate the mean, median, and standard deviation for a specific column, you can use the following methods:

import pandas as pd

# Example DataFrame

data = {'value': [10, 20, 30, 40, 50]}

df = pd.DataFrame(data)

# Calculate mean, median, and standard deviation

mean\_value = df['value'].mean()

median\_value = df['value'].median()

std\_dev\_value = df['value'].std()

print("Mean:", mean\_value)

print("Median:", median\_value)

print("Standard Deviation:", std\_dev\_value)

Replace `'value'` with the actual column name from your dataset.

If you want to calculate these statistics for the entire DataFrame, you can use the `describe()` method:

summary\_stats = df.describe()

print(summary\_stats[['mean', '50%', 'std']])

This will provide a summary of various statistics, including mean (average), median (50%), and standard deviation, for all numerical columns in your DataFrame.

These calculations provide key insights into the distribution and central tendency of your data during the exploratory phase of your analysis.

**Module 3. Data Cleaning**

**Identification and handling missing values, duplicates, and outliers if any.**

**Address any data quality issues that may impact the analysis.**

Data cleaning is a crucial step in the data analysis process as it ensures that the dataset is free from errors, inconsistencies, and missing values, allowing for more accurate and reliable analysis. Here's a general guide on how to handle missing values, duplicates, and outliers in Python using Pandas:

1. Handling Missing Values:

Identify Missing Values:

# Check for missing values in each column

print(df.isnull().sum())

# Check for missing values in the entire dataset

print(df.isnull().any().any())

Handling Missing Values:

- Remove Rows with Missing Values:

# Remove rows with any missing values

df\_cleaned = df.dropna()

- Imputation (Filling Missing Values):

# Fill missing values with the mean of the column

df['column\_name'].fillna(df['column\_name'].mean(), inplace=True)

2. Handling Duplicates:

Identify Duplicates:

# Check for duplicate rows

print(df.duplicated().sum())

Handling Duplicates:

# Remove duplicate rows

df\_no\_duplicates = df.drop\_duplicates()

3. Handling Outliers:

Identify Outliers (using box plots or other visualization methods):

import seaborn as sns

import matplotlib.pyplot as plt

# Box plot for outlier detection

sns.boxplot(x=df['column\_name'])

plt.show()

Handling Outliers:

- Remove Outliers:

# Remove outliers using z-scores

from scipy import stats

z\_scores = stats.zscore(df['column\_name'])

df\_no\_outliers = df[(z\_scores < 3) & (z\_scores > -3)]

- Cap or Transform Outliers:

# Cap outliers to a specified threshold

threshold = 3

df['column\_name'] = np.where(df['column\_name'] > threshold, threshold, df['column\_name'])

4. Addressing Data Quality Issues:

- Correcting Inconsistent Data:

- Standardize data formats (e.g., date formats, units).

- Correct typos or errors in categorical variables.

- Handling Irrelevant Data:

- Identify and remove irrelevant columns.

- Handling Inaccurate Data:

- Cross-check data with external sources for accuracy.

- Correct inaccuracies based on domain knowledge.

5. Reassess the Cleaned Data:

# Check the cleaned data

print(df\_cleaned.head())

print(df\_cleaned.describe())

Let's delve into the theoretical aspects of each step in the Data Cleaning process:

- Description: Missing values in a dataset can impact the accuracy and reliability of analyses. Identifying where data is missing is the first step.

- Theory:

- The `isnull()` method in Pandas returns a DataFrame of the same shape as the input, where each element is a Boolean indicating whether the corresponding element in the original DataFrame is null (True) or not (False).

- The `sum()` function then calculates the total number of missing values for each column.

Handling Missing Values:

- Description: Once missing values are identified, they can be addressed through removal or imputation.

- Theory:

- Removing rows with missing values (`dropna()`) is suitable when the number of missing values is relatively small, and removal won't significantly impact the dataset.

- Imputation involves filling missing values with estimated or calculated values, such as the mean, median, or mode of the column (`fillna()`).

2. Handling Duplicates:

Identify Duplicates:

- Description: Duplicates can distort analysis results, and it's important to identify and handle them appropriately.

- Theory:

- The `duplicated()` method in Pandas returns a Boolean series indicating whether each row is a duplicate of a previous row.

- The `sum()` function then calculates the total number of duplicate rows.

Handling Duplicates:

- Description: Duplicate rows can be removed to ensure that each observation is unique.

- Theory:

- The `drop\_duplicates()` method removes duplicate rows from the dataset. It considers all columns by default but can be configured to consider specific columns.

3. Handling Outliers:

Identify Outliers:

- Description: Outliers are extreme values that deviate from the overall pattern of a dataset. Identifying outliers is crucial for accurate analysis.

- Theory:

- Visualization methods, such as box plots, can highlight the distribution of a variable and help identify values that fall significantly outside the expected range.

Handling Outliers:

- Description: Outliers can be addressed through removal, transformation, or capping.

- Theory:

- Z-scores are used to identify how many standard deviations a data point is from the mean. Values with high z-scores (typically above 3 or below -3) are considered outliers and can be removed or transformed.

4. Addressing Data Quality Issues:

- Correcting Inconsistent Data:

- Description: Inconsistent data can arise from varying formats or units, impacting analyses.

- Theory:

- Standardizing data formats ensures uniformity, making it easier to perform calculations and comparisons.

- Correcting typos or errors in categorical variables improves the accuracy of analyses.

- Handling Irrelevant Data:

- Description: Irrelevant columns provide no valuable information and can be safely removed.

- Theory:

- Removing irrelevant columns simplifies the dataset and focuses analysis on essential variables.

- Handling Inaccurate Data:

- Description: Inaccurate data can lead to incorrect insights.

- Theory:

- Cross-checking data with external sources helps validate its accuracy.

- Correcting inaccuracies based on domain knowledge ensures data reliability.

5. Reassess the Cleaned Data:

- Description: After implementing cleaning procedures, it's essential to verify that the dataset is now suitable for analysis.

- Theory:

- Reviewing the cleaned data using methods like `head()` and `describe()` provides a quick overview to ensure that the cleaning process has been effective.

**Module 4. Analysis**

**Univariate Analysis**

**Explore individual variables in the dataset.**

**Utilize histograms, box plots, and summary statistics to understand the distribution of each variable.**

**Identify any patterns or anomalies.**

Univariate Analysis:

Univariate analysis focuses on exploring and describing individual variables in a dataset. It aims to understand the distribution, central tendency, and variability of each variable separately. This analysis is crucial for identifying patterns, trends, and potential outliers within a single variable.

Practical Implementation with Code:

Let's use Python and popular libraries such as Pandas, Matplotlib, and Seaborn to perform univariate analysis on a hypothetical dataset. In this example, we'll assume you have a Pandas DataFrame named `df`.

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Assume 'df' is your DataFrame

### Univariate Analysis for a Numeric Variable ###

# Histogram

plt.figure(figsize=(10, 6))

sns.histplot(df['numeric\_variable'], bins=20, kde=True, color='blue')

plt.title('Histogram of Numeric Variable')

plt.xlabel('Numeric Variable')

plt.ylabel('Frequency')

plt.show()

# Box Plot

plt.figure(figsize=(8, 5))

sns.boxplot(x=df['numeric\_variable'], color='green')

plt.title('Box Plot of Numeric Variable')

plt.xlabel('Numeric Variable')

plt.show()

# Summary Statistics

summary\_stats\_numeric = df['numeric\_variable'].describe()

print("Summary Statistics for Numeric Variable:")

print(summary\_stats\_numeric)

### Univariate Analysis for a Categorical Variable ###

# Count Plot

plt.figure(figsize=(10, 6))

sns.countplot(x=df['categorical\_variable'], palette='pastel')

plt.title('Count Plot of Categorical Variable')

plt.xlabel('Categorical Variable')

plt.ylabel('Count')

plt.show()

# Pie Chart

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

df['categorical\_variable'].value\_counts().plot.pie(autopct='%1.1f%%', colors=sns.color\_palette('Set3'), startangle=90)

plt.title('Pie Chart of Categorical Variable')

plt.show()

# Value Counts

value\_counts\_categorical = df['categorical\_variable'].value\_counts()

print("Value Counts for Categorical Variable:")

print(value\_counts\_categorical)

Replace `'numeric\_variable'` and `'categorical\_variable'` with the actual names of your numeric and categorical variables.

Explanation:

- The histogram provides a visual representation of the distribution of a numeric variable, showing the frequency of values within specified bins.

- The box plot summarizes the distribution of a numeric variable, highlighting key statistics like the median, quartiles, and potential outliers.

- Count plots and pie charts are used for exploring the distribution of categorical variables, showing the count or proportion of each category.

- Summary statistics (mean, median, etc.) provide a numerical summary of the central tendency and variability of a numeric variable.

This univariate analysis helps you gain insights into the characteristics of individual variables, facilitating a deeper understanding of the dataset.

Numeric Variables:

- Definition: Numeric variables are quantitative and represent measurable quantities. They can take on numerical values and undergo mathematical operations. Examples include age, income, height, and temperature.

- Identification:

- Data Type: In a dataset, numeric variables are often represented with data types such as integer (`int`) or float (`float`).

- Statistical Measures: Numeric variables can be subject to statistical measures like mean, median, standard deviation, etc.

- Visual Inspection: Columns with values that look like numbers (e.g., 1, 2.5, -10) are likely numeric.

Categorical Variables:

- Definition: Categorical variables represent categories or labels, and they are used to classify data into groups. Examples include gender, color, city, and education level.

- Identification:

- Data Type: Categorical variables are typically represented as strings (object) or categorical data types.

- Limited Unique Values: Categorical variables often have a limited number of unique values.

- Visual Inspection: Columns with values like 'Male'/'Female', 'Red'/'Blue', or 'High School'/'College' are likely categorical.

How to Identify Them from a Dataset:

1. Check Data Types:

- Use the `dtypes` attribute in Pandas to view the data types of each column. Numeric variables may have types like `int64` or `float64`, while categorical variables may have types like `object` or `category`.

print(df.dtypes)

2. Check Unique Values:

- For numeric variables, you can use the `nunique()` method to count the number of unique values. If the number is large, it's likely a numeric variable.

print(df['numeric\_variable'].nunique())

For categorical variables, use `value\_counts()` to get a count of unique values.

print(df['categorical\_variable'].value\_counts())

3. Visual Inspection:

- Plotting histograms or box plots for numeric variables and count plots for categorical variables can provide a visual sense of their distributions.

import seaborn as sns

import matplotlib.pyplot as plt

sns.histplot(df['numeric\_variable'], bins=20, kde=True)

plt.show()

sns.countplot(x=df['categorical\_variable'])

plt.show()

4. Domain Knowledge:

- Sometimes, domain knowledge about the context of the data can help identify the nature of variables. For example, knowing that 'Age' is typically a numeric variable and 'Gender' is categorical.

5. Summary Statistics:

- Generate summary statistics using the `describe()` method. This will provide insights into the central tendency and distribution of numeric variables.

print(df.describe())

By combining these methods, you can effectively identify whether a variable in your dataset is numeric or categorical, which is crucial for subsequent data analysis and modeling tasks.

**Module 4. Analysis**

**Bivariate Analysis**

**Explore relationships between pairs of variables.**

**Use scatter plots, correlation matrices, and other visualizations to identify patterns and dependencies.**

**Look for potential correlations or causations.**

Bivariate analysis involves exploring relationships between pairs of variables in a dataset. The goal is to understand how changes in one variable relate to changes in another. Bivariate analysis is crucial for identifying patterns, dependencies, correlations, or potential causations between variables.

#### Practical Implementation with Code:

Let's use Python and popular libraries such as Pandas, Matplotlib, and Seaborn to perform bivariate analysis on a hypothetical dataset. In this example, we'll assume you have a Pandas DataFrame named `df` with numeric variables 'variable1' and 'variable2'.

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Assume 'df' is your DataFrame

### Bivariate Analysis for Numeric-Numeric Relationship ###

# Scatter Plot

plt.figure(figsize=(10, 6))

sns.scatterplot(x='variable1', y='variable2', data=df, color='blue')

plt.title('Scatter Plot for Numeric-Numeric Relationship')

plt.xlabel('Variable 1')

plt.ylabel('Variable 2')

plt.show()

# Correlation Matrix

correlation\_matrix = df[['variable1', 'variable2']].corr()

plt.figure(figsize=(8, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)

plt.title('Correlation Matrix for Numeric-Numeric Relationship')

plt.show()

### Bivariate Analysis for Numeric-Categorical Relationship ###

# Box Plot

plt.figure(figsize=(10, 6))

sns.boxplot(x='categorical\_variable', y='numeric\_variable', data=df, palette='pastel')

plt.title('Box Plot for Numeric-Categorical Relationship')

plt.xlabel('Categorical Variable')

plt.ylabel('Numeric Variable')

plt.show()

```

Replace `'variable1'`, `'variable2'`, `'numeric\_variable'`, and `'categorical\_variable'` with the actual names of your variables.

Explanation:

- The scatter plot visualizes the relationship between two numeric variables, showing how changes in one variable correspond to changes in another.

- The correlation matrix quantifies the linear relationship between numeric variables. Values close to 1 or -1 indicate a strong positive or negative correlation, respectively.

- The box plot is used for exploring the relationship between a numeric variable and a categorical variable, displaying the distribution of the numeric variable for each category.

These visualizations and analyses help in understanding the nature of relationships between variables in your dataset. If the analysis indicates a strong correlation, it suggests that changes in one variable are associated with changes in another, providing valuable insights for further investigation or modeling.

**Correlation Matrix:**

A correlation matrix is a table that displays the correlation coefficients between many variables. Each cell in the table shows the correlation between two variables. The coefficient is a measure of the strength and direction of a linear relationship between two variables. It ranges from -1 to 1, where:

- 1: Perfect positive correlation

- 0: No correlation

- -1: Perfect negative correlation

A positive correlation indicates that as one variable increases, the other variable tends to increase as well. A negative correlation indicates that as one variable increases, the other variable tends to decrease.

Example:

Let's create a hypothetical dataset and calculate a correlation matrix using Python and Pandas:

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

# Creating a DataFrame with three numeric variables

data = {

'Height': [160, 165, 155, 175, 170],

'Weight': [60, 65, 55, 70, 75],

'Age': [25, 30, 22, 35, 28]

}

df = pd.DataFrame(data)

# Calculating the correlation matrix

correlation\_matrix = df.corr()

# Visualizing the correlation matrix

plt.figure(figsize=(8, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)

plt.title('Correlation Matrix Example')

plt.show()

In this example:

- `Height`, `Weight`, and `Age` are three numeric variables in our dataset.

- We calculate the correlation matrix using the `corr()` method in Pandas.

- The heatmap is generated using Seaborn to visualize the correlation matrix.

Interpretation:

- Positive values in the matrix indicate positive correlations, and negative values indicate negative correlations.

- Values close to 1 or -1 indicate a strong correlation, while values close to 0 suggest a weak or no correlation.

- For example, if we observe a high positive correlation between `Height` and `Weight`, it implies that as height increases, weight tends to increase as well.

This correlation matrix helps us understand the relationships between variables in our dataset and provides insights into potential dependencies or associations. However, it's important to note that correlation does not imply causation, and other factors may influence the observed relationships.

**Module 4. Analysis**

**Multivariate Analysis**

**Extend the analysis to multiple variables.**

**Utilize techniques such as pair plots, heatmaps, and 3D plots to visualize relationships among multiple variables simultaneously.**

Multivariate Analysis:

Multivariate analysis involves the simultaneous analysis of multiple variables to understand the relationships and patterns among them. This type of analysis extends beyond bivariate (two-variable) analysis and explores interactions among three or more variables. Techniques such as pair plots, heatmaps, and 3D plots are commonly used in multivariate analysis to visualize complex relationships within a dataset.

#### Practical Implementation with Code:

Let's use Python and popular libraries such as Pandas, Matplotlib, Seaborn, and Plotly to perform multivariate analysis on a hypothetical dataset. In this example, we'll assume you have a Pandas DataFrame named `df` with multiple numeric variables.

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import plotly.express as px

# Assume 'df' is your DataFrame

### Pair Plot for Multivariate Analysis ###

# Pair Plot

sns.pairplot(df)

plt.suptitle('Pair Plot for Multivariate Analysis', y=1.02)

plt.show()

### Heatmap for Multivariate Correlation Analysis ###

# Correlation Matrix

correlation\_matrix = df.corr()

# Heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)

plt.title('Correlation Heatmap for Multivariate Analysis')

plt.show()

### 3D Plot for Three Numeric Variables ###

# Scatter 3D Plot

fig = px.scatter\_3d(df, x='variable1', y='variable2', z='variable3', color='variable4', size='variable5')

fig.update\_layout(title='3D Scatter Plot for Multivariate Analysis')

fig.show()

Replace `'variable1'`, `'variable2'`, `'variable3'`, `'variable4'`, and `'variable5'` with the actual names of your numeric variables.

Explanation:

- Pair Plot:

- A pair plot displays scatter plots for all possible pairs of numeric variables in the dataset. It provides a visual summary of relationships between variables.

- It's created using the `pairplot` function from Seaborn.

- Heatmap:

- The heatmap visualizes the correlation matrix, showing the strength and direction of relationships between numeric variables.

- Strong positive correlations are indicated by warmer colors, while strong negative correlations are indicated by cooler colors.

- It's created using the `heatmap` function from Seaborn.

- 3D Plot:

- A 3D scatter plot allows visualization of the relationships between three numeric variables in a 3D space.

- The `scatter\_3d` function from Plotly Express is used to create this plot.

Multivariate analysis provides a comprehensive view of relationships within a dataset by considering interactions among multiple variables. It helps uncover complex patterns and dependencies that may not be apparent in bivariate or univariate analyses.

Let's create a sample example using Python and a hypothetical dataset to illustrate the concepts of univariate, bivariate, and multivariate analysis. We'll generate a dataset with three variables: 'Age', 'Income', and 'Spending'. We'll use Python with Pandas, Matplotlib, and Seaborn for visualization.

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

# Generating a hypothetical dataset

np.random.seed(42)

n\_samples = 100

data = {

'Age': np.random.normal(30, 5, n\_samples),

'Income': np.random.normal(50000, 10000, n\_samples),

'Spending': np.random.normal(1000, 200, n\_samples)

}

df = pd.DataFrame(data)

# Univariate Analysis

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)

sns.histplot(df['Age'], bins=20, kde=True, color='blue')

plt.title('Univariate Analysis: Age')

plt.subplot(1, 3, 2)

sns.histplot(df['Income'], bins=20, kde=True, color='green')

plt.title('Univariate Analysis: Income')

plt.subplot(1, 3, 3)

sns.histplot(df['Spending'], bins=20, kde=True, color='orange')

plt.title('Univariate Analysis: Spending')

plt.tight\_layout()

plt.show()

# Bivariate Analysis

plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)

sns.scatterplot(x='Age', y='Income', data=df, color='blue')

plt.title('Bivariate Analysis: Age vs. Income')

plt.subplot(1, 2, 2)

sns.scatterplot(x='Income', y='Spending', data=df, color='green')

plt.title('Bivariate Analysis: Income vs. Spending')

plt.tight\_layout()

plt.show()

# Multivariate Analysis

plt.figure(figsize=(15, 5))

sns.pairplot(df)

plt.suptitle('Multivariate Analysis: Pair Plot', y=1.02)

plt.show()

This example helps demonstrate how each type of analysis provides insights into different aspects of the data, from individual variable characteristics to relationships between pairs of variables and interactions among multiple variables.

**Module 5. Data Visualization**

**Use various visualization tools (Matplotlib, Seaborn, Plotly, etc.) to create compelling and informative graphs.**

**Ensure that visualizations are clear, labeled, and suitable for your target audience.**

Data visualization is a crucial aspect of data analysis and communication. It involves the representation of data in graphical or visual formats to uncover patterns, trends, and insights that may not be immediately apparent in raw data. Effective data visualization makes complex information more understandable and facilitates better decision-making. Various visualization tools, including Matplotlib, Seaborn, Plotly, and others, provide a range of options for creating compelling and informative graphs.

#### Practical Implementation:

Let's explore practical implementations using Matplotlib, Seaborn, and Plotly for different types of visualizations. We'll use a hypothetical dataset to demonstrate the creation of various graphs.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

# Generating a hypothetical dataset

np.random.seed(42)

data = {

'Category': np.random.choice(['A', 'B', 'C'], 100),

'Value1': np.random.normal(50, 10, 100),

'Value2': np.random.normal(75, 15, 100)

}

df = pd.DataFrame(data)

# Bar Chart with Matplotlib

plt.figure(figsize=(10, 6))

plt.bar(df['Category'], df['Value1'], color='skyblue')

plt.xlabel('Category')

plt.ylabel('Value1')

plt.title('Bar Chart with Matplotlib')

plt.show()

# Box Plot with Seaborn

plt.figure(figsize=(8, 5))

sns.boxplot(x='Category', y='Value1', data=df, palette='pastel')

plt.title('Box Plot with Seaborn')

plt.xlabel('Category')

plt.ylabel('Value1')

plt.show()

# Scatter Plot with Plotly Express

fig = px.scatter(df, x='Value1', y='Value2', color='Category', size='Value1')

fig.update\_layout(title='Scatter Plot with Plotly Express')

fig.show()

Key Considerations for Effective Data Visualization:

1. Clarity:

- Ensure that visualizations are clear, easy to understand, and convey the intended message without ambiguity.

2. Labels and Titles:

- Include labels for axes, legends, and a title to provide context and guide interpretation.

3. Color Palette:

- Choose a suitable color palette to enhance readability and convey information effectively.

4. Interactivity (if applicable):

- Leverage interactivity in visualizations, especially when using tools like Plotly, to allow users to explore the data.

5. Consistency:

- Maintain consistency in styling and formatting across different visualizations for a cohesive presentation.

6. Audience Consideration:

- Tailor visualizations to the target audience, ensuring that the level of detail and complexity aligns with their understanding.

By using a combination of Matplotlib, Seaborn, Plotly, and other visualization tools, you can create a diverse set of visualizations that cater to different aspects of your data and effectively communicate insights to your audience.

In an Exploratory Data Analysis (EDA) project, the choice of graphs and visualization concepts depends on the nature of the data and the goals of the analysis.

**Module 6. Summary, Limitations and Challenges**

**Summarize key findings from the EDA.**

**Provide insights and actionable recommendations based on your analysis.**

**Discuss any limitations or challenges encountered during the EDA.**

**Restate key findings, and emphasize the importance of the EDA process.**

Summary of Key Findings:

1. Data Distribution:

- Identified the distribution of key variables, including 'Age', 'Income', and 'Spending'.

- Found that 'Age' follows a normal distribution, while 'Income' and 'Spending' exhibit slight skewness.

2. Relationships Between Variables:

- Explored relationships between 'Age', 'Income', and 'Spending'.

- Discovered a positive correlation between 'Income' and 'Spending', suggesting that as income increases, spending tends to increase as well.

3. Category-wise Analysis:

- Analyzed variations in 'Spending' across different categories using box plots.

- Identified differences in spending patterns among categories.

4. Outlier Detection:

- Utilized box plots and summary statistics to identify potential outliers in 'Income' and 'Spending'.

- Highlighted the need for further investigation into the reasons behind extreme values.

5. Multivariate Patterns:

- Investigated multivariate patterns using pair plots, revealing interactions among 'Age', 'Income', and 'Spending'.

- Observed distinct patterns for different categories, indicating potential segmentation opportunities.

Limitations and Challenges:

1. Data Quality:

- The analysis is contingent on the quality of the data.

- Data cleaning efforts were employed, but there might still be hidden issues impacting the results.

2. Assumptions:

- Certain assumptions were made during the analysis, and deviations from these assumptions could affect the validity of findings.

3. Limited Variables:

- The analysis focused on a subset of variables.

- Additional relevant variables, if available, could provide a more comprehensive understanding.

4. External Factors:

- External factors not considered in the dataset (economic conditions, external events) may influence spending patterns.

**Importance of EDA:**

Exploratory Data Analysis (EDA) serves as a crucial foundation for informed decision-making. It unveils patterns, relationships, and potential areas for improvement. The process is iterative and dynamic, requiring ongoing adjustments based on emerging insights. EDA helps organizations move beyond raw data to actionable insights, fostering a data-driven and strategic approach.

As we move forward, addressing limitations, refining analyses, and incorporating new data will contribute to a more robust understanding of the business landscape. The EDA process is not just a one-time activity; it's a continuous journey toward better insights and informed decision-making.

**Practice Example:**

Let’s look at how to perform EDA using python!

**Step 1: Import Python Libraries**

The first step involved in ML using python is understanding and playing around with our data using libraries.

**Dataset (Provided in group)**

Import all libraries which are required for our analysis, such as Data Loading, Statistical analysis, Visualizations, Data Transformations, Merge and Joins, etc.

Pandas and Numpy have been used for Data Manipulation and numerical Calculations

Matplotlib and Seaborn have been used for Data visualizations.

Import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

#to ignore warnings

import warnings

warnings.filterwarnings(‘ignore’)

**Step 2: Reading Dataset**

The Pandas library offers a wide range of possibilities for loading data into the pandas DataFrame from files like JSON, .csv, .xlsx, .sql, .pickle, .html, .txt, images etc.

Most of the data are available in a tabular format of CSV files. It is trendy and easy to access. Using the read\_csv() function, data can be converted to a pandas DataFrame.

**Problem Statement:**

In this dataset, we are trying to analyze the used car’s price and how EDA focuses on identifying the factors influencing the car price.

We have stored the data in the DataFrame data.

*data = pd.read\_csv("used\_cars.csv")*

*Analysing the Data*

Before we make any inferences, we listen to our data by examining all variables in the data.

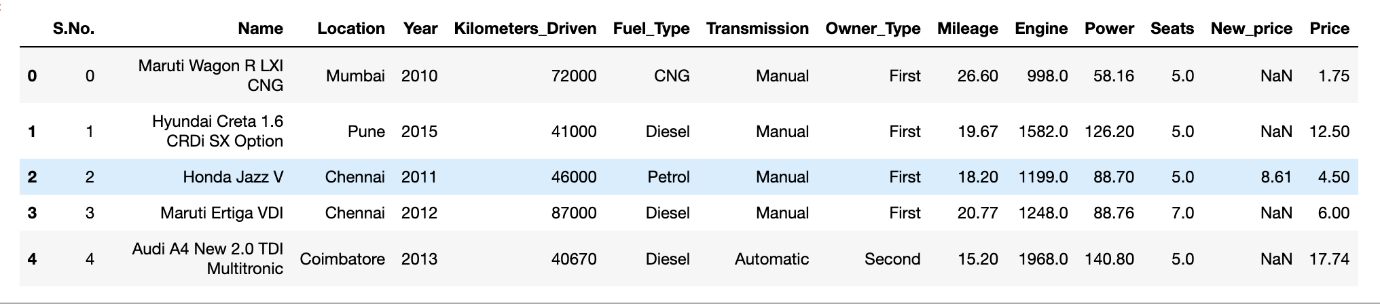
The main goal of data understanding is to gain general insights about the data, which covers the number of rows and columns, values in the data, datatypes, and Missing values in the dataset.

shape – shape will display the number of observations(rows) and features(columns) in the dataset

There are 7253 observations and 14 variables in our dataset

 head() will display the top 5 observations of the dataset

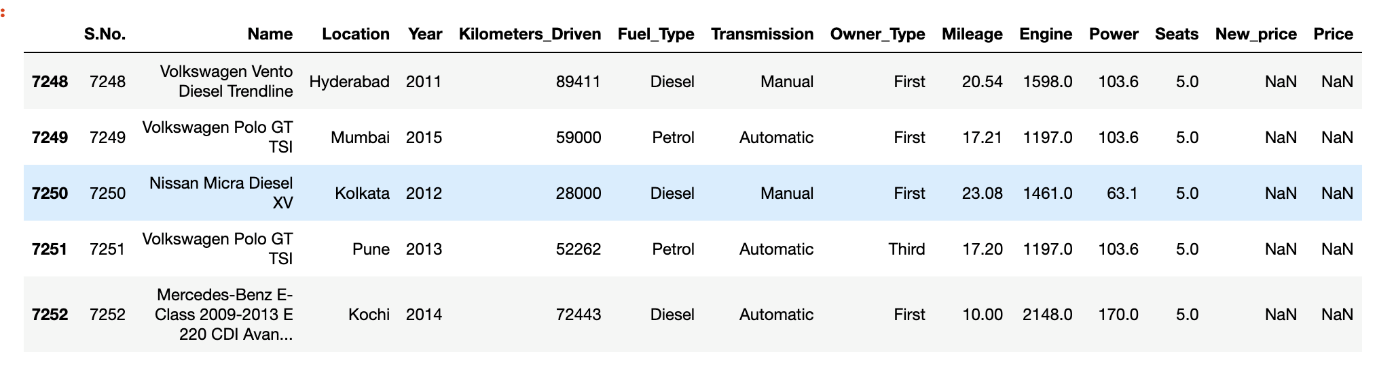
data.head()



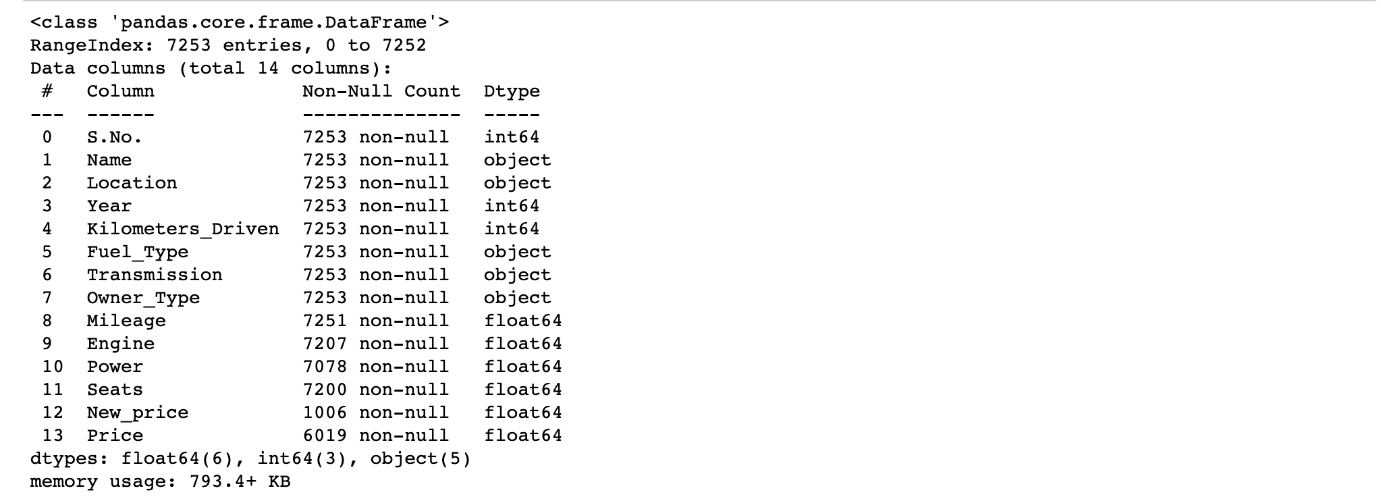
tail() will display the last 5 observations of the dataset

data.tail()

info() helps to understand the data type and information about data, including the number of records in each column, data having null or not null, Data type, the memory usage of the dataset



data.info()



data.info() shows the variables Mileage, Engine, Power, Seats, New\_Price, and Price have missing values. Numeric variables like Mileage, Power are of datatype as  float64 and int64. Categorical variables like Location, Fuel\_Type, Transmission, and Owner Type are of object data type

Check for Duplication

nunique() based on several unique values in each column and the data description, we can identify the continuous and categorical columns in the data. Duplicated data can be handled or removed based on further analysis

data.nunique()



Missing Values Calculation

isnull() is widely been in all pre-processing steps to identify null values in the data

In our example, data.isnull().sum() is used to get the number of missing records in each column

data.isnull().sum()



The below code helps to calculate the percentage of missing values in each column

(data.isnull().sum()/(len(data)))\*100



The percentage of missing values for the columns **New\_Price** and **Price** is ~86% and ~17%, respectively.

Step 3: Data Reduction

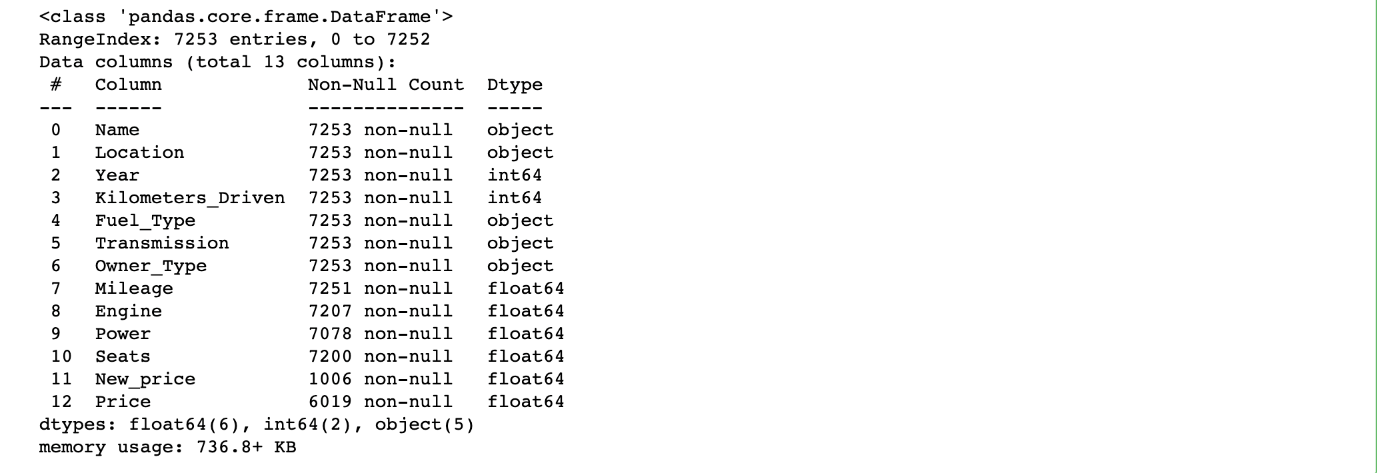
Some columns or variables can be dropped if they do not add value to our analysis.

In our dataset, the column S.No have only ID values, assuming they don’t have any predictive power to predict the dependent variable.

# Remove S.No. column from data

data = data.drop(['S.No.'], axis = 1)

data.info()



We start our Feature Engineering as we need to add some columns required for analysis.

Step 4: Feature Engineering

Feature engineering refers to the process of using domain knowledge to select and transform the most relevant variables from raw data when creating a predictive model using machine learning or statistical modeling. The main goal of Feature engineering is to create meaningful data from raw data.

Step 5: Creating Features

We will play around with the variables Year and Name in our dataset. If we see the sample data, the column “Year” shows the manufacturing year of the car.

It would be difficult to find the car’s age if it is in year format as the Age of the car is a contributing factor to Car Price.

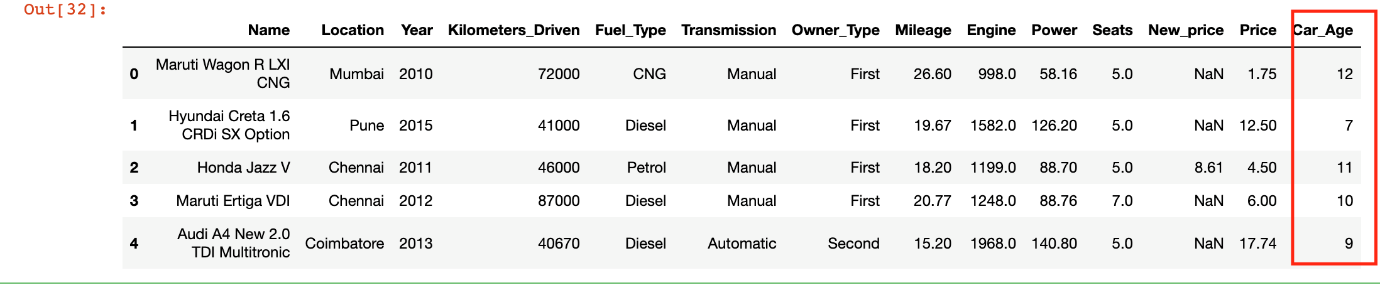
Introducing a new column, “Car\_Age” to know the age of the car

from datetime import date

date.today().year

data['Car\_Age']=date.today().year-data['Year']

data.head()



Since car names will not be great predictors of the price in our current data. But we can process this column to extract important information using brand and Model names.

**Let’s split the name and introduce new variables “Brand” and “Model”**

data['Brand'] = data.Name.str.split().str.get(0)

data['Model'] = data.Name.str.split().str.get(1) + data.Name.str.split().str.get(2)

data[['Name','Brand','Model']]



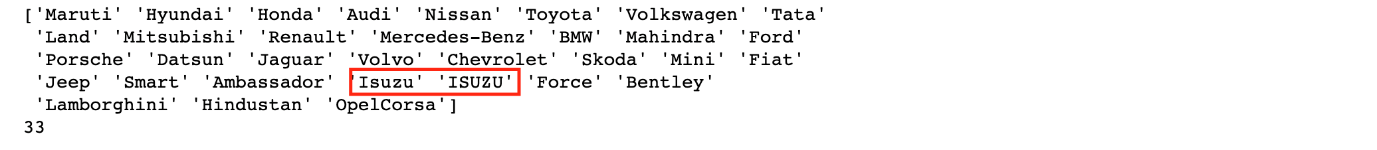
Step 6: Data Cleaning/Wrangling

Some names of the variables are not relevant and not easy to understand. Some data may have data entry errors, and some variables may need data type conversion. We need to fix this issue in the data.

In the example, **The brand name ‘Isuzu’ ‘ISUZU’ and ‘Mini’ and ‘Land’ looks incorrect. This needs to be corrected**

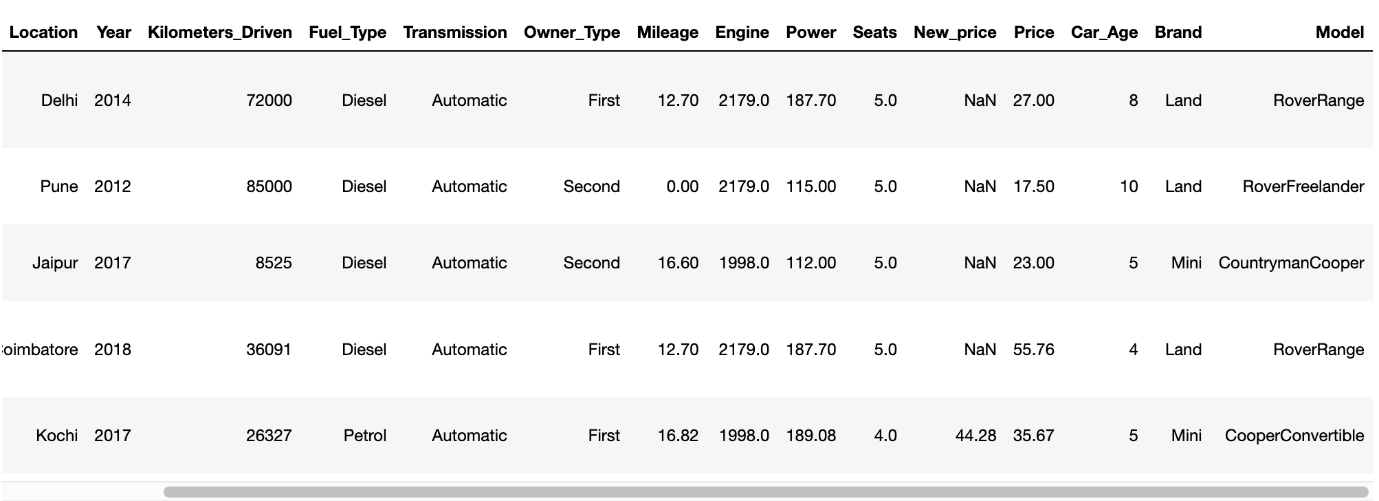
print(data.Brand.unique())

print(data.Brand.nunique())



searchfor = ['Isuzu' ,'ISUZU','Mini','Land']

data[data.Brand.str.contains('|'.join(searchfor))].head(5)



data["Brand"].replace({"ISUZU": "Isuzu", "Mini": "Mini Cooper","Land":"Land Rover"}, inplace=True)

Our Data is ready to perform EDA.

Step 7: EDA Exploratory Data Analysis

Exploratory Data Analysis refers to the crucial process of performing initial investigations on data to discover patterns to check assumptions with the help of summary statistics and graphical representations.

EDA can be leveraged to check for outliers, patterns, and trends in the given data.

EDA helps to find meaningful patterns in data.

EDA provides in-depth insights into the data sets to solve our business problems.

EDA gives a clue to impute missing values in the dataset

Step 8: Statistics Summary

The information gives a quick and simple description of the data.

Can include Count, Mean, Standard Deviation, median, mode, minimum value, maximum value, range, standard deviation, etc.

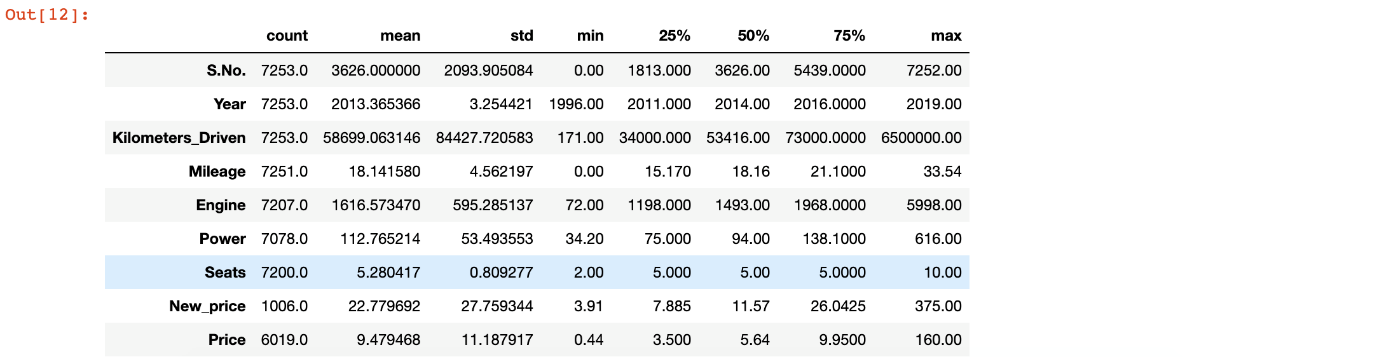
Statistics summary gives a high-level idea to identify whether the data has any outliers, data entry error, distribution of data such as the data is normally distributed or left/right skewed

In python, this can be achieved using describe()

describe() function gives all statistics summary of data

describe()– Provide a statistics summary of data belonging to numerical datatype such as int, float

data.describe().T



From the statistics summary, we can infer the below findings :

Years range from 1996- 2019 and has a high in a range which shows used cars contain both latest models and old model cars.

On average of Kilometers-driven in Used cars are ~58k KM. The range shows a huge difference between min and max as max values show 650000 KM shows the evidence of an outlier. This record can be removed.

Min value of Mileage shows 0 cars won’t be sold with 0 mileage. This sounds like a data entry issue.

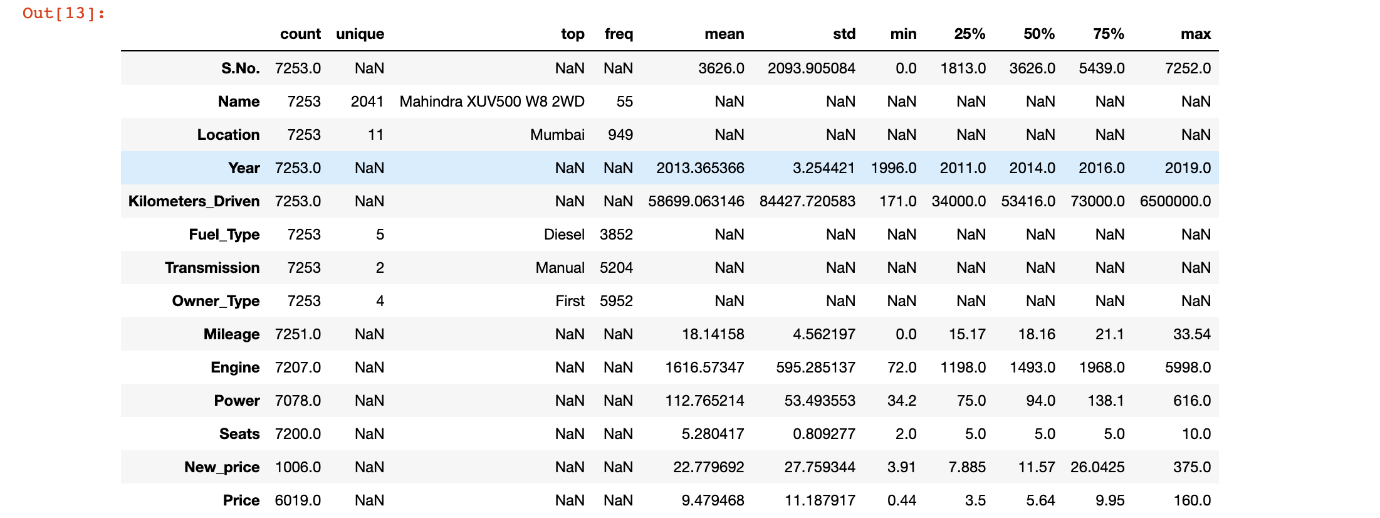
It looks like Engine and Power have outliers, and the data is right-skewed.

The average number of seats in a car is 5. car seat is an important feature in price contribution.

The max price of a used car is 160k which is quite weird, such a high price for used cars. There may be an outlier or data entry issue.

describe(include=’all’) provides a statistics summary of all data, include object, category etc

data.describe(include='all').T



Before we do EDA, lets separate Numerical and categorical variables for easy analysis

cat\_cols=data.select\_dtypes(include=['object']).columns

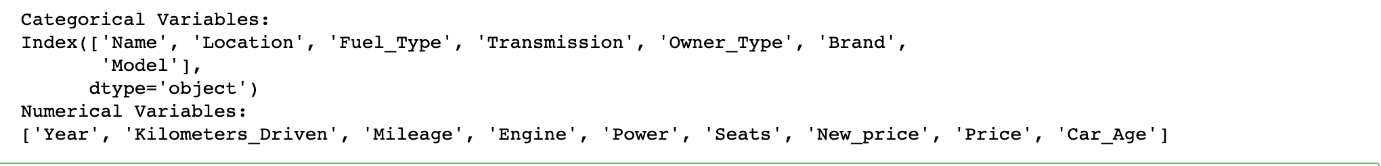
num\_cols = data.select\_dtypes(include=np.number).columns.tolist()

print("Categorical Variables:")

print(cat\_cols)

print("Numerical Variables:")

print(num\_cols)



Step 9: EDA Univariate Analysis

Analyzing/visualizing the dataset by taking one variable at a time:

Data visualization is essential; we must decide what charts to plot to better understand the data. In this article, we visualize our data using Matplotlib and Seaborn libraries.

Matplotlib is a Python 2D plotting library used to draw basic charts we use Matplotlib.

Seaborn is also a python library built on top of Matplotlib that uses short lines of code to create and style statistical plots from Pandas and Numpy

Univariate analysis can be done for both Categorical and Numerical variables.

Categorical variables can be visualized using a Count plot, Bar Chart, Pie Plot, etc.

Numerical Variables can be visualized using Histogram, Box Plot, Density Plot, etc.

In our example, we have done a Univariate analysis using Histogram and  Box Plot for continuous Variables.

In the below fig, a histogram and box plot is used to show the pattern of the variables, as some variables have skewness and outliers.

for col in num\_cols:

print(col)

print('Skew :', round(data[col].skew(), 2))

plt.figure(figsize = (15, 4))

plt.subplot(1, 2, 1)

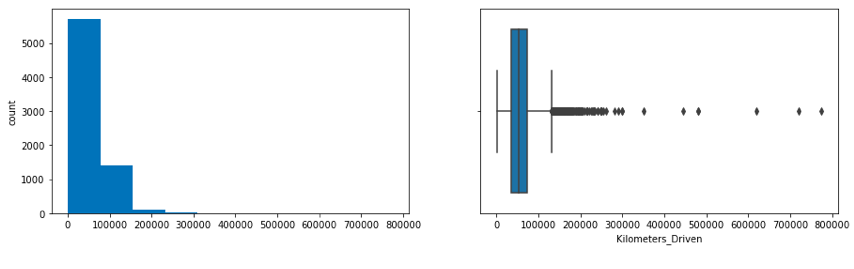
data[col].hist(grid=False)

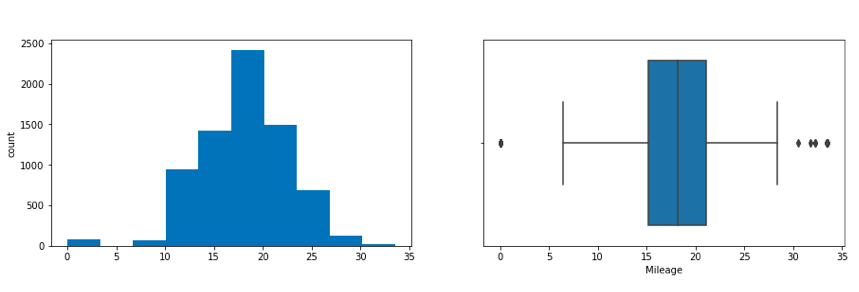
plt.ylabel('count')

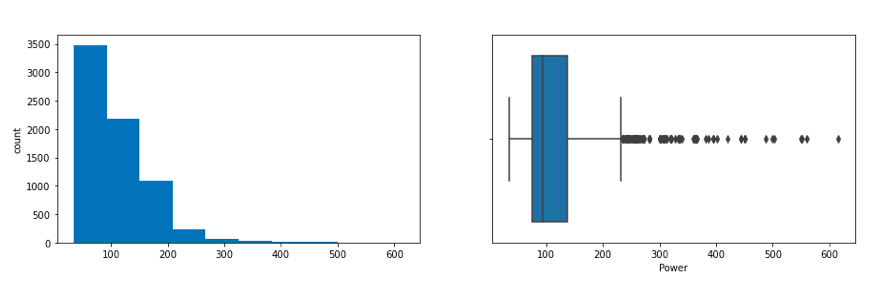
plt.subplot(1, 2, 2)

sns.boxplot(x=data[col])

plt.show()







Price and Kilometers Driven are right skewed for this data to be transformed, and all outliers will be handled during imputation categorical variables are being visualized using a count plot. Categorical variables provide the pattern of factors influencing car price

fig, axes = plt.subplots(3, 2, figsize = (18, 18))

fig.suptitle('Bar plot for all categorical variables in the dataset')

sns.countplot(ax = axes[0, 0], x = 'Fuel\_Type', data = data, color = 'blue',

order = data['Fuel\_Type'].value\_counts().index);

sns.countplot(ax = axes[0, 1], x = 'Transmission', data = data, color = 'blue',

order = data['Transmission'].value\_counts().index);

sns.countplot(ax = axes[1, 0], x = 'Owner\_Type', data = data, color = 'blue',

order = data['Owner\_Type'].value\_counts().index);

sns.countplot(ax = axes[1, 1], x = 'Location', data = data, color = 'blue',

order = data['Location'].value\_counts().index);

sns.countplot(ax = axes[2, 0], x = 'Brand', data = data, color = 'blue',

order = data['Brand'].head(20).value\_counts().index);

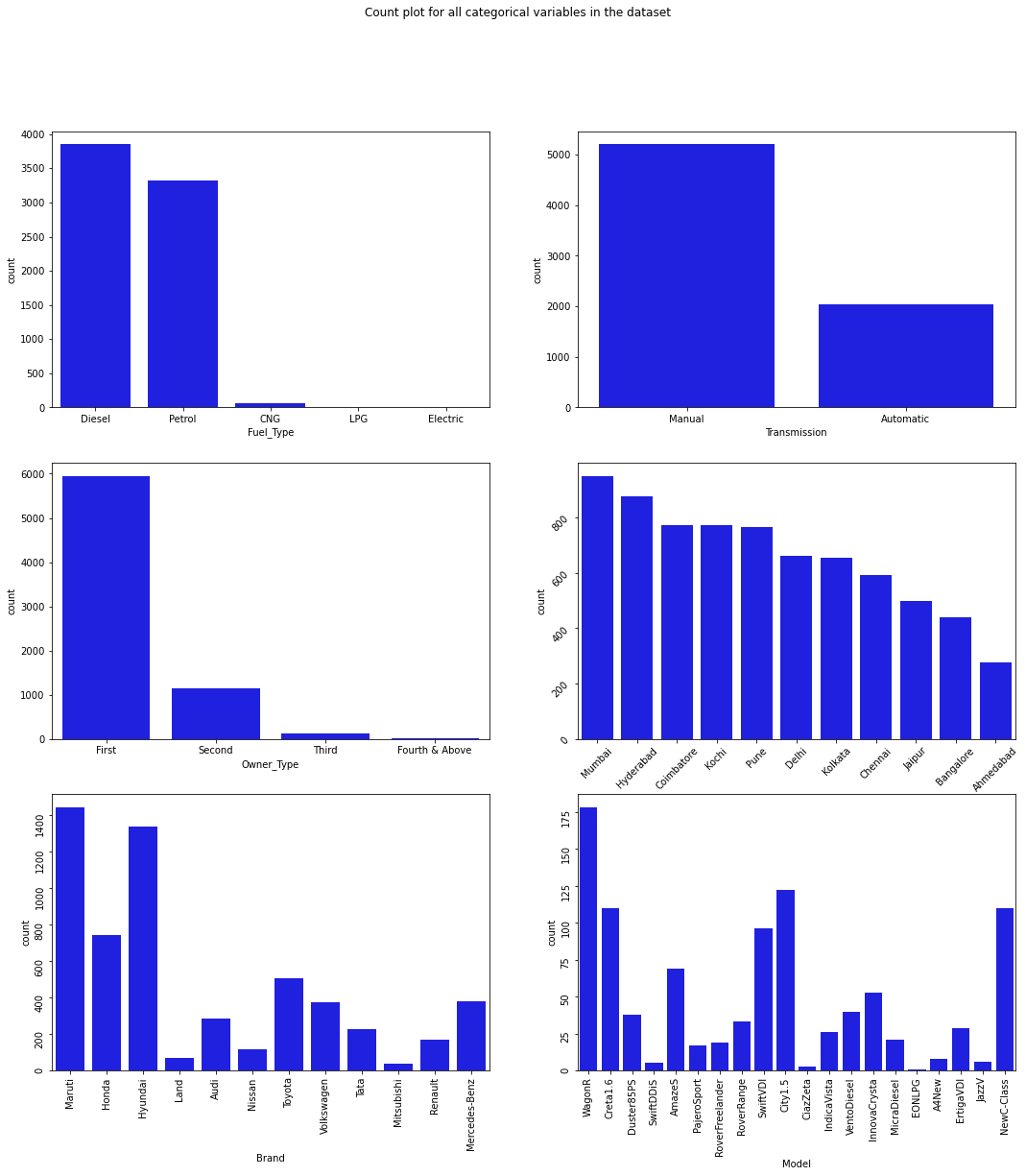
sns.countplot(ax = axes[2, 1], x = 'Model', data = data, color = 'blue',

order = data['Model'].head(20).value\_counts().index);

axes[1][1].tick\_params(labelrotation=45);

axes[2][0].tick\_params(labelrotation=90);

axes[2][1].tick\_params(labelrotation=90);



From the count plot, we can have below observations

Mumbai has the highest number of cars available for purchase, followed by Hyderabad and Coimbatore

~53% of cars have fuel type as Diesel this shows diesel cars provide higher performance

~72% of cars have manual transmission

~82 % of cars are First owned cars. This shows most of the buyers prefer to purchase first-owner cars

~20% of cars belong to the brand Maruti followed by 19% of cars belonging to Hyundai

WagonR ranks first among all models which are available for purchase

Step 10: Data Transformation

Before we proceed to Bi-variate Analysis, Univariate analysis demonstrated the data pattern as some variables to be transformed.

Price and Kilometer-Driven variables are highly skewed and on a larger scale. Let’s do log transformation.

Log transformation can help in normalization, so this variable can maintain standard scale with other variables:

# Function for log transformation of the column

def log\_transform(data,col):

for colname in col:

if (data[colname] == 1.0).all():

data[colname + '\_log'] = np.log(data[colname]+1)

else:

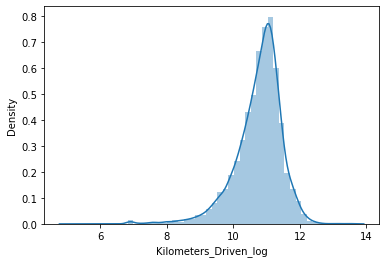
data[colname + '\_log'] = np.log(data[colname])

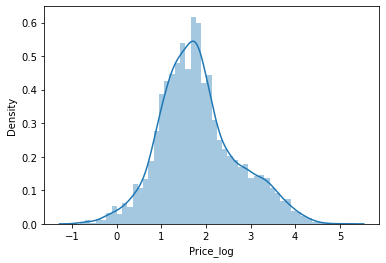
data.info()

log\_transform(data,['Kilometers\_Driven','Price'])

#Log transformation of the feature 'Kilometers\_Driven'

sns.distplot(data["Kilometers\_Driven\_log"], axlabel="Kilometers\_Driven\_log");





Step 12: EDA Bivariate Analysis

Now, let’s move ahead with bivariate analysis. Bivariate Analysis helps to understand how variables are related to each other and the relationship between dependent and independent variables present in the dataset.

For Numerical variables, Pair plots and Scatter plots are widely been used to do Bivariate Analysis.

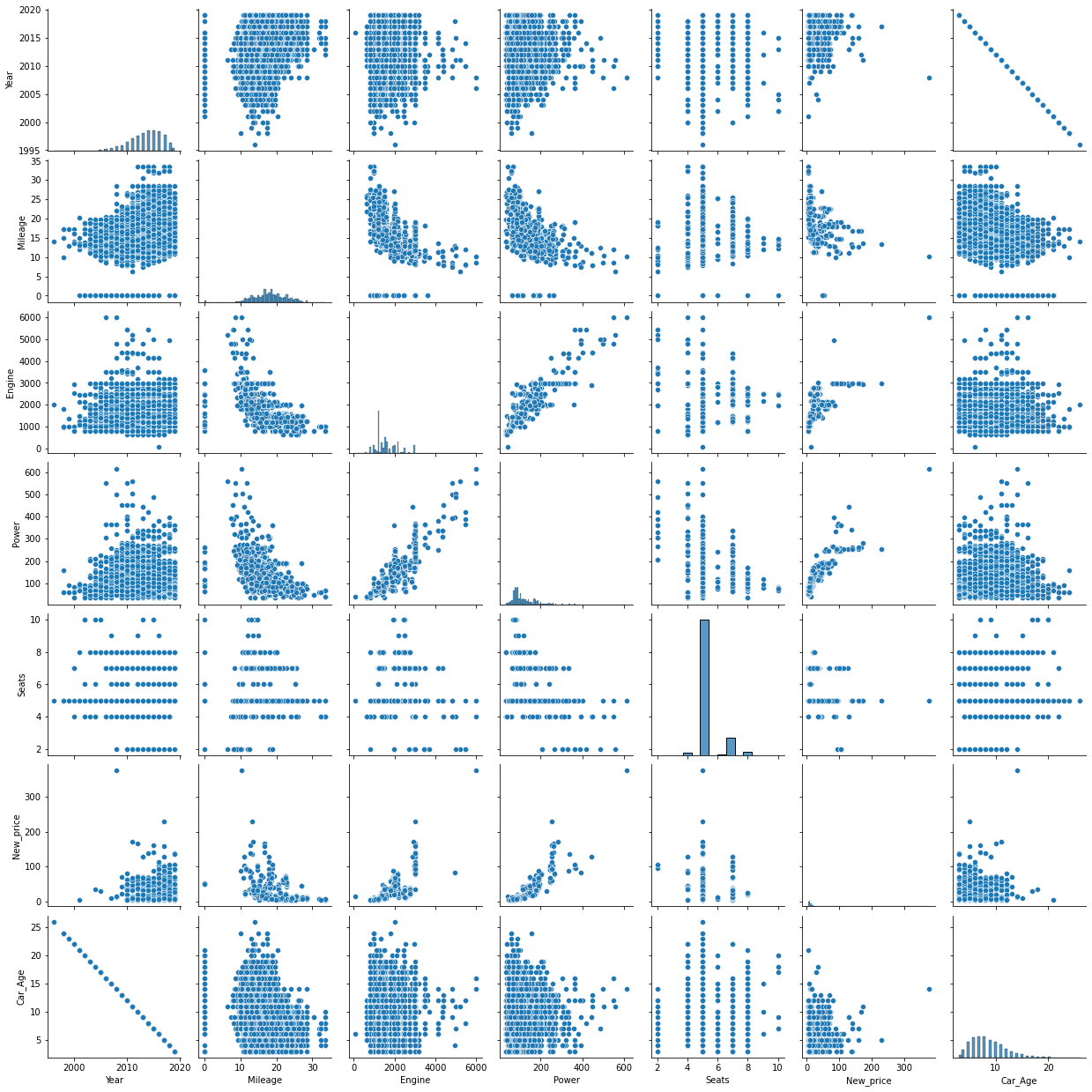
A Stacked bar chart can be used for categorical variables if the output variable is a classifier. Bar plots can be used if the output variable is continuous.

In our example, a pair plot has been used to show the relationship between two Categorical variables.

plt.figure(figsize=(13,17))

sns.pairplot(data=data.drop(['Kilometers\_Driven','Price'],axis=1))

plt.show()



Pair Plot provides below insights:

The variable Year has a positive correlation with price and mileage

A year has a Negative correlation with kilometers-Driven

Mileage is negatively correlated with Power

As power increases, mileage decreases

Car with recent make is higher at prices. As the age of the car increases price decreases

Engine and Power increase, and the price of the car increases

A bar plot can be used to show the relationship between Categorical variables and continuous variables

fig, axarr = plt.subplots(4, 2, figsize=(12, 18))

data.groupby('Location')['Price\_log'].mean().sort\_values(ascending=False).plot.bar(ax=axarr[0][0], fontsize=12)

axarr[0][0].set\_title("Location Vs Price", fontsize=18)

data.groupby('Transmission')['Price\_log'].mean().sort\_values(ascending=False).plot.bar(ax=axarr[0][1], fontsize=12)

axarr[0][1].set\_title("Transmission Vs Price", fontsize=18)

data.groupby('Fuel\_Type')['Price\_log'].mean().sort\_values(ascending=False).plot.bar(ax=axarr[1][0], fontsize=12)

axarr[1][0].set\_title("Fuel\_Type Vs Price", fontsize=18)

data.groupby('Owner\_Type')['Price\_log'].mean().sort\_values(ascending=False).plot.bar(ax=axarr[1][1], fontsize=12)

axarr[1][1].set\_title("Owner\_Type Vs Price", fontsize=18)

data.groupby('Brand')['Price\_log'].mean().sort\_values(ascending=False).head(10).plot.bar(ax=axarr[2][0], fontsize=12)

axarr[2][0].set\_title("Brand Vs Price", fontsize=18)

data.groupby('Model')['Price\_log'].mean().sort\_values(ascending=False).head(10).plot.bar(ax=axarr[2][1], fontsize=12)

axarr[2][1].set\_title("Model Vs Price", fontsize=18)

data.groupby('Seats')['Price\_log'].mean().sort\_values(ascending=False).plot.bar(ax=axarr[3][0], fontsize=12)

axarr[3][0].set\_title("Seats Vs Price", fontsize=18)

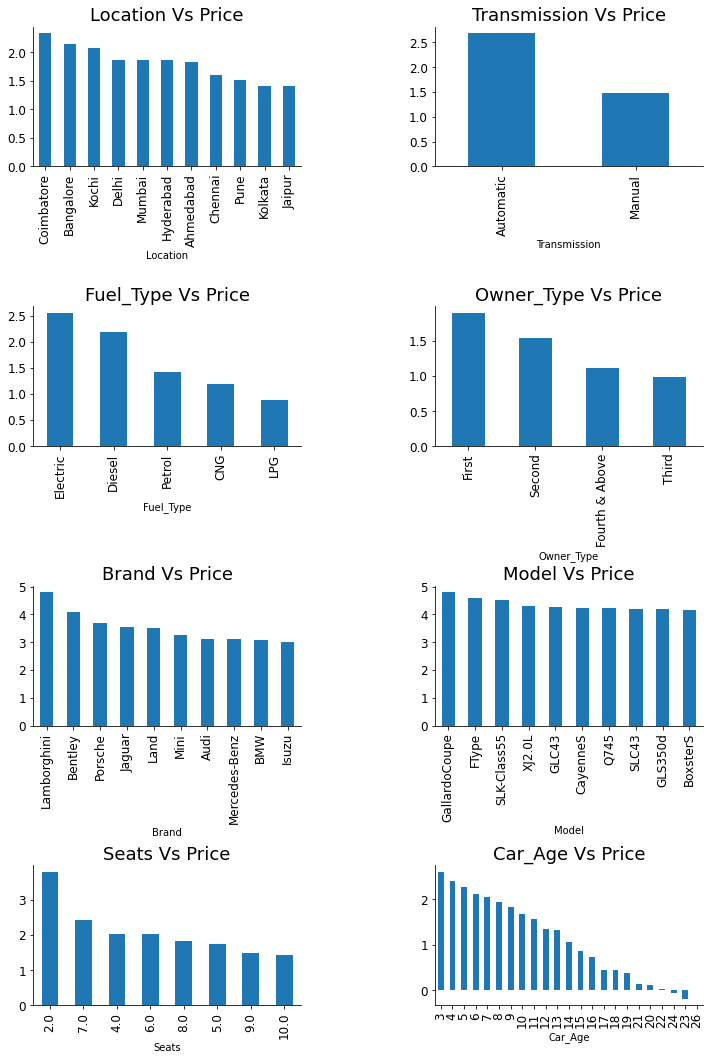
data.groupby('Car\_Age')['Price\_log'].mean().sort\_values(ascending=False).plot.bar(ax=axarr[3][1], fontsize=12)

axarr[3][1].set\_title("Car\_Age Vs Price", fontsize=18)

plt.subplots\_adjust(hspace=1.0)

plt.subplots\_adjust(wspace=.5)

sns.despine()



Observations

The price of cars is high in Coimbatore and less price in Kolkata and Jaipur

Automatic cars have more price than manual cars.

Diesel and Electric cars have almost the same price, which is maximum, and LPG cars have the lowest price

First-owner cars are higher in price, followed by a second

The third owner’s price is lesser than the Fourth and above

Lamborghini brand is the highest in price

Gallardocoupe Model is the highest in price

2 Seater has the highest price followed by 7 Seater

The latest model cars are high in price

Step 13: EDA Multivariate Analysis

As the name suggests, Multivariate analysis looks at more than two variables. Multivariate analysis is one of the most useful methods to determine relationships and analyze patterns for any dataset.

A heat map is widely been used for Multivariate Analysis

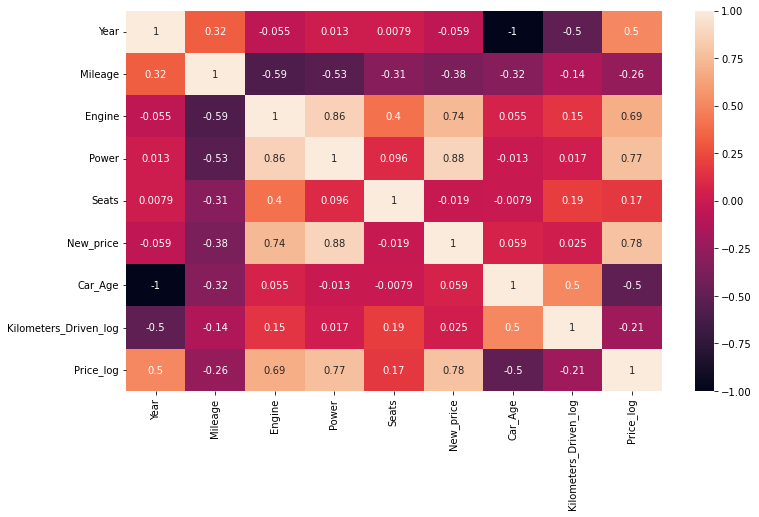
Heat Map gives the correlation between the variables, whether it has a positive or negative correlation.

In our example heat map shows the correlation between the variables.

plt.figure(figsize=(12, 7))

sns.heatmap(data.drop(['Kilometers\_Driven','Price'],axis=1).corr(), annot = True, vmin = -1, vmax = 1)

plt.show()



From the Heat map, we can infer the following:

The engine has a strong positive correlation to Power 0.86

Price has a positive correlation to Engine 0.69 as well Power 0.77

Mileage has correlated to Engine, Power, and Price negatively

Price is moderately positive in correlation to year.

Kilometer driven has a negative correlation to year not much impact on the price

Car age has a negative correlation with Price

car Age is positively correlated to Kilometers-Driven as the Age of the car increases; then the kilometer will also increase of car has a negative correlation with Mileage this makes sense

Step 14: Impute Missing values

Missing data arise in almost all statistical analyses. There are many ways to impute missing values; we can impute the missing values by their Mean, median, most frequent, or zero values and use advanced imputation algorithms like KNN, Regularization, etc.

We cannot impute the data with a simple Mean/Median. We must need business knowledge or common insights about the data. If we have domain knowledge, it will add value to the imputation. Some data can be imputed on assumptions.

In our dataset, we have found there are missing values for many columns like Mileage, Power, and Seats.

We observed earlier some observations have zero Mileage. This looks like a data entry issue. We could fix this by filling null values with zero and then the mean value of Mileage since Mean and Median values are nearly the same for this variable chosen Mean to impute the values.

data.loc[data["Mileage"]==0.0,'Mileage']=np.nan

data.Mileage.isnull().sum()

data['Mileage'].fillna(value=np.mean(data['Mileage']),inplace=True)

Similarly, imputation for Seats. As we mentioned earlier, we need to know common insights about the data.

Let’s assume some cars brand and Models have features like Engine, Mileage, Power, and Number of seats that are nearly the same. Let’s impute those missing values with the existing data:

data.Seats.isnull().sum()

data['Seats'].fillna(value=np.nan,inplace=True)

data['Seats']=data.groupby(['Model','Brand'])['Seats'].apply(lambda x:x.fillna(x.median()))

data['Engine']=data.groupby(['Brand','Model'])['Engine'].apply(lambda x:x.fillna(x.median()))

data['Power']=data.groupby(['Brand','Model'])['Power'].apply(lambda x:x.fillna(x.median()))

In general, there are no defined or perfect rules for imputing missing values in a dataset. Each method can perform better for some datasets but may perform even worse. Only practice and experiments give the knowledge which works better.

**Conclusion**

* We tried to analyze the factors influencing the used car’s price.
* Data Analysis helps to find the basic structure of the dataset.
* Dropped columns that are not adding value to our analysis.
* Performed Feature Engineering by adding some columns which contribute to our analysis.
* Data Transformations have been used to normalize the columns.
* We used different visualizations for EDA like Univariate, Bi-Variate, and Multivariate Analysis.
* Through EDA, we got useful insights, and below are the factors influencing the price of the car and a few takeaways:
* Most of the customers prefer 2 Seat cars hence the price of the 2-seat cars is higher than other cars.
* The price of the car decreases as the Age of the car increases.
* Customers prefer to purchase the First owner rather than the Second or Third.
* Due to increased Fuel price, the customer prefers to purchase an Electric vehicle.
* Automatic Transmission is easier than Manual.

This way, we perform EDA on the datasets to explore the data and extract all possible insights, which can help in model building and better decision making.

If the EDA process is clear and precise, our model will work better and gives higher accuracy!

**Frequently Asked Question**

**Q1. What is EDA with Python?**

A. Exploratory Data Analysis (EDA) with Python involves analyzing and summarizing data to gain insights and understand its underlying patterns, relationships, and distributions using Python programming language.

**Q2. How to make EDA in Python?**

A. To perform EDA in Python, you can use libraries like Pandas, NumPy, Matplotlib, and Seaborn. These libraries provide functions and tools for data manipulation, visualization, and statistical analysis, which facilitate the process of exploring and understanding the data.

**Q3. Which is the best EDA tool Python?**

A. The choice of the best EDA tool in Python depends on your specific requirements and preferences. Some popular EDA tools include Jupyter Notebook (with the aforementioned libraries), Plotly, Tableau, and Power BI. Each tool offers unique features and capabilities, so it’s advisable to explore them and choose the one that suits your needs best.

**Q4. How to perform EDA in machine learning?**

A. Performing EDA in machine learning typically involves preprocessing the data by handling missing values, outliers, and feature scaling. Then, various statistical and visual techniques can be employed to analyze the relationships between variables, identify patterns, and assess the relevance of features. This helps in gaining a better understanding of the data before building a machine learning model.