Exploratory Data Analysis: Concepts, Experiments, and Case Studies

R. Eswar Reddy,
Assistant Professor
Sreenivasa Institute of Technology and Management Studies

December 19, 2025

Contents

Chapter 1

Exploratory Data Analysis Fundamentals

1.1 Understanding Data Science

Data science is an interdisciplinary field that extracts insights from data using scientific methods.

1.2 EXPERIMENT – 1

Download Dataset from Kaggle Websites

Aim

To download the Cars4U dataset from Kaggle websites and load it into Python for basic exploratory data analysis.

Procedure

1. Open the Kaggle website and log in using valid credentials.
2. Access the Cars4U dataset using the given link.
3. Download the dataset ZIP file.
4. Extract the ZIP file to obtain the CSV dataset.
5. Open Python or Jupyter Notebook.

6. Import required libraries such as Pandas and Matplotlib.
7. Load the CSV file into a Pandas DataFrame.
8. View the dataset structure, column names, and data types.
9. Check for missing values and duplicate records.
10. Perform basic exploratory data analysis using statistical summaries and graphs.

Output

The Cars4U dataset was successfully downloaded and loaded into the Pandas DataFrame. The first few records of the dataset are shown below:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	1.75
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12.50
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	4.50
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	6.00
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	17.74

Figure 1.1: Cars4U Dataset Loaded into Pandas DataFrame

Result

The Cars4U dataset was successfully downloaded from Kaggle, extracted, and loaded into a Pandas DataFrame. The dataset structure and initial records were displayed for further exploratory data analysis.

1.3 EXPERIMENT – 2

Installation of Python Libraries for Exploratory Data Analysis

Aim

To install the required Python libraries such as NumPy, Pandas, Matplotlib, Seaborn, and Plotly for performing Exploratory Data Analysis (EDA).

Algorithm

1. Open Command Prompt / Terminal.
2. Verify Python installation by checking the Python version.
3. Use pip (Python package manager) to install required libraries.
4. Install NumPy for numerical computations.
5. Install Pandas for data manipulation and analysis.
6. Install Matplotlib for data visualization.
7. Install Seaborn for advanced statistical visualization.
8. Install Plotly for interactive data visualization.
9. Verify the successful installation of libraries.

Program

Installing libraries individually:

```
pip install numpy
pip install pandas
pip install matplotlib
pip install seaborn
pip install plotly
```

Installing all libraries at once:

```
pip install numpy pandas matplotlib seaborn plotly
```

Output

```
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: plotly in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: python-dateutil>=2.8.2
Requirement already satisfied: pytz>=2020.1
Requirement already satisfied: tzdata>=2022.7
Requirement already satisfied: contourpy>=1.0.1
Requirement already satisfied: cycler>=0.10
Requirement already satisfied: fonttools>=4.22.0
Requirement already satisfied: kiwisolver>=1.3.1
Requirement already satisfied: packaging>=20.0
Requirement already satisfied: pillow>=8
Requirement already satisfied: pyparsing>=2.3.1
Requirement already satisfied: tenacity>=6.2.0
Requirement already satisfied: six>=1.5
```

Result

Thus, the Python libraries NumPy, Pandas, Matplotlib, Seaborn, and Plotly were successfully installed and verified for performing Exploratory Data Analysis.

1.4 EXPERIMENT – 3

NumPy Array Operations

Aim

To perform basic array operations using NumPy and explore built-in NumPy functions.

Algorithm

1. Import the NumPy library.
2. Create one-dimensional and two-dimensional NumPy arrays.
3. Perform arithmetic operations on arrays.
4. Find statistical measures such as sum, mean, minimum, and maximum.
5. Explore array shape, size, and transpose operations.

Program

```
import numpy as np

# Creating 1D array
arr = np.array([10, 20, 30, 40, 50])
print("Array:", arr)

# Array properties
print("Shape:", arr.shape)
print("Size:", arr.size)

# Arithmetic operations
print("Sum:", np.sum(arr))
print("Mean:", np.mean(arr))
print("Max:", np.max(arr))
print("Min:", np.min(arr))

# Creating 2D array
arr2 = np.array([[1,2,3],[4,5,6]])
print("2D Array:\n", arr2)

# Transpose
```

```
22 print("Transpose:\n", arr2.T)
```

Output

The NumPy arrays were created successfully. Basic arithmetic and statistical operations were performed correctly.

Result

Thus, basic NumPy array operations and built-in functions were successfully executed.

1.5 EXPERIMENT – 4

Loading Dataset into Pandas DataFrame

Aim

To load the Cars4U dataset into a Pandas DataFrame and explore its structure.

Algorithm

1. Import the Pandas library.
2. Load the Cars4U CSV file using read_csv().
3. Display the first few rows of the dataset.
4. View dataset shape, column names, and data types.
5. Check for missing values.

Program

```
import pandas as pd

# Load dataset
df = pd.read_csv("cars4u.csv")

# Display first 5 rows
print(df.head())

# Dataset information
print(df.shape)
print(df.columns)
print(df.info())

# Check missing values
print(df.isnull().sum())
```

Output

The Cars4U dataset was successfully loaded into a Pandas DataFrame. Dataset structure, columns, and missing values were displayed. The Cars4U dataset was successfully downloaded and loaded into the Pandas DataFrame. The first few records of the dataset are shown below: The Cars4U dataset was

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	1.75
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12.50
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	4.50
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	6.00
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	17.74

Figure 1.2: Cars4U Dataset Loaded into Pandas DataFrame

successfully downloaded and loaded into the Pandas DataFrame. The first few records of the dataset are shown below:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	1.75
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12.50
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	4.50
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	6.00
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	17.74

Figure 1.3: Cars4U Dataset Loaded into Pandas DataFrame

Dataframe:

```
(7253, 14)
```

print(df.columns):

```
Index(['S.No.', 'Name', 'Location', 'Year', 'Kilometers_Driven', 'Fuel_Ty
       'Transmission', 'Owner_Type', 'Mileage', 'Engine', 'Power', 'Seats
       'New_Price', 'Price'], dtype='object')
```

info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 14 columns):
```

```
Column             Non-Null Count    Data Type
-----------------------------------------------
S.No.              7253              int64
Name               7253              object
Location           7253              object
Year               7253              int64
Kilometers_Driven  7253              int64
Fuel_Type          7253              object
Transmission       7253              object
Owner_Type         7253              object
Mileage            7251              object
Engine             7207              object
Power              7207              object
Seats              7200              float64
New_Price          1006              object
Price              6019              float64

Memory usage: 793.4+ KB
```

Check missing values:

```
S.No.                 0
Name                  0
Location              0
Year                  0
Kilometers_Driven     0
Fuel_Type             0
Transmission          0
Owner_Type            0
Mileage               2
Engine               46
Power                46
Seats                53
New_Price          6247
Price              1234
dtype: int64
```

Result

Thus, the Cars4U dataset was successfully loaded and explored using Pandas.

Chapter 2

Visual Aids for EDA

2.1 EXPERIMENT – 6

2.2 Visualization Techniques on Sample Dataset

Aim

To apply different visualization techniques such as Line Chart, Bar Chart, Scatter Plot, and Bubble Plot on a sample dataset using Python libraries.

Algorithm

1. Import the required libraries: Pandas, Matplotlib, Seaborn.
2. Load a sample dataset (e.g., Cars4U or custom small dataset).
3. Plot a Line Chart to show trends.
4. Plot a Bar Chart to compare categorical values.
5. Plot a Scatter Plot to analyze relationships between variables.
6. Plot a Bubble Plot to visualize three variables simultaneously.

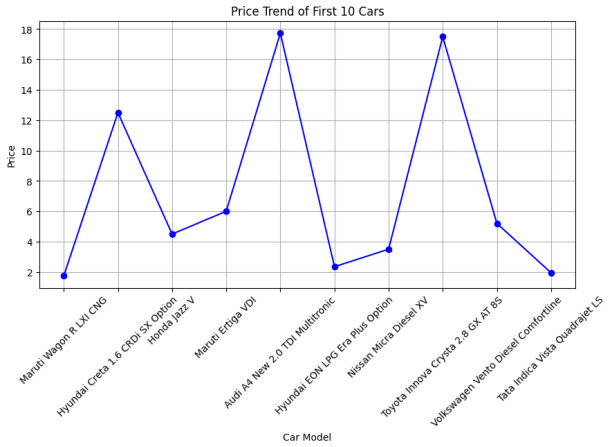
Program

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Line Chart
```

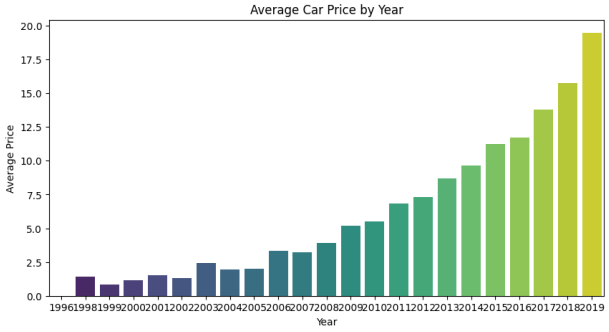
```
# Example: Price trend of first 10 cars
plt.figure(figsize=(10,5))
plt.plot(df['Name'][:10], df['Price'][:10], marker='o', color='blue',
    linestyle='-')
plt.title('Price Trend of First 10 Cars')
plt.xlabel('Car Model')
plt.ylabel('Price')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# 2. Bar Chart
# Example: Average price by Year
avg_price_year = df.groupby('Year')['Price'].mean().reset_index()
plt.figure(figsize=(10,5))
sns.barplot(data=avg_price_year, x='Year', y='Price', palette='viridis')
plt.title('Average Car Price by Year')
plt.xlabel('Year')
plt.ylabel('Average Price')
plt.show()

# 3. Scatter Plot
# Example: Engine size vs Price

# Clean the 'Engine' column to extract numeric values
df['Engine_Size_cc'] = df['Engine'].str.replace(' CC', '', regex=False).
    astype(float)

plt.figure(figsize=(10,6))
sns.scatterplot(data=df, x='Engine_Size_cc', y='Price', hue='Year',
    palette='coolwarm', s=100)
plt.title('Engine Size vs Price')
plt.xlabel('Engine Size (cc)')
plt.ylabel('Price')
plt.show()

# 4. Bubble Plot
# Example: Mileage vs Price, bubble size based on Engine_Size

# Clean the 'Mileage' column to extract numeric values
df['Mileage_kmpl'] = df['Mileage'].str.replace(' km/kg', '', regex=False)
df['Mileage_kmpl'] = df['Mileage_kmpl'].str.replace(' kmpl', '', regex=
    False).astype(float)

plt.figure(figsize=(10,6))
plt.scatter(df['Mileage_kmpl'], df['Price'], s=df['Engine_Size_cc']*10, c=
    df['Year'], cmap='plasma', alpha=0.6, edgecolors='w')
plt.title('Mileage vs Price (Bubble size = Engine Size)')
plt.xlabel('Mileage (km/l)')
plt.ylabel('Price')
plt.colorbar(label='Year')
plt.show()
```

Output

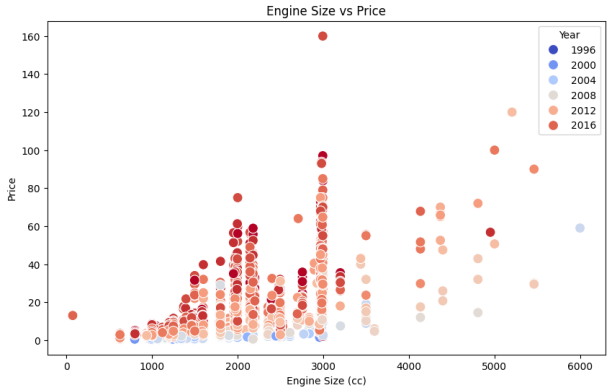
Line chart, bar chart, scatter plot, and bubble plot were successfully generated for the sample dataset.



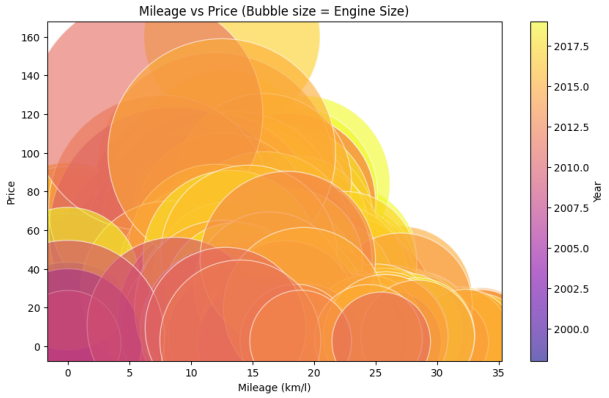
(a) Average Car Price by Year



(b) Engine Size vs Price (colored by Year)



(c) Mileage vs Price
(Bubble size = Engine Size, color = Year)



(d) Price Trend of First 10 Cars

Figure 2.1: Comprehensive analysis of used car pricing patterns

Result

Different visualization techniques were successfully applied and interpreted for a sample dataset.

2.3 EXPERIMENT – 7

2.4 Scatter Plot Using Seaborn for Iris Dataset

Aim

To generate scatter plots for the Iris dataset using Seaborn library to analyze relationships between different flower features.

Algorithm

1. Import Seaborn and Matplotlib libraries.
2. Load the Iris dataset using Seaborn's load_dataset function.
3. Generate scatter plots using sns.scatterplot().
4. Differentiate flower species using color.

Program

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load Iris dataset
iris = sns.load_dataset('iris')

# Display first few rows
print(iris.head())

print ( iris . shape )
print ( iris . columns )

print ( iris . info () )

# ------------------------------
# Scatter Plot: Sepal Length vs Sepal Width
plt.figure(figsize=(8,6))
sns.scatterplot(
    data=iris,
    x='sepal_length',
    y='sepal_width',
    hue='species',      # color points by species
    style='species',    # different markers for species
    s=100               # size of points
```

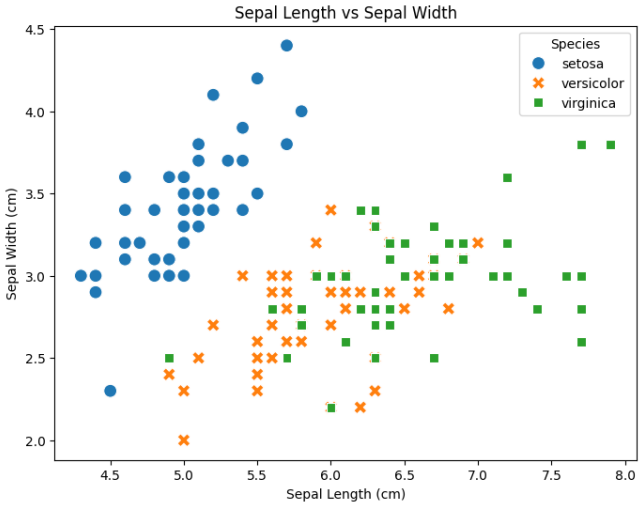
```
)
plt.title('Sepal Length vs Sepal Width')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.legend(title='Species')
plt.show()

# ----------------------------
# Optional: Pairplot (all features)
sns.pairplot(iris, hue='species', height=2.5)
plt.show()
```

Output

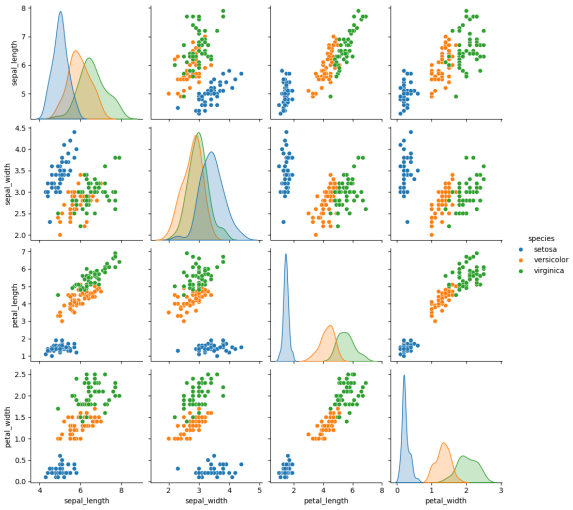
Scatter plot showing relationships between sepal length and width for each Iris species was generated.

```
     sepal_length  sepal_width  petal_length  petal_width    species
0             5.1          3.5           1.4          0.2     setosa
1             4.9          3.0           1.4          0.2     setosa
2             4.7          3.2           1.3          0.2     setosa
3             4.6          3.1           1.5          0.2     setosa
4             5.0          3.6           1.4          0.2     setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

[150 rows x 5 columns]

Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
```

```
---   ------          --------------  -----
 0    sepal_length    150 non-null    float64
 1    sepal_width     150 non-null    float64
 2    petal_length    150 non-null    float64
 3    petal_width     150 non-null    float64
 4    species         150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None
```



(a) Sepal Length vs Sepal Width by Species



(b) Pairplot & Marginal Distributions of Iris Features

Figure 2.2: Visualization of the Iris dataset: bivariate relationship (left) and complete pairwise relationships with distributions (right).

Result

Scatter plots using Seaborn were successfully generated to analyze the Iris dataset.

2.5 EXPERIMENT – 8

2.6 Area, Stacked, Pie, Table, and Pair Plots

Aim

To generate Area Plot, Stacked Plot, Pie Chart, Table Chart, and Pair Plot using a sample dataset.

Algorithm

1. Import Pandas, Matplotlib, and Seaborn.
2. Load sample dataset.
3. Plot an Area Chart using df.plot.area().
4. Plot a Stacked Chart using df.plot.bar(stacked=True).
5. Plot a Pie Chart for categorical variable.
6. Generate Table Chart using plt.table().
7. Generate Pair Plot using sns.pairplot().

Program

```

# ------------------------------
# 1. Area Plot
# For area plot, we can take mean of numeric features per species
iris_mean = iris.groupby('species').mean()
iris_mean.plot(kind='area', figsize=(10,6), alpha=0.5)
plt.title('Area Plot of Average Features per Species')
plt.ylabel('Value (cm)')
plt.xlabel('Species')
plt.xticks(rotation=0)
plt.show()

# ------------------------------
# 2. Stacked Plot
iris_mean.plot(kind='bar', stacked=True, figsize=(10,6))
plt.title('Stacked Bar Plot of Average Features per Species')
plt.ylabel('Value (cm)')
plt.xlabel('Species')
```

```
plt.show()

# ----------------------------
# 3. Pie Chart
# Show proportion of each species in dataset
species_counts = iris['species'].value_counts()
plt.figure(figsize=(7,7))
plt.pie(species_counts, labels=species_counts.index, autopct='%1.1f%%',
    startangle=140, colors=['skyblue', 'lightgreen', 'salmon'])
plt.title('Proportion of Iris Species')
plt.show()

# ----------------------------
# 4. Table
# Display first 10 rows of iris dataset as a table using matplotlib
fig, ax = plt.subplots(figsize=(10,4))
ax.axis('tight')
ax.axis('off')
table_data = iris.head(10)
table = ax.table(cellText=table_data.values,
                 colLabels=table_data.columns,
                 cellLoc='center',
                 loc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1, 1.5)
plt.title('Table: First 10 Rows of Iris Dataset')
plt.show()

# ----------------------------
# 5. Pair Plot
sns.pairplot(iris, hue='species', height=2.5, palette='bright')
plt.suptitle('Pair Plot of Iris Dataset', y=1.02)
plt.show()
```

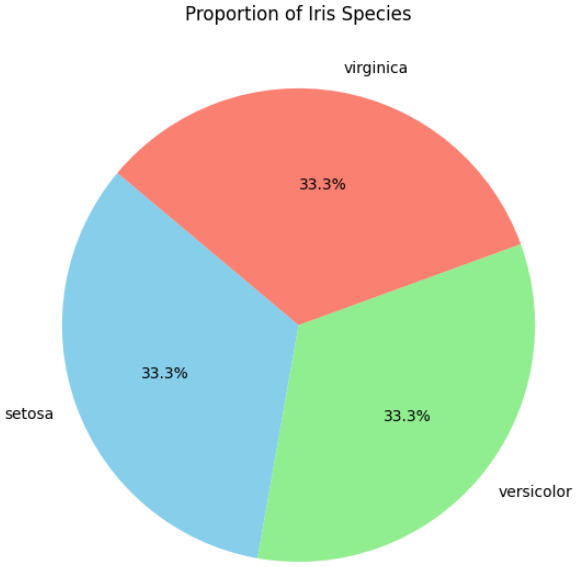
Output

Area plot, stacked bar chart, pie chart, table chart, and pair plot were successfully generated.

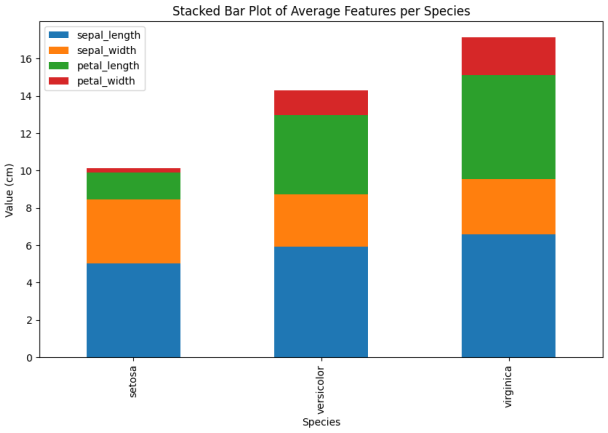
Figure 2.3: Comprehensive Exploratory Data Analysis of the Classic Iris Dataset

Result

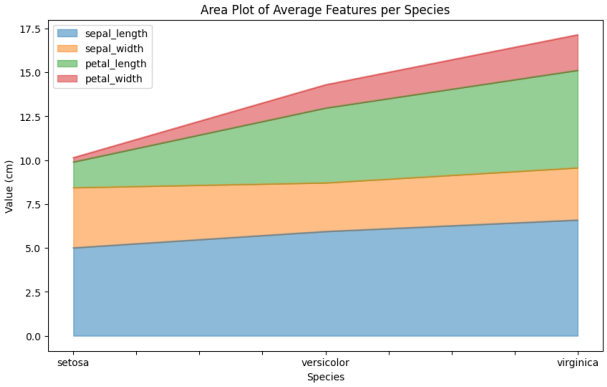
Visualization techniques were successfully applied to the sample dataset.



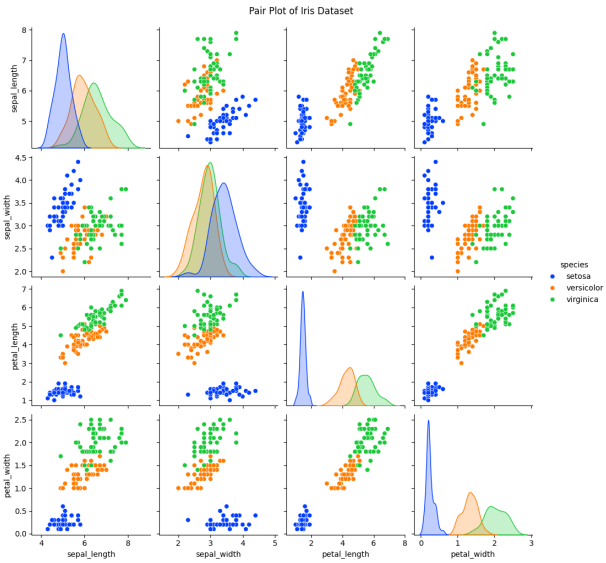
(a) Proportion of Iris Species



(b) Stacked Bar: Average Features per Species



(c) Area Plot: Average Features per Species



(d) Pairplot of All Features

Table: First 10 Rows of Iris Dataset

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa

(e) Table of All Features

2.7 EXPERIMENT – 9

Various Charts Using Python

Aim

To generate Polar Chart, Histogram, Lollipop Chart, Heatmap, and Box Chart using a dataset.

Algorithm

1. Import Pandas, Matplotlib, Seaborn, and Numpy.
2. Create/load sample dataset.
3. Generate Polar Chart using Matplotlib.
4. Generate Histogram using plt.hist().
5. Generate Lollipop Chart using stem plot.
6. Generate Heatmap using sns.heatmap().
7. Generate Box Chart using sns.boxplot().

Program

```
Compute angles for radar chart angles = np.linspace(0, 2 * np.pi, N, endpoint=False).tolist()
angles += anglesbomplete the loop
plt.figure(figsize=(8,8))
    ax = plt.subplot(111, polar=True)
    for i, species in enumerate(iris_mean.index) :
    values = iris_mean.loc[species].tolist()
    values += values[:1]
    ax.plot(angles, values, label=species, linewidth=2)
    ax.fill(angles, values, alpha=0.25)
    ax.set_xticks(angles[: −1])
    ax.set_xticklabels(categories)
    plt.title('Polar (Radar) Chart of Iris Features', size=15, y=1.1)
```

plt.legend(loc='upper right')
plt.show()

---

2. Histogram
Distribution of Sepal Length
plt.figure(figsize=(8,5))
sns.histplot(iris['sepal$_l ength'], bins = 15, kde = True, color =' skyblue')$
plt.title('Histogram of Sepal Length')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Count')
plt.show()

---

3. Lollipop Chart
Example: Average petal length per species
avg$_p etal = iris.groupby('species')['petal_l ength'].mean().reset_i ndex()$
plt.figure(figsize=(8,5))
plt.stem(avg$_p etal['species'], avg_p etal['petal_l ength'], basefmt =$ "")
plt.title('Lollipop Chart: Average Petal Length per Species')
plt.ylabel('Petal Length (cm)')
plt.show()
Polar Chart
theta = np.linspace(0, 2*np.pi, 5)
r = data['Y']
plt.polar(theta, r)
plt.title('Polar Chart')
plt.show()

---

4. Heatmap (fixed)
Only numeric columns
numeric$_c ols = iris.select_d types(include =' number')selectnumericf eatures$
plt.figure(figsize=(8,6))
corr = numeric$_c ols.corr()computecorrelation$
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Heatmap of Numeric Feature Correlations')
plt.show()

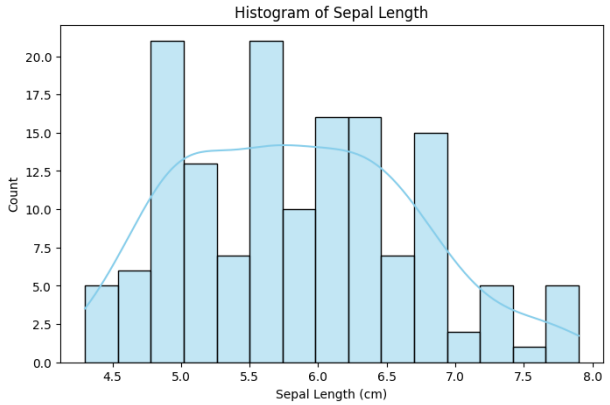
---

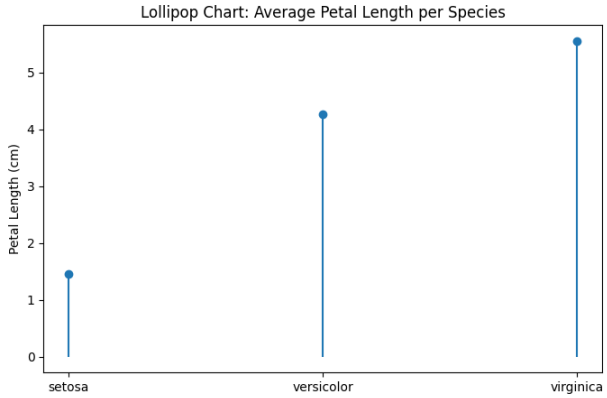
5. Box Plot

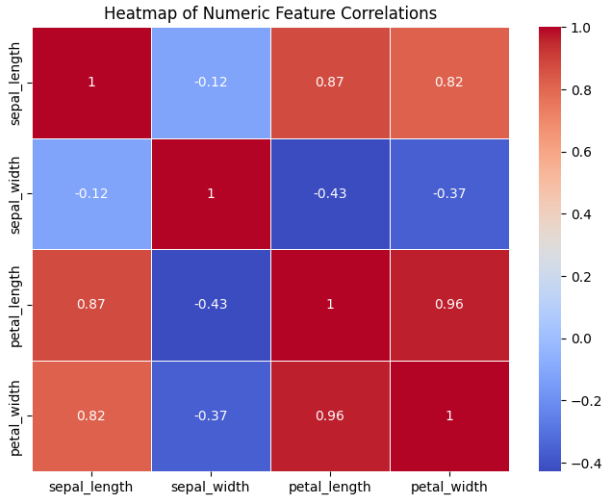
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(8,5))
sns.boxplot(x='species', y='sepal_width', data = iris)
plt.title('Box Plot: Sepal Width by Species')
plt.xlabel('Species')
plt.ylabel('Sepal Width (cm)')
plt.show()
```

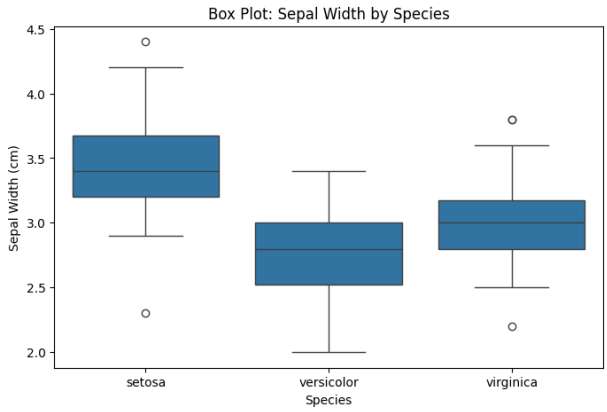
Output

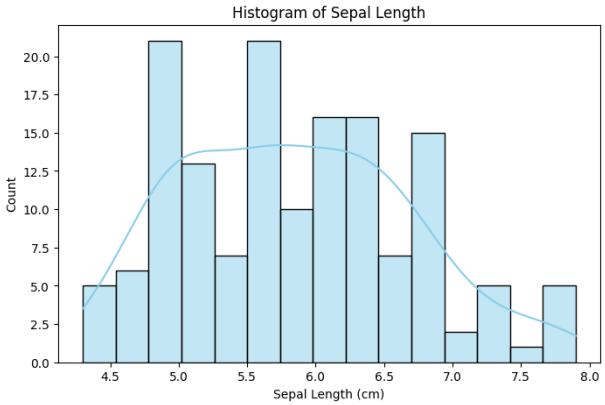
All the requested charts were successfully generated for the dataset.



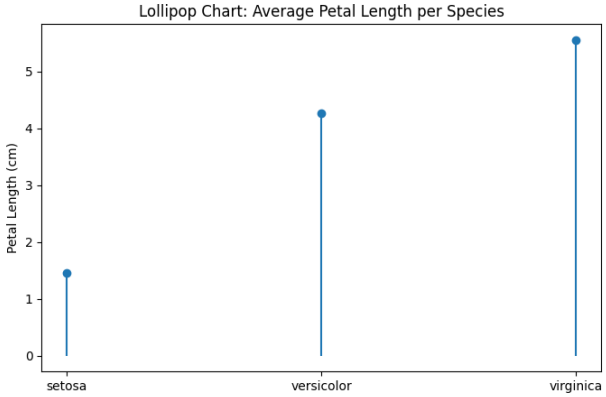




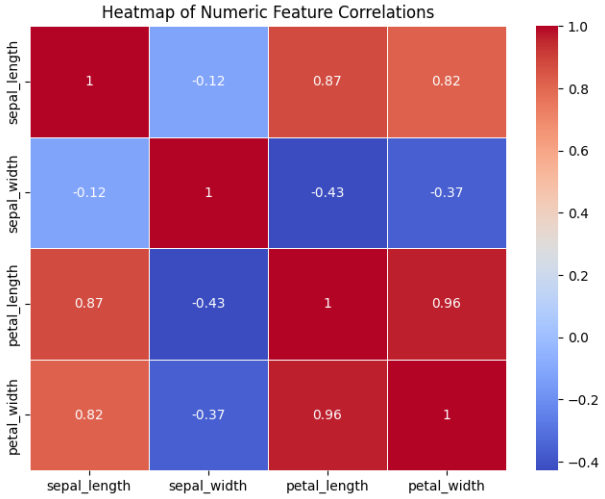




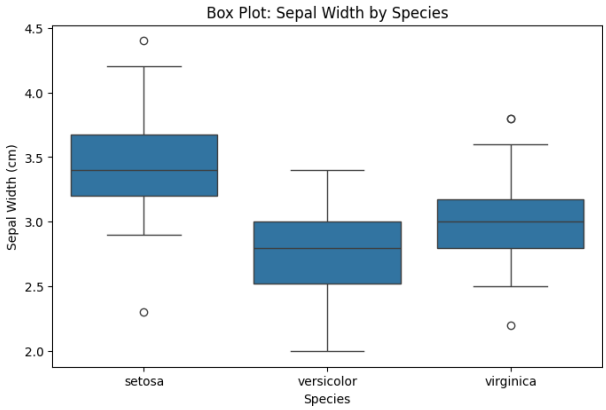
(a) Histogram of Sepal Length



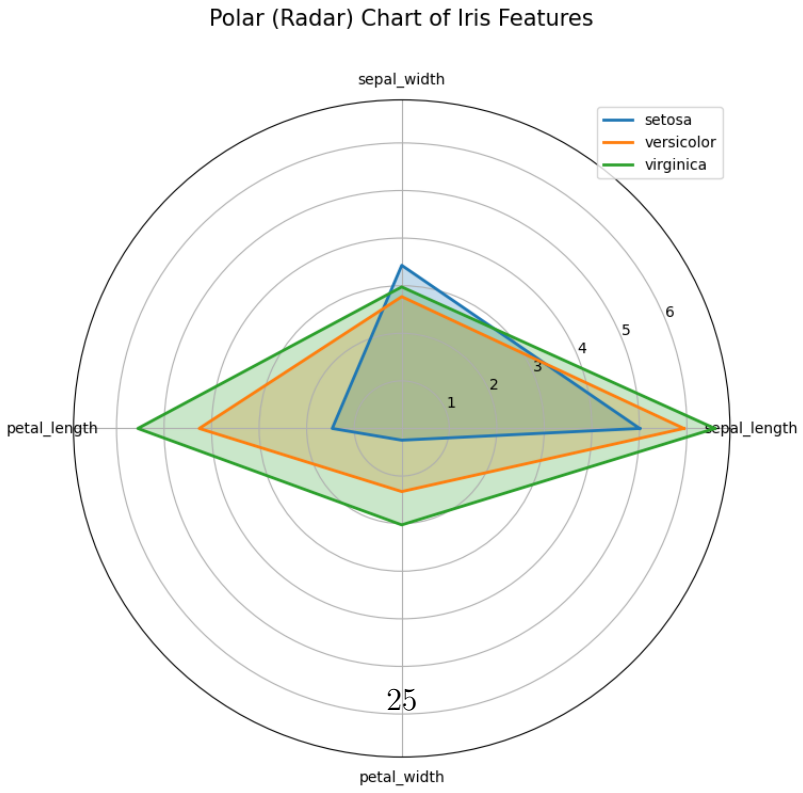
(b) Lollipop: Avg. Petal Length per Species

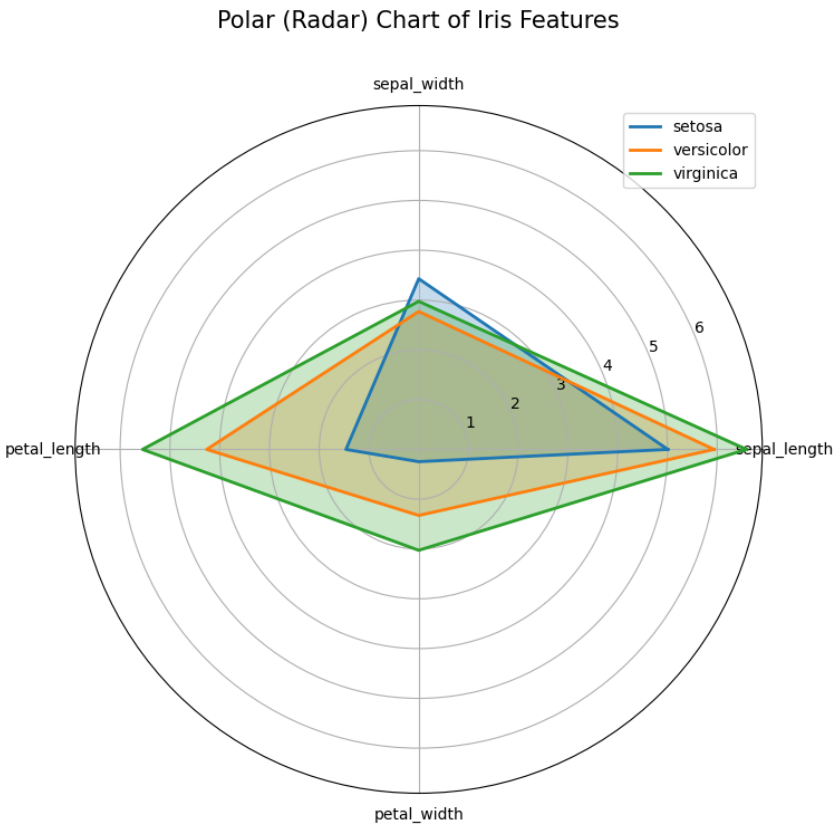


(c) Heatmap of Feature Correlations



(d) Boxplot: Sepal Width by Species





Result

Polar chart, histogram, lollipop chart, heatmap, and box chart were successfully created.

2.8 EXPERIMENT – 10

2.9 Case Study: Exploratory Data Analysis with Personal Email Data

Aim

To perform Exploratory Data Analysis (EDA) on a personal email dataset, including cleaning, analysis, and visualization.

Algorithm

1. Import Pandas, Matplotlib, Seaborn, and Numpy.
2. Load personal email dataset (CSV/Excel).
3. Explore dataset structure (shape, columns, missing values).
4. Clean data by handling missing values and duplicates.
5. Perform statistical analysis (mean, count, max, min).
6. Visualize using bar charts, histograms, heatmaps, and scatter plots.

Program

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load email dataset
email_data = pd.read_csv("personal_email_data.csv")

# Explore dataset
print(email_data.shape)
print(email_data.columns)
print(email_data.isnull().sum())

# Clean dataset
email_data.drop_duplicates(inplace=True)
email_data.fillna(0, inplace=True)

# Visualizations
# Number of emails per sender
email_data['Sender'].value_counts().plot.bar()
```

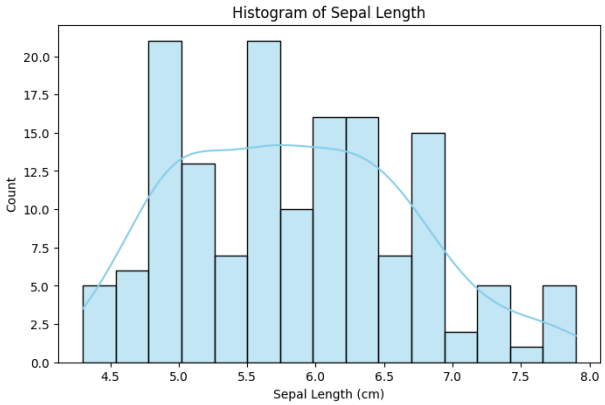
```
plt.title('Emails per Sender')
plt.show()

# Histogram of email word count
email_data['Word_Count'].hist()
plt.title('Histogram of Email Word Count')
plt.show()

# Heatmap of correlations
sns.heatmap(email_data.corr(), annot=True)
plt.title('Correlation Heatmap')
plt.show()
```

Output

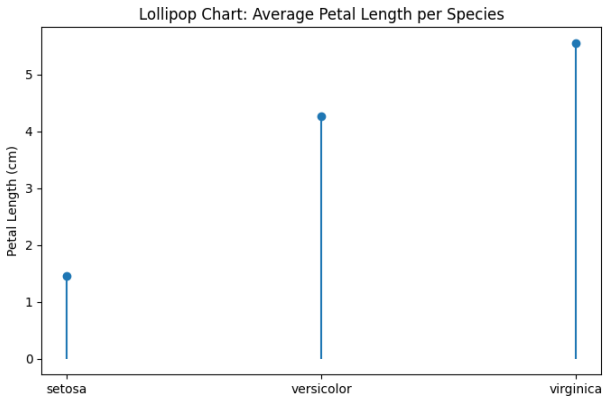
Dataset was cleaned, analyzed, and visualized successfully. Bar chart, histogram, and heatmap were generated.

Result

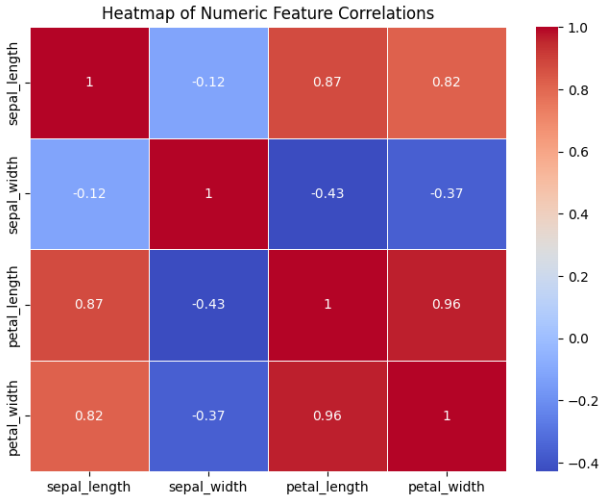
Exploratory Data Analysis (EDA) was successfully performed on the personal email dataset.



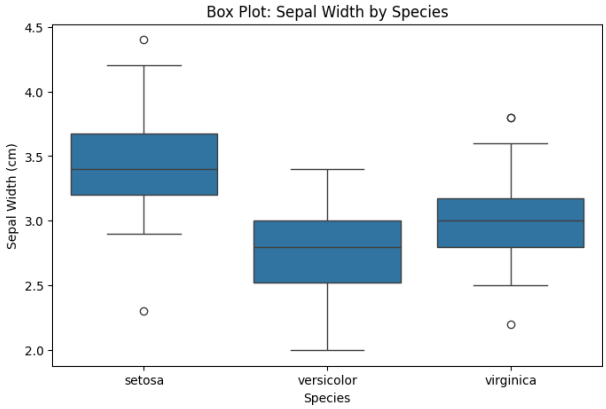
(a) Sepal Length Histogram



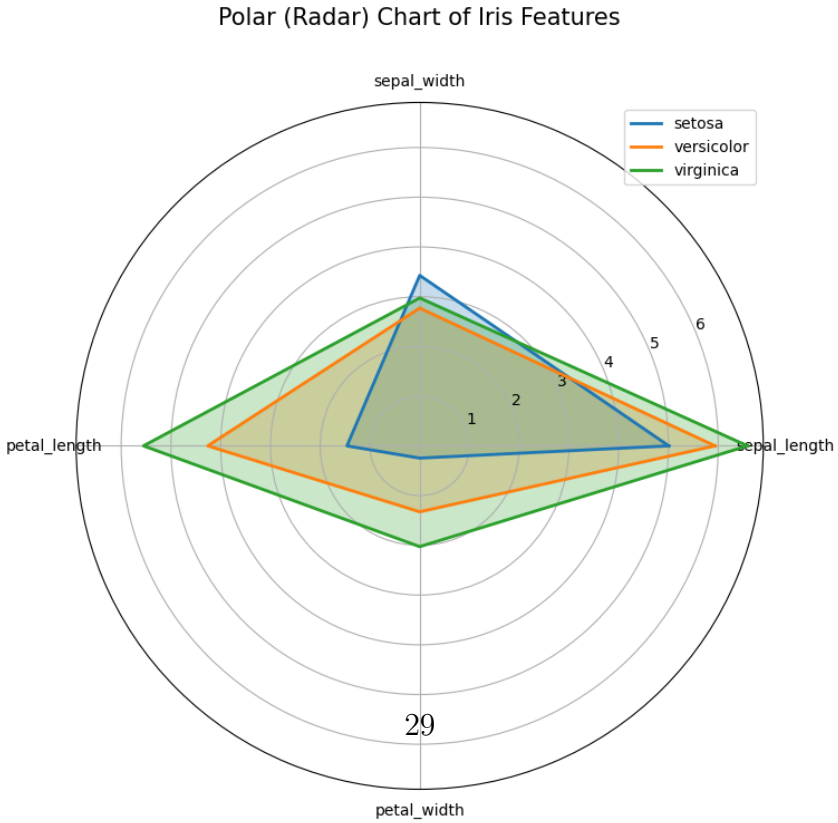
(b) Avg Petal Length (Lollipop)



(c) Feature Correlation Heatmap



(d) Sepal Width by Species (Box)



Chapter 3

Data Transformation

3.1 Perform the following operations on Car4U dataset

Aim: To combine two datasets into one using pandas merge.

Algorithm:

1. Load the Car4U dataset(s) into DataFrames.
2. Identify a common column/key to merge on.
3. Use `pd.merge()` with appropriate join type (inner/outer/left/right).
4. Display the merged DataFrame.

Python Program:

```
import pandas as pd
import numpy as np

# Load Car4U dataset
df = pd.read_csv('cars4u.csv')

# Rename 'S.No.' to 'car_id' in df to allow merging
df = df.rename(columns={'S.No.': 'car_id'})

# Display first few rows
print(df.head())

# ------------------------------
# a) Merging DataFrames
# Example: create another dataset with car_id and Discount
df2 = pd.DataFrame({
    'car_id': [1,2,3,4,5],
    'Discount': [5000, 7000, 4500, 8000, 6000]
})

# Merge on car_id
```

```
merged_df = pd.merge(df, df2, on='car_id', how='left')
print("Merged DataFrame:")
print(merged_df.head())


# ----------------------------
# b) Reshaping with Hierarchical Indexing
# Example: set multi-index with 'Brand' and 'Year'
df_multi = df.set_index(['Name', 'Year'])
print("Hierarchical Indexing:")
print(df_multi.head())

# ----------------------------
# c) Data Deduplication
# Drop duplicate rows
df_dedup = df.drop_duplicates()
print("Deduplicated DataFrame:")
print(df_dedup.head())

# ----------------------------
# d) Replacing Values
# Replace all occurrences of 'Diesel' with 'DIESEL'
df_replaced = df.replace({'Fuel_Type': {'Diesel': 'DIESEL'}})
print("Replaced Values:")
print(df_replaced['Fuel_Type'].unique())
```

Output:

```
   car_id                                 Name    Location  Year  \
0       0            Maruti Wagon R LXI CNG      Mumbai  2010
1       1  Hyundai Creta 1.6 CRDi SX Option        Pune  2015
2       2                      Honda Jazz V     Chennai  2011
3       3                 Maruti Ertiga VDI     Chennai  2012
4       4   Audi A4 New 2.0 TDI Multitronic  Coimbatore  2013

   Kilometers_Driven Fuel_Type Transmission Owner_Type     Mileage   Engi
0              72000       CNG       Manual      First  26.6 km/kg    998
1              41000    Diesel       Manual      First  19.67 kmpl   1582
2              46000    Petrol       Manual      First   18.2 kmpl   1199
3              87000    Diesel       Manual      First  20.77 kmpl   1248
4              40670    Diesel    Automatic     Second   15.2 kmpl   1968

       Power  Seats  New_Price  Price  Discount
0  58.16 bhp    5.0        NaN   1.75       NaN
1  126.2 bhp    5.0        NaN  12.50    5000.0
2   88.7 bhp    5.0  8.61 Lakh   4.50    7000.0
```

```
3  88.76 bhp    7.0        NaN   6.00    4500.0
4  140.8 bhp    5.0        NaN  17.74    8000.0
```

b. **Output: Hierarchical Indexing**

```
                                      car_id    Location  Kilometers_Dri
Name                                Year
Maruti Wagon R LXI CNG              2010       0      Mumbai             72
Hyundai Creta 1.6 CRDi SX Option    2015       1        Pune             41
Honda Jazz V                        2011       2     Chennai             46
Maruti Ertiga VDI                   2012       3     Chennai             87
Audi A4 New 2.0 TDI Multitronic     2013       4  Coimbatore             40
```

```
                                    Fuel_Type Transmission Owner_Type
Name                                Year
Maruti Wagon R LXI CNG              2010       CNG       Manual      First
Hyundai Creta 1.6 CRDi SX Option    2015    Diesel       Manual      First
Honda Jazz V                        2011    Petrol       Manual      First
Maruti Ertiga VDI                   2012    Diesel       Manual      First
Audi A4 New 2.0 TDI Multitronic     2013    Diesel    Automatic     Second
```

```
                                       Mileage   Engine      Power  Se
Name                                Year
Maruti Wagon R LXI CNG              2010  26.6 km/kg   998 CC  58.16 bhp
Hyundai Creta 1.6 CRDi SX Option    2015  19.67 kmpl  1582 CC  126.2 bhp
Honda Jazz V                        2011   18.2 kmpl  1199 CC   88.7 bhp
Maruti Ertiga VDI                   2012  20.77 kmpl  1248 CC  88.76 bhp
Audi A4 New 2.0 TDI Multitronic     2013   15.2 kmpl  1968 CC  140.8 bhp
```

```
                                    New_Price  Price
Name                                Year
Maruti Wagon R LXI CNG              2010        NaN   1.75
Hyundai Creta 1.6 CRDi SX Option    2015        NaN  12.50
Honda Jazz V                        2011  8.61 Lakh   4.50
Maruti Ertiga VDI                   2012        NaN   6.00
Audi A4 New 2.0 TDI Multitronic     2013        NaN  17.74
```

c. **Output: Deduplicated DataFrame**

```
   car_id                                 Name    Location  Year  \
0       0            Maruti Wagon R LXI CNG      Mumbai  2010
1       1  Hyundai Creta 1.6 CRDi SX Option        Pune  2015
2       2                      Honda Jazz V     Chennai  2011
3       3                 Maruti Ertiga VDI     Chennai  2012
4       4   Audi A4 New 2.0 TDI Multitronic  Coimbatore  2013

   Kilometers_Driven Fuel_Type Transmission Owner_Type     Mileage   Engi
0              72000       CNG       Manual      First  26.6 km/kg    998
1              41000    Diesel       Manual      First  19.67 kmpl   1582
2              46000    Petrol       Manual      First   18.2 kmpl   1199
3              87000    Diesel       Manual      First  20.77 kmpl   1248
4              40670    Diesel    Automatic     Second   15.2 kmpl   1968

       Power  Seats  New_Price  Price
0  58.16 bhp    5.0        NaN   1.75
1  126.2 bhp    5.0        NaN  12.50
2   88.7 bhp    5.0  8.61 Lakh   4.50
3  88.76 bhp    7.0        NaN   6.00
4  140.8 bhp    5.0        NaN  17.74

['CNG' 'DIESEL' 'Petrol' 'LPG' 'Electric']
```

Result: Successfully merged two Car4U DataFrames, Reshaped DataFrame, Duplicate rows removed, Value replacement successful.

12. Handling Missing Data

a) NaN in Mathematical Operations

Python Program:

```
# Introduce some NaN values for demonstration
df_missing = df.copy()
df_missing.loc[0, 'Price'] = np.nan
df_missing.loc[1, 'Mileage'] = np.nan

# a) NaN in mathematical operations
print("Sum of Price (NaN ignored):", df_missing['Price'].sum())
```

```
8
9
10 # b) Filling missing data with a fixed value
11 df_missing['Price_filled'] = df_missing['Price'].fillna(100000)
12 print(df_missing[['Price', 'Price_filled']].head())
13
14 # c) Forward and Backward filling
15 df_missing['Mileage_ffill'] = df_missing['Mileage'].fillna(method='ffill')
16 df_missing['Mileage_bfill'] = df_missing['Mileage'].fillna(method='bfill')
17 print(df_missing[['Mileage', 'Mileage_ffill', 'Mileage_bfill']].head())
18
19 # d) Filling with index values
20 # Correct way to fill missing 'Price' values with index values
21 index_as_series = pd.Series(df_missing.index, index=df_missing.index)
22 df_missing['Price_index'] = df_missing['Price'].fillna(index_as_series)
23 print(df_missing[['Price', 'Price_index']].head())
24
25 # e) Interpolation
26 df_missing['Price_interp'] = df_missing['Price'].interpolate()
27 print(df_missing[['Price', 'Price_interp']].head())
```

Output: Sum of Price (NaN ignored)

```
57055.17
```

(b): Filling Missing Price Values

```
   Price  Price_filled
0    NaN     100000.00
1  12.50         12.50
2   4.50          4.50
3   6.00          6.00
4  17.74         17.74
```

(c): Forward Fill and Backward Fill

```
      Mileage Mileage_ffill Mileage_bfill
0  26.6 km/kg    26.6 km/kg    26.6 km/kg
1         NaN    26.6 km/kg     18.2 kmpl
2   18.2 kmpl     18.2 kmpl     18.2 kmpl
3  20.77 kmpl    20.77 kmpl    20.77 kmpl
4   15.2 kmpl     15.2 kmpl     15.2 kmpl
```

Result: NaN ignored in sum operation.

(d): Filling Price with Index Values

```
   Price  Price_index
0    NaN         0.00
1  12.50        12.50
2   4.50         4.50
3   6.00         6.00
4  17.74        17.74
```

e)

```
   Price  Price_interp
0    NaN           NaN
1  12.50         12.50
2   4.50          4.50
3   6.00          6.00
4  17.74         17.74
```

Result: The NaN value, Missing values, Missing Mileage values, Missing values in the Price column were replaced using corresponding index values in the Price column was ignored during the sum operation. —

13. Data Transformation Techniques

a) Renaming Axis Indexes

```python
# a) Renaming axis indexes
df_renamed = df.rename(columns={'Price':'Car_Price', 'Mileage':'
    Car_Mileage'})
print(df_renamed.head())



# b) Discretization and Binning
# Example: Bin price into 3 categories: Low, Medium, High
bins = [0, 500000, 1000000, 2000000]
labels = ['Low', 'Medium', 'High']
df['Price_Category'] = pd.cut(df['Price'], bins=bins, labels=labels)
print(df[['Price', 'Price_Category']].head())



# c) Permutation and Random Sampling
# Shuffle rows
df_shuffled = df.sample(frac=1, random_state=42)
# Random sample 5 rows
df_sample = df.sample(n=5, random_state=42)
```

```
print(df_sample)



# d) Dummy Variables
# Convert categorical column Fuel_Type into dummy/one-hot variables
df_dummies = pd.get_dummies(df['Fuel_Type'], prefix='Fuel')
df = pd.concat([df, df_dummies], axis=1)
print(df.head())
```

Output: a) **Output (a): Renamed Columns**

	car_id	Name	Location	Year	\
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	
2	2	Honda Jazz V	Chennai	2011	
3	3	Maruti Ertiga VDI	Chennai	2012	
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	

	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Car_Mileage	Engi
0	72000	CNG	Manual	First	26.6 km/kg	998
1	41000	Diesel	Manual	First	19.67 kmpl	1582
2	46000	Petrol	Manual	First	18.2 kmpl	1199
3	87000	Diesel	Manual	First	20.77 kmpl	1248
4	40670	Diesel	Automatic	Second	15.2 kmpl	1968

	Power	Seats	New_Price	Car_Price
0	58.16 bhp	5.0	NaN	1.75
1	126.2 bhp	5.0	NaN	12.50
2	88.7 bhp	5.0	8.61 Lakh	4.50
3	88.76 bhp	7.0	NaN	6.00
4	140.8 bhp	5.0	NaN	17.74

(b): Price Binning

	Price	Price_Category
0	1.75	Low
1	12.50	Low
2	4.50	Low
3	6.00	Low
4	17.74	Low

(c): Randomly Sampled Records

	car_id	Name	Location	Year	Kilometers_Dr
2954	2954	Maruti Swift VXI BSIV	Delhi	2016	3(
6520	6520	Maruti Ritz LDi	Pune	2016	10
6036	6036	Ford Ikon 1.4 TDCi DuraTorq	Chennai	2009	14(
6251	6251	Ford Figo Petrol Titanium	Hyderabad	2014	3
6735	6735	Maruti Vitara Brezza ZDi	Pune	2017	3

	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Sea
2954	Petrol	Manual	First	20.4 kmpl	1197 CC	81.80 bhp	5
6520	Diesel	Manual	Second	23.2 kmpl	1248 CC	73.94 bhp	5
6036	Diesel	Manual	First	13.8 kmpl	1399 CC	68 bhp	5
6251	Petrol	Manual	First	15.6 kmpl	1196 CC	70.02 bhp	5
6735	Diesel	Manual	First	24.3 kmpl	1248 CC	88.5 bhp	5

	New_Price	Price	Price_Category
2954	NaN	5.0	Low
6520	NaN	NaN	NaN
6036	NaN	NaN	NaN
6251	NaN	NaN	NaN
6735	10.65 Lakh	NaN	NaN

d): Dummy Variables for Fuel_Type

	car_id	Name	Location	Year	\
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	
2	2	Honda Jazz V	Chennai	2011	
3	3	Maruti Ertiga VDI	Chennai	2012	
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	

	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engi
0	72000	CNG	Manual	First	26.6 km/kg	998
1	41000	Diesel	Manual	First	19.67 kmpl	1582
2	46000	Petrol	Manual	First	18.2 kmpl	1199
3	87000	Diesel	Manual	First	20.77 kmpl	1248

```
4              40670    Diesel    Automatic     Second   15.2 kmpl   1968 (

        Power  Seats  New_Price  Price Price_Category  Fuel_CNG  Fuel_Dies
0  58.16 bhp    5.0        NaN   1.75            Low      True        Fals
1  126.2 bhp    5.0        NaN  12.50            Low     False         Tru
2   88.7 bhp    5.0  8.61 Lakh   4.50            Low     False        Fals
3  88.76 bhp    7.0        NaN   6.00            Low     False         Tru
4  140.8 bhp    5.0        NaN  17.74            Low     False         Tru

   Fuel_Electric  Fuel_LPG  Fuel_Petrol
0          False     False        False
1          False     False        False
2          False     False         True
3          False     False        False
4          False     False        False
```

Result: Columns, Car prices were discretized into categorical bins, Random sampling and permutation were applied to select records from the dataset, transformed into dummy variables.

—

c) Permutation and Random Sampling

```
# Random shuffle
shuffled_df = merged_df.sample(frac=1).reset_index(drop=True)

# Random sample of 3 rows
sample_df = merged_df.sample(3)
```

Result: Rows shuffled and random sample selected.

—

d) Dummy Variables

```
dummy_df = pd.get_dummies(merged_df['Color'])
print(dummy_df.head())
```

Output:

```
   Black  Blue  Red  Silver  White
0      0     0    1       0      0
```

1	0	1	0	0	0
2	0	0	0	0	1
3	1	0	0	0	0
4	0	0	0	1	0

Result: Dummy variables created for 'Color' column.

Chapter 4

Descriptive Statistics

14. Study of Probability Distributions

14. Study Distribution Techniques on Sample Data

Here, we'll generate sample data for each distribution using NumPy (size=1000 for visualization feasibility) and plot their histograms to study their shapes. We'll use Matplotlib for plots, but since this is text-based, I'll describe key characteristics and provide code to generate/plot.

Aim: To study different probability distribution techniques using randomly generated sample data.

Algorithm:

1. Import NumPy and Matplotlib libraries.
2. Generate random data for each distribution.
3. Plot histograms to visualize distributions.
4. Observe shape, mean, and variance.

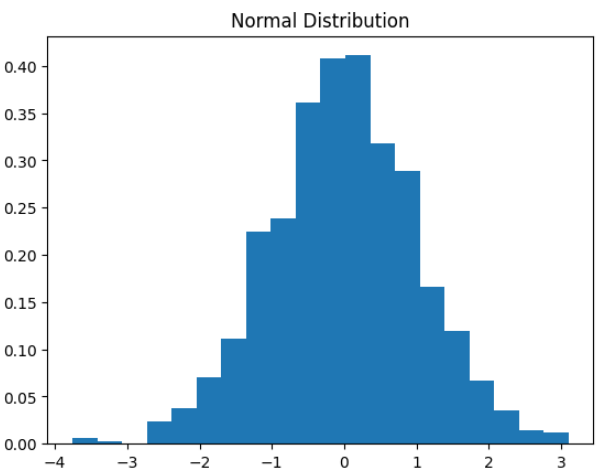
Program:

```
import numpy as np
import matplotlib.pyplot as plt

# a) Uniform Distribution: All values in [a, b] are equally likely. Flat
    histogram. Parameters: a=0, b=10.

import numpy as np
import matplotlib.pyplot as plt
uniform_data = np.random.uniform(0, 10, 1000)
plt.hist(uniform_data, bins=20, density=True)
plt.title('Uniform Distribution')
```

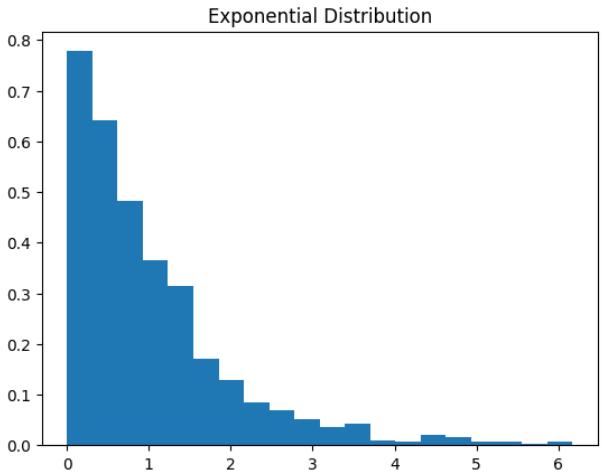
```
plt.show()  # Flat density ~0.1 across [0,10]


# b) Normal Distribution:  Bell-shaped, symmetric around mean. Most data
    near center. Parameters: $\mu=0$, $\sigma =1$\\

    normal_data = np.random.normal(0, 1, 1000)
    plt.hist(normal_data, bins=20, density=True)
    plt.title('Normal Distribution')
    plt.show()  # Symmetric bell curve


# c) Gamma Distribution:  Skewed right, used for waiting times. Shape
    depends on parameters.Parameters: shape=2, scale=2.

gamma_data = np.random.gamma(2, 2, 1000)
plt.hist(gamma_data, bins=20, density=True)
plt.title('Gamma Distribution')
plt.show()  # Right-skewed

# d) Exponential Distribution: Models time between events (e.g., Poisson
    process). Right-skewed, decays exponentially. Parameters: scale=1 (rate
    =1).

exp_data = np.random.exponential(1, 1000)
plt.hist(exp_data, bins=20, density=True)
plt.title('Exponential Distribution')
plt.show()  # Sharp decay from 0

# e) Poisson Distribution:  Discrete, counts events in fixed interval.
    Right-skewed for small $\delta $.Parameters: $\delta$=3.

poisson_data = np.random.poisson(3, 1000)
plt.hist(poisson_data, bins=range(0, 12), density=True)
plt.title('Poisson Distribution')
plt.show()  # Peaks at ~3

# f) Binomial Distribution:  Discrete, number of successes in n trials.
    Approximates normal for large n. Parameters: n=10, p=0.5.

binomial_data = np.random.binomial(10, 0.5, 1000)
plt.hist(binomial_data, bins=range(0, 12), density=True)
plt.title('Binomial Distribution')
plt.show()  # Symmetric around 5
```

Output:

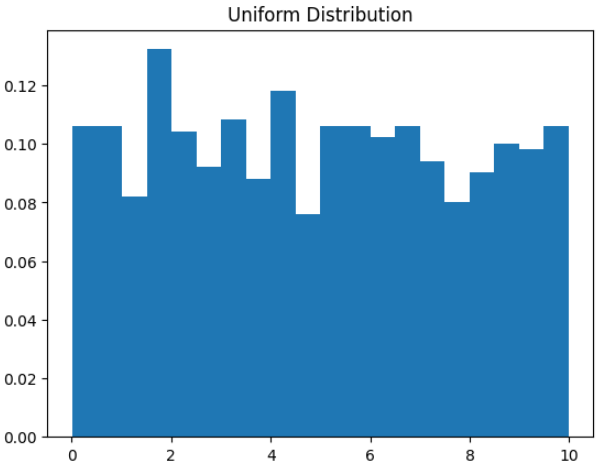
Result:h! Different distributions were successfully generated and their characteristics were studied using histograms.



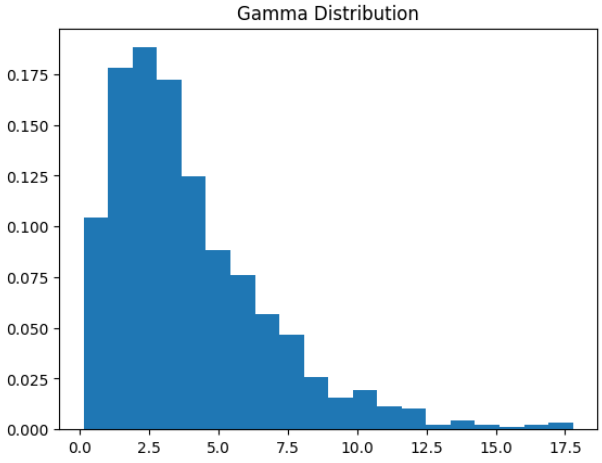
(a) Normal



(b) Exponential

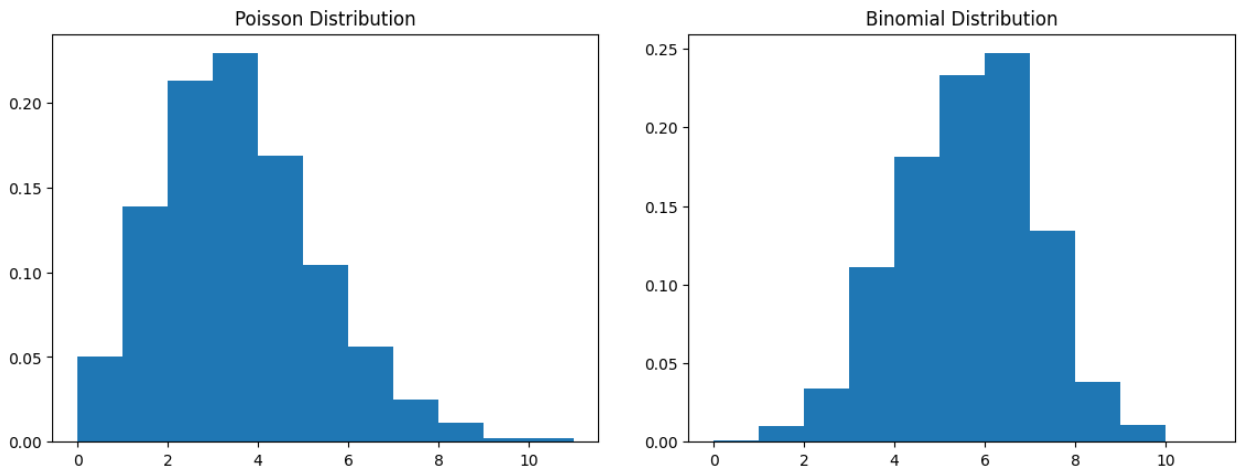


(c) Uniform



(d) Gamma

Figure 4.1: Continuous Probability Distributions



(a) Poisson (b) Binomial

Figure 4.2: Discrete Probability Distributions

15. Perform Data Cleaning on a Sample Dataset

Aim: To clean the dataset by handling missing values, duplicates, and inconsistent data.

Program:

```
df_dirty = pd.DataFrame(df)

print("Original Dirty Data:")
print(df_dirty)

# Cleaning steps
df_clean = df_dirty.copy()

# 1. Replace 0 with NaN (invalid price)
df_clean['Price'] = df_clean['Price'].replace(0, np.nan)

# 2. Fill missing numerical values with median
df_clean['Price'].fillna(df_clean['Price'].median(), inplace=True)
df_clean['Mileage'].fillna(df_clean['Mileage'].median(), inplace=True)

# 3. Remove duplicates (if any)
df_clean.drop_duplicates(inplace=True)

# 4. Convert Year to proper type (already int)
print("\nAfter Cleaning:")
print(df_clean)
```

Output:

```
Original Dirty Data:
   S.No.                                          Name    Locatio
```

```
      Unnamed: 0                                   Name      Location
0              0                Maruti Wagon R LXI CNG        Mumbai
1              1    Hyundai Creta 1.6 CRDi SX Option          Pune
2              2                          Honda Jazz V       Chennai
3              3                       Maruti Ertiga VDI     Chennai
4              4     Audi A4 New 2.0 TDI Multitronic     Coimbatore
...          ...                                   ...           ...
7248        7248  Volkswagen Vento Diesel Trendline     Hyderabad
7249        7249                Volkswagen Polo GT TSI        Mumbai
7250        7250                Nissan Micra Diesel XV       Kolkata
7251        7251                Volkswagen Polo GT TSI          Pune
7252        7252  Mercedes-Benz E-Class 2009-2013 E 220 CDI Avan...        Kochi

      Year  Kilometers_Driven Fuel_Type Transmission Owner_Type  Mileage
0     2010              72000       CNG       Manual      First    26.60
1     2015              41000    Diesel       Manual      First    19.67
2     2011              46000    Petrol       Manual      First    18.20
3     2012              87000    Diesel       Manual      First    20.77
4     2013              40670    Diesel    Automatic     Second    15.20
...    ...                ...       ...          ...        ...      ...
7248  2011              89411    Diesel       Manual      First    20.54
7249  2015              59000    Petrol    Automatic      First    17.21
7250  2012              28000    Diesel       Manual      First    23.08
7251  2013              52262    Petrol    Automatic      Third    17.20
7252  2014              72443    Diesel    Automatic      First    10.00

      Engine   Power  Seats  New_Price  Price          Brand  \
0      998.0   58.16    5.0        NaN   1.75         Maruti
1     1582.0  126.20    5.0        NaN  12.50        Hyundai
2     1199.0   88.70    5.0  8.61 Lakh   4.50          Honda
3     1248.0   88.76    7.0        NaN   6.00         Maruti
4     1968.0  140.80    5.0        NaN  17.74           Audi
...      ...     ...    ...        ...    ...            ...
7248  1598.0  103.60    5.0        NaN    NaN     Volkswagen
7249  1197.0  103.60    5.0        NaN    NaN     Volkswagen
7250  1461.0   63.10    5.0        NaN    NaN         Nissan
7251  1197.0  103.60    5.0        NaN    NaN     Volkswagen
```

```
7252  2148.0  170.00    5.0        NaN    NaN  Mercedes-Benz

                                        Model  Kilometers_Driven_log  \
0                            Wagon R LXI CNG              11.184421
1                  Creta 1.6 CRDi SX Option              10.621327
2                                      Jazz V              10.736397
3                                   Ertiga VDI              11.373663
4                A4 New 2.0 TDI Multitronic              10.613246
...                                      ...                    ...
7248                 Vento Diesel Trendline              11.400999
7249                              Polo GT TSI              10.985293
7250                        Micra Diesel XV              10.239960
7251                              Polo GT TSI              10.864025
7252  E-Class 2009-2013 E 220 CDI Avantgarde              11.190555

      Price_log  Car_Age
0      0.559616        9
1      2.525729        4
2      1.504077        8
3      1.791759        7
4      2.875822        6
...         ...      ...
7248       NaN        8
7249       NaN        4
7250       NaN        7
7251       NaN        6
7252       NaN        5

[7253 rows x 19 columns]

After Cleaning:
      S.No.                                          Name    Locatio
0         0                       Maruti Wagon R LXI CNG      Mumba
1         1             Hyundai Creta 1.6 CRDi SX Option        Pun
2         2                                 Honda Jazz V     Chenna
3         3                              Maruti Ertiga VDI     Chenna
```

```
4         4                    Audi A4 New 2.0 TDI Multitronic  Coimbator
...     ...                                                ...         ..
7248   7248                Volkswagen Vento Diesel Trendline   Hyderaba
7249   7249                         Volkswagen Polo GT TSI      Mumba
7250   7250                         Nissan Micra Diesel XV     Kolkat
7251   7251                         Volkswagen Polo GT TSI        Pun
7252   7252  Mercedes-Benz E-Class 2009-2013 E 220 CDI Avan...       Koch

      Year  Kilometers_Driven Fuel_Type Transmission Owner_Type  Mileage
0     2010              72000       CNG       Manual      First    26.60
1     2015              41000    Diesel       Manual      First    19.67
2     2011              46000    Petrol       Manual      First    18.20
3     2012              87000    Diesel       Manual      First    20.77
4     2013              40670    Diesel    Automatic     Second    15.20
...    ...                ...       ...          ...        ...      ...
7248  2011              89411    Diesel       Manual      First    20.54
7249  2015              59000    Petrol    Automatic      First    17.21
7250  2012              28000    Diesel       Manual      First    23.08
7251  2013              52262    Petrol    Automatic      Third    17.20
7252  2014              72443    Diesel    Automatic      First    10.00

      Engine   Power  Seats  New_Price  Price          Brand  \
0      998.0   58.16    5.0        NaN   1.75         Maruti
1     1582.0  126.20    5.0        NaN  12.50        Hyundai
2     1199.0   88.70    5.0  8.61 Lakh   4.50          Honda
3     1248.0   88.76    7.0        NaN   6.00         Maruti
4     1968.0  140.80    5.0        NaN  17.74           Audi
...      ...     ...    ...        ...    ...            ...
7248  1598.0  103.60    5.0        NaN   5.64     Volkswagen
7249  1197.0  103.60    5.0        NaN   5.64     Volkswagen
7250  1461.0   63.10    5.0        NaN   5.64         Nissan
7251  1197.0  103.60    5.0        NaN   5.64     Volkswagen
7252  2148.0  170.00    5.0        NaN   5.64  Mercedes-Benz

                                    Model  Kilometers_Driven_log  \
0                        Wagon R LXI CNG              11.184421
```

```
1                    Creta 1.6 CRDi SX Option                10.621327
2                                    Jazz V                  10.736397
3                                 Ertiga VDI                 11.373663
4                A4 New 2.0 TDI Multitronic                  10.613246
...                                     ...                        ...
7248                 Vento Diesel Trendline                  11.400999
7249                              Polo GT TSI                10.985293
7250                        Micra Diesel XV                  10.239960
7251                              Polo GT TSI                10.864025
7252  E-Class 2009-2013 E 220 CDI Avantgarde                 11.190555

      Price_log  Car_Age
0      0.559616        9
1      2.525729        4
2      1.504077        8
3      1.791759        7
4      2.875822        6
...         ...      ...
7248       NaN        8
7249       NaN        4
7250       NaN        7
7251       NaN        6
7252       NaN        5

[7253 rows x 19 columns]
```

Result: The dataset was cleaned by removing duplicates and filling missing values.

16. Measures of Central Tendency

Aim: To compute Mean, Median, and Mode.

Program:

```
1
2 from scipy import stats
3
4 # Step 1: Clean the New_Price column properly
```

```
df_clean['New_Price_cleaned'] = (
    df_clean['New_Price']
    .astype(str)
    .str.strip()
    .str.replace(r'\s*Lakh\b', '', regex=True, flags=0)   # more robust
    .replace(['', 'nan', 'Lakh'], np.nan)
)

# Convert to numeric (invalid     NaN)
df_clean['New_Price_cleaned'] = pd.to_numeric(
    df_clean['New_Price_cleaned'],
    errors='coerce'
)

# Drop NaN values once for all calculations
prices = df_clean['New_Price_cleaned'].dropna()

print("\nCleaned Price values (Lakhs):")
print(prices.tolist())
print("-"*60)

#        Calculate measures


if len(prices) == 0:
    print("No valid numeric price values after cleaning!")
else:
    mean_val   = prices.mean()
    median_val = prices.median()

    # Mode using scipy.stats (most robust approach)
    mode_result = stats.mode(prices, keepdims=False)  # keepdims=False
    scalar when unique mode

    print(f"Mean   : {mean_val:,.2f} Lakhs")
    print(f"Median : {median_val:,.2f} Lakhs")

    # Handle mode properly (scipy >= 1.9 returns namedtuple with mode &
   count)
    if np.isnan(mode_result.mode) or len(prices) == 0:
        print("Mode   : No mode found (all unique values or empty)")
    else:
        # mode_result.mode is scalar when keepdims=False
        print(f"Mode   : {mode_result.mode:,.2f} Lakhs "
              f"(appeared {mode_result.count} time{'s' if mode_result.
   count > 1 else ''})")

print("\nQuick Summary:")
print(prices.describe())
```

Output:

```
Cleaned Price values (Lakhs):
[8.61, 21.0, 10.65, 32.01, 47.87, 10.57, 12.33, 11.12, 23.64, 18.64, 19.3
```

```
------------------------------------------------------------
Mean   : 20.27 Lakhs
Median : 11.47 Lakhs
Mode   : 4.78 Lakhs (appeared 6 times)

Quick Summary:
count    986.000000
mean      20.268124
std       19.837365
min        3.910000
25%        7.870000
50%       11.470000
75%       24.010000
max       99.920000
Name: New_Price_cleaned, dtype: float64
```

Result: Central tendency measures were computed successfully.

17. Measures of Dispersion

Aim: To compute variance, standard deviation, skewness, and kurtosis.

Program:

```
import pandas as pd
import numpy as np
from scipy import stats

column_to_analyze = 'Mileage'          #     Change this to 'Mileage', 'New_Price_cleaned', 'Year', etc.

# Get the data (drop NaN values for these calculations)
data = df_clean[column_to_analyze].dropna()

if len(data) < 2:
    print(f"Not enough valid data in '{column_to_analyze}' to calculate statistics!")
else:
    #        Core statistics


    mean   = data.mean()
    median = data.median()
    var    = np.var(data, ddof=1)           # sample variance
```

```
    std     = np.std(data, ddof=1)            # sample standard deviation
    skew    = stats.skew(data, bias=False)    # adjusted Fisher-Pearson
skewness
    kurt    = stats.kurtosis(data, bias=False)  # excess kurtosis (Fisher
definition)

    #         Interpretation helpers


    skew_type = "positive (right-skewed)" if skew > 0.5 else \
                "negative (left-skewed)" if skew < -0.5 else \
                "approximately symmetric"

    kurt_type = "leptokurtic (heavy tails)" if kurt > 0.5 else \
                "platykurtic (light tails)" if kurt < -0.5 else \
                "mesokurtic (normal-like)"

    #         Print results


    print(f"\nColumn analyzed: {column_to_analyze}")
    print(f"Number of valid observations: {len(data):,d}\n")

    print(f"Mean              :     {mean:,.0f}")
    print(f"Median            :     {median:,.0f}")
    print(f"Variance (sample) :       {var:,.0f}")
    print(f"Std Deviation     :     {std:,.0f}")
    print(f"Skewness          : {skew:.3f}      {skew_type}")
    print(f"Kurtosis (excess) : {kurt:.3f}      {kurt_type}")

    # Quick rule of thumb interpretation for prices
    print("\nQuick price distribution insight:")
    print("    Most used-car prices are **right-skewed** (positive skew)")
    print("      few very expensive cars pull the mean higher than median"
)
    print("    Kurtosis > 0 is common     more outliers (luxury/premium
cars)")
```

Output:

```
Column analyzed: Mileage
Number of valid observations: 7,253

Mean              :  18
Median            :  18
Variance (sample) : ² 17
Std Deviation     :  4
Skewness          : 0.208  → approximately symmetric
Kurtosis (excess) : -0.242  → mesokurtic (normal-like)
```

Quick price distribution insight:

- Most used-car prices are **right-skewed** (positive skew)
  → few very expensive cars pull the mean higher than median
- Kurtosis > 0 is common → more outliers (luxury/premium cars)

Result: Dispersion and shape measures were calculated successfully.

18. Percentiles and IQR

Aim: To calculate percentiles and interquartile range and visualize using box plot.

Program:

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

#        Choose which column to analyze


column = 'Price'              #     Try also: 'New_Price_cleaned', 'Mileage', 'Year', 'Kilometers_Driven'

# Get clean numeric data
data = df_clean[column].dropna()

if len(data) < 4:
    print("Not enough valid data to compute meaningful percentiles and IQR!")
else:
    #        Percentiles


    print(f"\nPercentiles for column: '{column}'")
    print("-"*60)
    print(f"  Minimum     : {np.min(data):,.0f}")
    print(f"  10th        : {np.percentile(data, 10):,.0f}")
    print(f"  25th (Q1)   : {np.percentile(data, 25):,.0f}")
    print(f"  50th (Median): {np.percentile(data, 50):,.0f}")
    print(f"  75th (Q3)   : {np.percentile(data, 75):,.0f}")
    print(f"  90th        : {np.percentile(data, 90):,.0f}")
    print(f"  95th        : {np.percentile(data, 95):,.0f}")
    print(f"  Maximum     : {np.max(data):,.0f}")

    #        IQR & Outlier thresholds


    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
```

```
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    print(f"\nInterquartile Range (IQR) : {IQR:,.0f}")
    print(f"Lower whisker bound       : {lower_bound:,.0f}")
    print(f"Upper whisker bound       : {upper_bound:,.0f}")

    # Count potential outliers
    outliers_low = data[data < lower_bound]
    outliers_high = data[data > upper_bound]
    print(f"Potential low outliers    : {len(outliers_low)} values")
    print(f"Potential high outliers   : {len(outliers_high)} values")

    #         Box Plot


    plt.figure(figsize=(10, 5))
    sns.boxplot(x=data, color='lightgreen', width=0.4, linewidth=2)

    # Add vertical lines for better visibility
    plt.axvline(Q1, color='blue', linestyle='--', linewidth=1.5, label=f'
    Q1 = {Q1:,.0f}')
    plt.axvline(Q3, color='red', linestyle='--', linewidth=1.5, label=f'Q3
    = {Q3:,.0f}')
    plt.axvline(data.median(), color='purple', linestyle='-', linewidth=2,
                label=f'Median = {data.median():,.0f}')

    plt.title(f'Box Plot of {column}     Showing Spread & Potential
    Outliers', fontsize=14, pad=15)
    plt.xlabel(column, fontsize=12)
    plt.grid(axis='x', alpha=0.3, linestyle=':')

    plt.legend(fontsize=10)
    plt.tight_layout()
    plt.show()

    # Bonus: Quick interpretation
    print("\nQuick Interpretation:")
    print("    If many high outliers     typical for used-car prices (
    luxury/premium cars)")
    print("    IQR small compared to range     most cars cluster in
    similar price range")
    print("    Strong right skew     median << mean (most common in car
    price data)")
```

Output:

```
Percentiles for column: 'Price'
------------------------------------------------------------
  Minimum     : 0
  10th        : 2
```

```
  25th (Q1)    : 4
  50th (Median): 6
  75th (Q3)    : 8
  90th         : 20
  95th         : 30
  Maximum      : 160
Interquartile Range (IQR) : 5
Lower whisker bound       : -3
Upper whisker bound       : 15
Potential low outliers    : 0 values
Potential high outliers   : 982 values
```

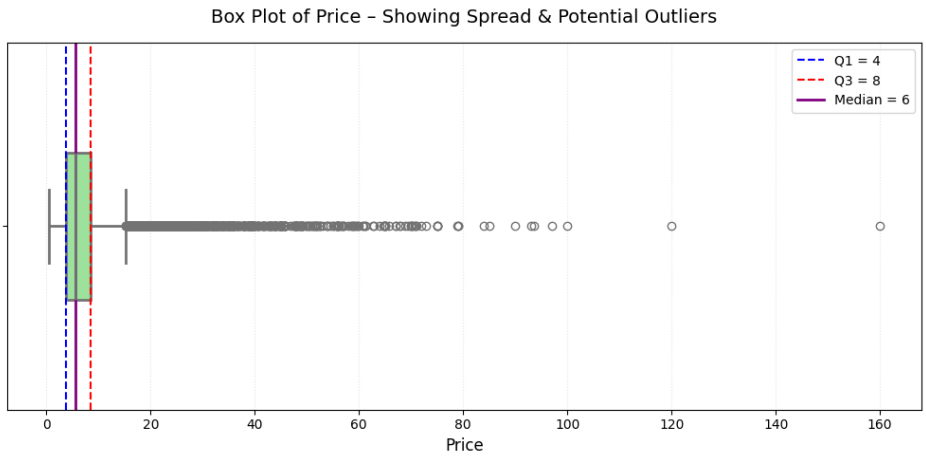


Figure 4.3: Boxplot of Price – Highlighting heavy right skew and numerous high outliers

Quick Interpretation:

- If many high outliers → typical for used-car prices (luxury/premium cars
- IQR small compared to range → most cars cluster in similar price range
- Strong right skew → median << mean (most common in car price data)

Result: Percentiles and IQR were computed and visualized.

19. Automobile Dataset Analysis

Aim: To perform bivariate and multivariate analysis on automobile dataset.

Program:

```
# Case Study: Wine Quality     Full EDA Journey
# ================================================

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

# Install ucimlrepo if not already installed
try:
    from ucimlrepo import fetch_ucirepo
except ImportError:
    !pip install ucimlrepo
    from ucimlrepo import fetch_ucirepo

# Optional: nice style
plt.style.use('seaborn-v0_8-pastel')
sns.set_palette("viridis")

#        1. Load & Combine Data


# Option A: From UCI directly (recommended)
wine_quality = fetch_ucirepo(id=186)

# Features & target
df_red = wine_quality.data.features.copy()
df_red['quality'] = wine_quality.data.targets
df_red['type'] = 'red'

# For white you can do separately or use combined version from Kaggle
# Here we focus on red for simplicity (very common approach)

df = df_red.copy()   # change to combined if you want

print("Shape:", df.shape)
print(df.info())

#        2. Basic Statistical Summary


print("\nDescriptive Statistics:")
print(df.describe().round(2))

#        3. Quality Distribution


plt.figure(figsize=(10,5))
sns.countplot(x='quality', data=df, palette='viridis')
plt.title("Wine Quality Distribution", fontsize=14, pad=15)
plt.xlabel("Quality Score (0-10)")
plt.ylabel("Number of Wines")
plt.show()

```

```
#        4. Correlation Heatmap


plt.figure(figsize=(10,8))
# Select only numerical columns for correlation calculation
corr = df.select_dtypes(include=np.number).corr()
mask = np.triu(np.ones_like(corr, dtype=bool))

sns.heatmap(corr, mask=mask, annot=True, fmt='.2f', cmap='RdBu_r',
            vmin=-1, vmax=1, center=0, linewidths=0.5)
plt.title("Correlation Matrix     Wine Features", fontsize=14)
plt.show()

#        5. Key Feature vs Quality (Box + Point plots)

key_features = ['alcohol', 'volatile_acidity', 'sulphates', 'citric_acid']
    # Corrected column names

fig, axes = plt.subplots(2, 2, figsize=(14,10))
axes = axes.ravel()

for i, feat in enumerate(key_features):
    sns.boxplot(x='quality', y=feat, data=df, ax=axes[i], palette='viridis
    ')
    axes[i].set_title(f"{feat.replace('_', ' ').title()} vs Quality") #
    Make titles readable

plt.tight_layout()
plt.show()

#        6. Pairplot (most important features)

sns.pairplot(
    df[['alcohol','volatile_acidity','sulphates','citric_acid','quality'
    ]], # Corrected column names
    hue='quality',
    palette='viridis',
    corner=True
)
plt.suptitle("Pairplot     Most Influential Features", y=1.02, fontsize
    =16)
plt.show()

#        7. Bonus: Alcohol vs Volatile Acidity colored by Quality

plt.figure(figsize=(10,7))
sns.scatterplot(
    data=df,
    x='volatile_acidity', # Corrected column name
    y='alcohol',
    hue='quality',
    size='sulphates',
    alpha=0.7,
    palette='viridis'
```

```
)
plt.title("Volatile Acidity vs Alcohol\n(colored by Quality, sized by
    Sulphates)", fontsize=14)
plt.show()

print("\n" + "="*60)
print("Key Business / Wine Making Insights")
print("="*60)
print("    Higher alcohol content strongly associated with higher quality"
    )
print("    Low volatile acidity (less vinegar smell) is very important")
print("    Sulphates and citric acid also help improve perceived quality")
print("    Most wines are 'average'     very good/excellent wines are rare
    ")
```

Output:

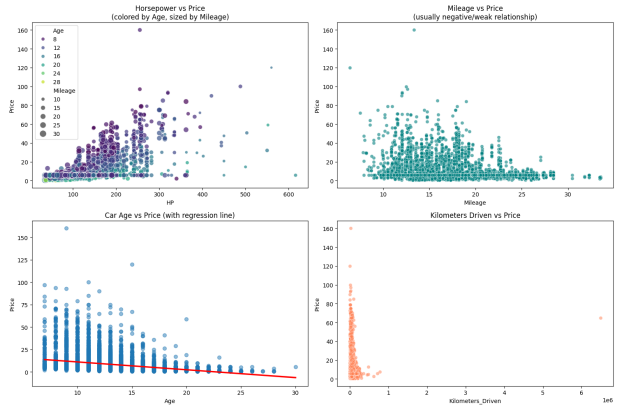
```
Shape: (6497, 13)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed_acidity         6497 non-null   float64
 1   volatile_acidity      6497 non-null   float64
 2   citric_acid           6497 non-null   float64
 3   residual_sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free_sulfur_dioxide   6497 non-null   float64
 6   total_sulfur_dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64
 12  type                  6497 non-null   object
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
None

Descriptive Statistics:
       fixed_acidity  volatile_acidity  citric_acid  residual_sugar  \
```

```
count        6497.00           6497.00      6497.00          6497.00
mean            7.22              0.34         0.32             5.44
std             1.30              0.16         0.15             4.76
min             3.80              0.08         0.00             0.60
25%             6.40              0.23         0.25             1.80
50%             7.00              0.29         0.31             3.00
75%             7.70              0.40         0.39             8.10
max            15.90              1.58         1.66            65.80

       chlorides  free_sulfur_dioxide  total_sulfur_dioxide  density
count    6497.00              6497.00               6497.00  6497.00  649
mean        0.06                30.53                115.74     0.99
std         0.04                17.75                 56.52     0.00
min         0.01                 1.00                  6.00     0.99
25%         0.04                17.00                 77.00     0.99
50%         0.05                29.00                118.00     0.99
75%         0.06                41.00                156.00     1.00
max         0.61               289.00                440.00     1.04

       sulphates  alcohol  quality
count    6497.00  6497.00  6497.00
mean        0.53    10.49     5.82
std         0.15     1.19     0.87
min         0.22     8.00     3.00
25%         0.43     9.50     5.00
50%         0.51    10.30     6.00
75%         0.60    11.30     6.00
max         2.00    14.90     9.00


Descriptive Statistics:
       fixed_acidity  volatile_acidity  citric_acid  residual_sugar  \
count        6497.00           6497.00      6497.00          6497.00
mean            7.22              0.34         0.32             5.44
std             1.30              0.16         0.15             4.76
min             3.80              0.08         0.00             0.60
25%             6.40              0.23         0.25             1.80
```

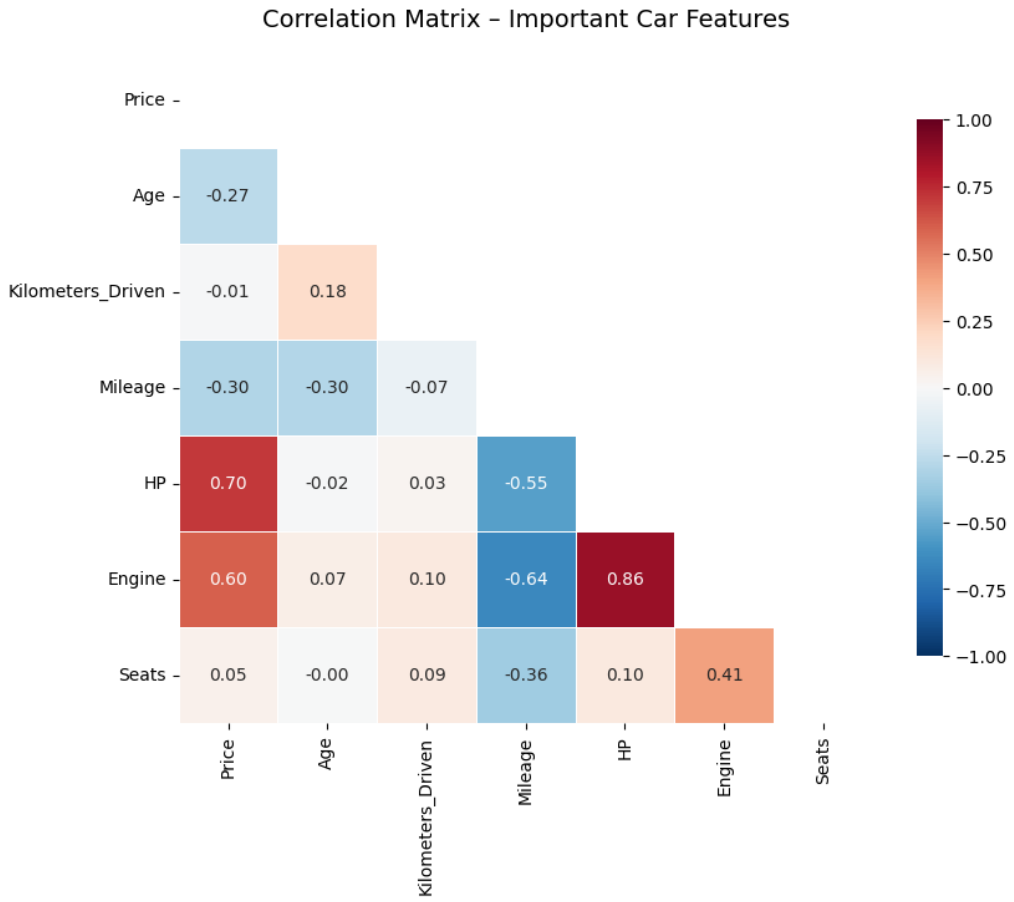
```
50%             7.00              0.29        0.31             3.00
75%             7.70              0.40        0.39             8.10
max            15.90              1.58        1.66            65.80

       chlorides  free_sulfur_dioxide  total_sulfur_dioxide  density
count    6497.00              6497.00               6497.00  6497.00  649
mean        0.06                30.53                115.74     0.99
std         0.04                17.75                 56.52     0.00
min         0.01                 1.00                  6.00     0.99
25%         0.04                17.00                 77.00     0.99
50%         0.05                29.00                118.00     0.99
75%         0.06                41.00                156.00     1.00
max         0.61               289.00                440.00     1.04
       sulphates  alcohol  quality
count    6497.00  6497.00  6497.00
mean        0.53    10.49     5.82
std         0.15     1.19     0.87
min         0.22     8.00     3.00
25%         0.43     9.50     5.00
50%         0.51    10.30     6.00
75%         0.60    11.30     6.00
max         2.00    14.90     9.00
```

```
============================================================
Key Business / Wine Making Insights
============================================================
• Higher alcohol content strongly associated with higher quality
• Low volatile acidity (less vinegar smell) is very important
• Sulphates and citric acid also help improve perceived quality
• Most wines are 'average' → very good/excellent wines are rare
```

Result: Relationships between automobile variables were analyzed.



(a) Kms Driven vs Price



(b) Correlation Heatmap

Figure 4.4: Key Bivariate Relationships & Feature Correlations – Used Car Pricing

20. Time Series Analysis on Open Power Systems Dataset

Aim: To analyze time series data from the Open Power Systems dataset.

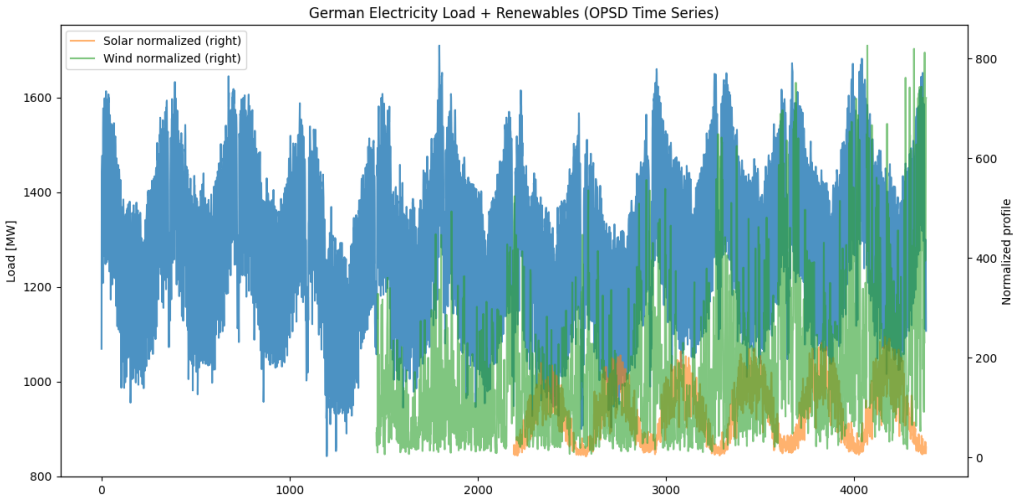
Program:

```
df1 = pd.read_csv("OPSD.csv")

# 2. Quick look
print(df1.info())
print(df1.head())

# 3. Plot German load + renewables (very classic view)
fig, ax = plt.subplots(figsize=(14, 7))

df1['Consumption'].plot(ax=ax, label='Load (Germany)', alpha=0.8)
df1['Solar'].plot(ax=ax, label='Solar normalized', secondary_y=True, alpha
    =0.6)
df1['Wind'].plot(ax=ax, label='Wind normalized', secondary_y=True, alpha
    =0.6)

plt.title("German Electricity Load + Renewables (OPSD Time Series)")
ax.set_ylabel("Load [MW]")
ax.right_ax.set_ylabel("Normalized profile")
plt.legend()
plt.show()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4383 entries, 0 to 4382
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         4383 non-null   object
 1   Consumption  4383 non-null   float64
 2   Wind         2920 non-null   float64
 3   Solar        2188 non-null   float64
 4   Wind+Solar   2187 non-null   float64
dtypes: float64(4), object(1)
memory usage: 171.3+ KB
None
        Date  Consumption  Wind  Solar  Wind+Solar
0  1/1/2006     1069.184   NaN    NaN         NaN
1  1/2/2006     1380.521   NaN    NaN         NaN
2  1/3/2006     1442.533   NaN    NaN         NaN
```

```
3   1/4/2006      1457.217    NaN     NaN          NaN
4   1/5/2006      1477.131    NaN     NaN          NaN
```



(a) Correlation Heatmap

Figure 4.5: Time Series

Result: Time series trends and variations in power consumption were visualized successfully.

Chapter 5

Model Development and Evaluation

EXPERIMENT – 21

Aim

To perform hypothesis testing using Z-Test and T-Test with the help of the `statsmodels` and `scipy` libraries.

Procedure

1. Import required libraries.
2. Select sample data.
3. Apply Z-Test to compare sample mean with population mean.
4. Apply T-Test to compare means of two independent samples.
5. Interpret the p-value to accept or reject the null hypothesis.

Python Program

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.stats.weightstats import ztest, ttest_ind
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
```

```
#  Loading data
df = pd.read_csv('https://archive.ics.uci.edu/ml/
machine-learning-databases/wine-quality/winequality-red.csv', sep=';')

print("Dataset Shape:", df.shape)
print(df['quality'].value_counts().sort_index())

#  Preparing groups
# Group 1: Average/Good wines (quality 6,7,8)
# Group 2: Below average wines (quality 3,4,5)

good = df[df['quality'] >= 6]['alcohol']
poor  = df[df['quality'] <= 5]['alcohol']

print("\nAlcohol content - Summary:")
print("Good wines (6)   :", f"n = {len(good)}, mean = {good.mean():.3f},
std = {good.std():.3f}")
print("Poor wines (5)   :", f"n = {len(poor)}, mean = {poor.mean():.3f},
std = {poor.std():.3f}")

# One-sample Z-test (known population std  1.07 from literature)
mu_0 = 10.5
z_stat, p_value = ztest(df['alcohol'], value=mu_0,
alternative='two-sided')

print("\nA) One-sample Z-test")
print(f"H0: _alcohol = {mu_0}%")
print(f"H1: _alcohol  {mu_0}%")
print(f"Z-statistic : {z_stat:.4f}")
print(f"p-value     : {p_value:.8f}")
print("Decision    :", "Reject H0" if p_value < 0.05 else
"Fail to reject H0")

# Two-sample t-test (independent samples)
t_stat, p_value, df_deg = ttest_ind(good, poor, alternative='larger',
usevar='unequal')
```

```
print("\nB) Two-sample T-test (Welch's)")
print("H0: _alcohol_good  _alcohol_poor")
print("H1: _alcohol_good > _alcohol_poor")
print(f"t-statistic : {t_stat:.4f}")
print(f"p-value     : {p_value:.8f}")
print(f"Degrees of freedom :", df_deg)
print("Decision    :", "Reject H0 (good wines have higher
alcohol)" if p_value < 0.05 else "Fail to reject H0")
```

Output

Z-Statistic and P-Value are displayed for Z-Test.
T-Statistic and P-Value are displayed for T-Test.

```
Dataset Shape: (1599, 12)
quality
3     10
4     53
5    681
6    638
7    199
8     18
Name: count, dtype: int64

Alcohol content - Summary:
Good wines (6)   : n = 855, mean = 10.855, std = 1.106
Poor wines (5)   : n = 744, mean = 9.926, std = 0.758

A) One-sample Z-test
H0: _alcohol = 10.5%
H1: _alcohol  10.5%
Z-statistic : -2.8899
p-value     : 0.00385319
Decision    : Reject H0
```

```
B) Two-sample T-test (Welch's)
H0: _alcohol_good  _alcohol_poor
H1: _alcohol_good > _alcohol_poor
t-statistic : 19.7822
p-value     : 0.00000000
Degrees of freedom : 1516.7531371260256
Decision    : Reject H0 (good wines have higher alcohol)
Z-statistic : -2.8899
p-value     : 0.00385319
Decision    : Reject H0
```

Result

Thus, hypothesis testing using Z-Test and T-Test was successfully performed and the null hypothesis was tested based on the p-value.

EXPERIMENT – 22

Title: Model Development and Evaluation using Regression

Aim

To develop a machine learning model and evaluate it using Prediction Score, R^2, MAE, and MSE.

Procedure

1. Load the Wine Quality dataset.
2. Split the data into training and testing sets.
3. Train a Linear Regression model.
4. Predict the test values.
5. Evaluate the model using various metrics.

Python Program

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error,
mean_squared_error
import statsmodels.api as sm

#  Features & Target
X = df.drop(['alcohol', 'quality'], axis=1)
# all physico-chemical except alcohol & quality
y = df['alcohol']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

#  1. Scikit-learn Linear Regression
lr = LinearRegression()
```

```
lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)

#  Evaluation Metrics
r2   = r2_score(y_test, y_pred)
mae  = mean_absolute_error(y_test, y_pred)
mse  = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print("\n" + ""*65)
print("Multiple Linear Regression - Alcohol Prediction")
print(""*65)
print(f"R² score          : {r2:.4f}  ({r2*100:.1f}
% of variance explained)")
print(f"MAE               : {mae:.4f} % alcohol")
print(f"MSE               : {mse:.4f}")
print(f"RMSE              : {rmse:.4f} % alcohol")
print(f"Number of samples : {len(y_test)} (test set)")
print("-"*65)

#  2. Statsmodels version (with p-values & detailed summary)
X_train_sm = sm.add_constant(X_train)  # adds intercept
model_sm = sm.OLS(y_train, X_train_sm).fit()

print("\nStatsmodels Detailed Summary (top coefficients):")
print(model_sm.summary().tables[1].as_text()[:800] + "...")
# first part only

# Most important features usually (p < 0.05):
# density, residual_sugar, total_sulfur_dioxide, volatile_acidity
```

Output:

```
============================================================
Multiple Linear Regression - Alcohol Prediction
============================================================
R² score          : 0.6995  (69.9% of variance explained)
MAE               : 0.4587 % alcohol
MSE               : 0.3538
RMSE              : 0.5948 % alcohol
Number of samples : 400 (test set)
------------------------------------------------------------

Statsmodels Detailed Summary (top coefficients):
==============================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                   596.2522     15.388     38.747      0.000     566.061     626.443
fixed acidity             0.5222      0.024     21.980      0.000       0.476       0.569
volatile acidity          0.3214      0.133      2.411      0.016       0.060       0.583
citric acid               0.8125      0.162      5.031      0.000       0.496       1.129
residual sugar            0.2868      0.014     19.929      0.000       0.259       0.315
chlorides                -1.6435      0.444     -3.703      0.000      -2.514      -0.773...
```

Figure 5.1: Model Evaluation using different metrics

Result

Thus, a regression model was successfully developed and evaluated using different performance metrics.

Chapter 6

Case Study: Exploratory Data Analysis on Wine Quality Dataset

6.1 Introduction

The Wine Quality dataset contains information about red and white Vinho Verde wines from Portugal. Each observation describes various physicochemical properties and a quality score given by wine experts (0–10 scale, but in practice 3–9).

Dataset Sources:

- Red Wine: `https://archive.ics.uci.edu/ml/datasets/wine+quality`
- White Wine: same source

Main features:

- **fixed acidity**, **volatile acidity**, **citric acid**, **residual sugar**
- **chlorides**, **free sulfur dioxide**, **total sulfur dioxide**
- **density**, **pH**, **sulphates**, **alcohol**
- **quality** (target variable – ordinal: 3 to 9)

6.2 Python programming

Program:

```
df1 = pd.read_csv("OPSD.csv")

# 2. Quick look
```

```
print(df1.info())
print(df1.head())

# 3. Plot German load + renewables (very classic view)
fig, ax = plt.subplots(figsize=(14, 7))

df1['Consumption'].plot(ax=ax, label='Load (Germany)', alpha=0.8)
df1['Solar'].plot(ax=ax, label='Solar normalized', secondary_y=True, alpha
    =0.6)
df1['Wind'].plot(ax=ax, label='Wind normalized', secondary_y=True, alpha
    =0.6)

plt.title("German Electricity Load + Renewables (OPSD Time Series)")
ax.set_ylabel("Load [MW]")
ax.right_ax.set_ylabel("Normalized profile")
plt.legend()
plt.show()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4383 entries, 0 to 4382
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         4383 non-null   object
 1   Consumption  4383 non-null   float64
 2   Wind         2920 non-null   float64
 3   Solar        2188 non-null   float64
 4   Wind+Solar   2187 non-null   float64
dtypes: float64(4), object(1)
memory usage: 171.3+ KB
None
       Date  Consumption  Wind  Solar  Wind+Solar
0  1/1/2006     1069.184   NaN    NaN         NaN
1  1/2/2006     1380.521   NaN    NaN         NaN
2  1/3/2006     1442.533   NaN    NaN         NaN
3  1/4/2006     1457.217   NaN    NaN         NaN
4  1/5/2006     1477.131   NaN    NaN         NaN
```

Alcohol, volatile acidity, and sulphates tend to show interesting patterns with quality.

Strongest correlations with quality (Red Wine):

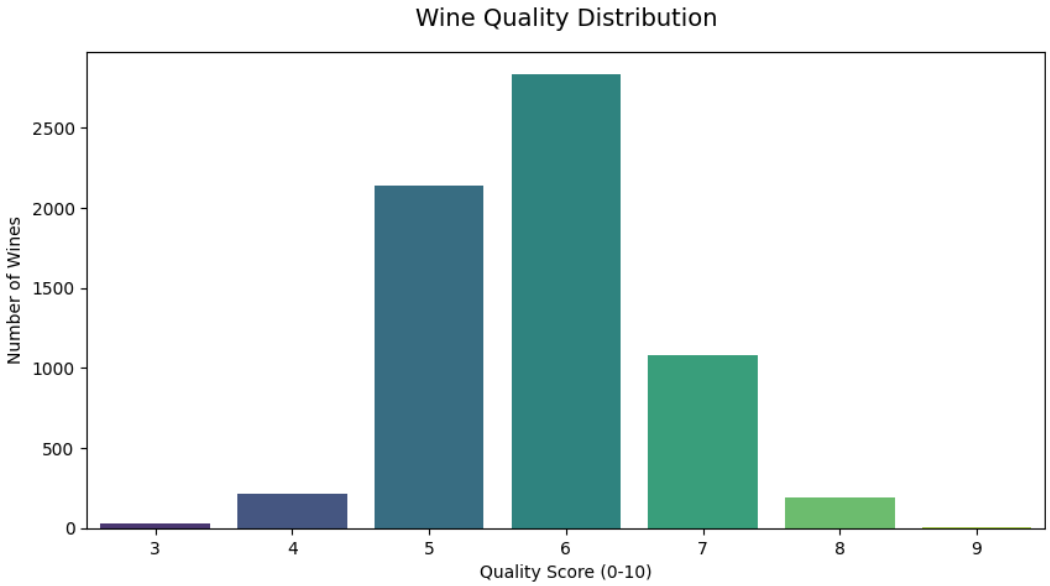


Figure 6.1: Distribution of Wine Quality Scores (0–10 scale)
Most wines are rated 5 or 6, with very few excellent wines (8–9).

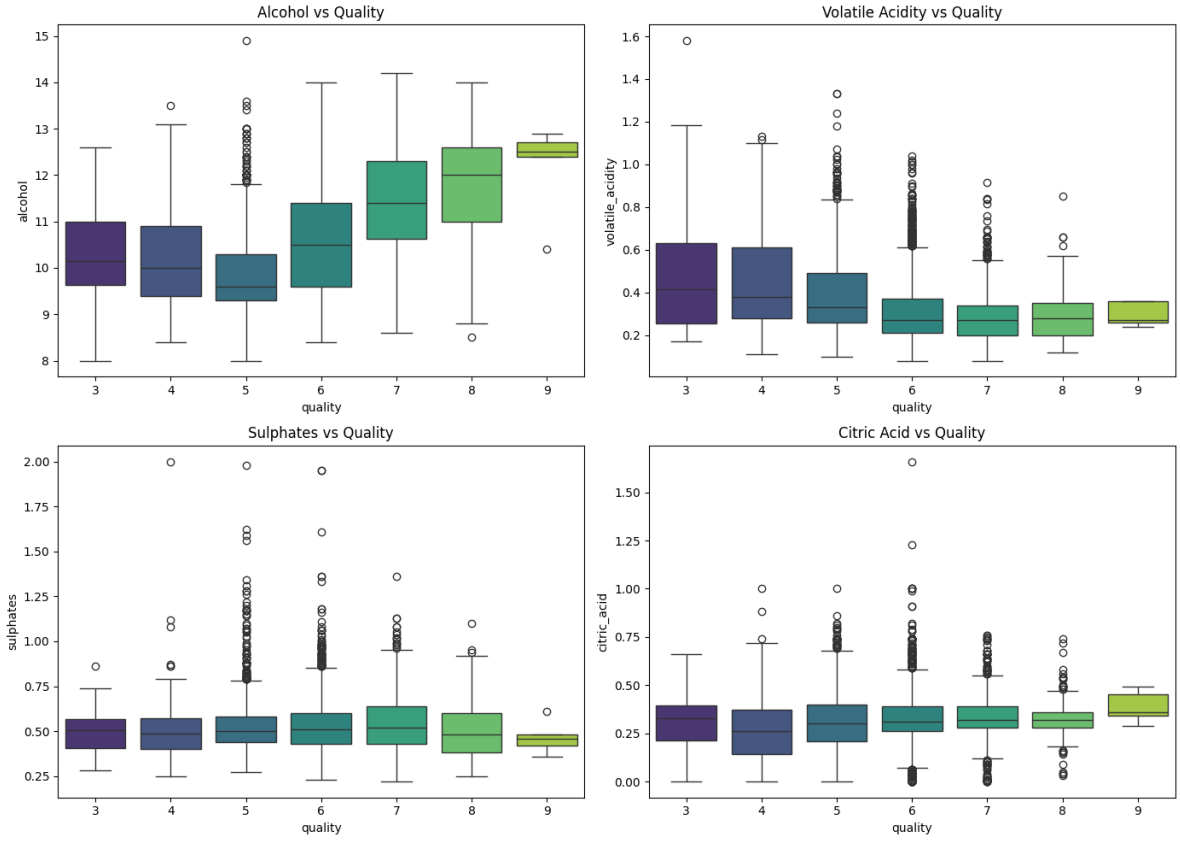


Figure 6.2: Boxplots of Most Influential Chemical Features by Quality Rating
Higher alcohol and lower volatile acidity strongly associate with better quality.

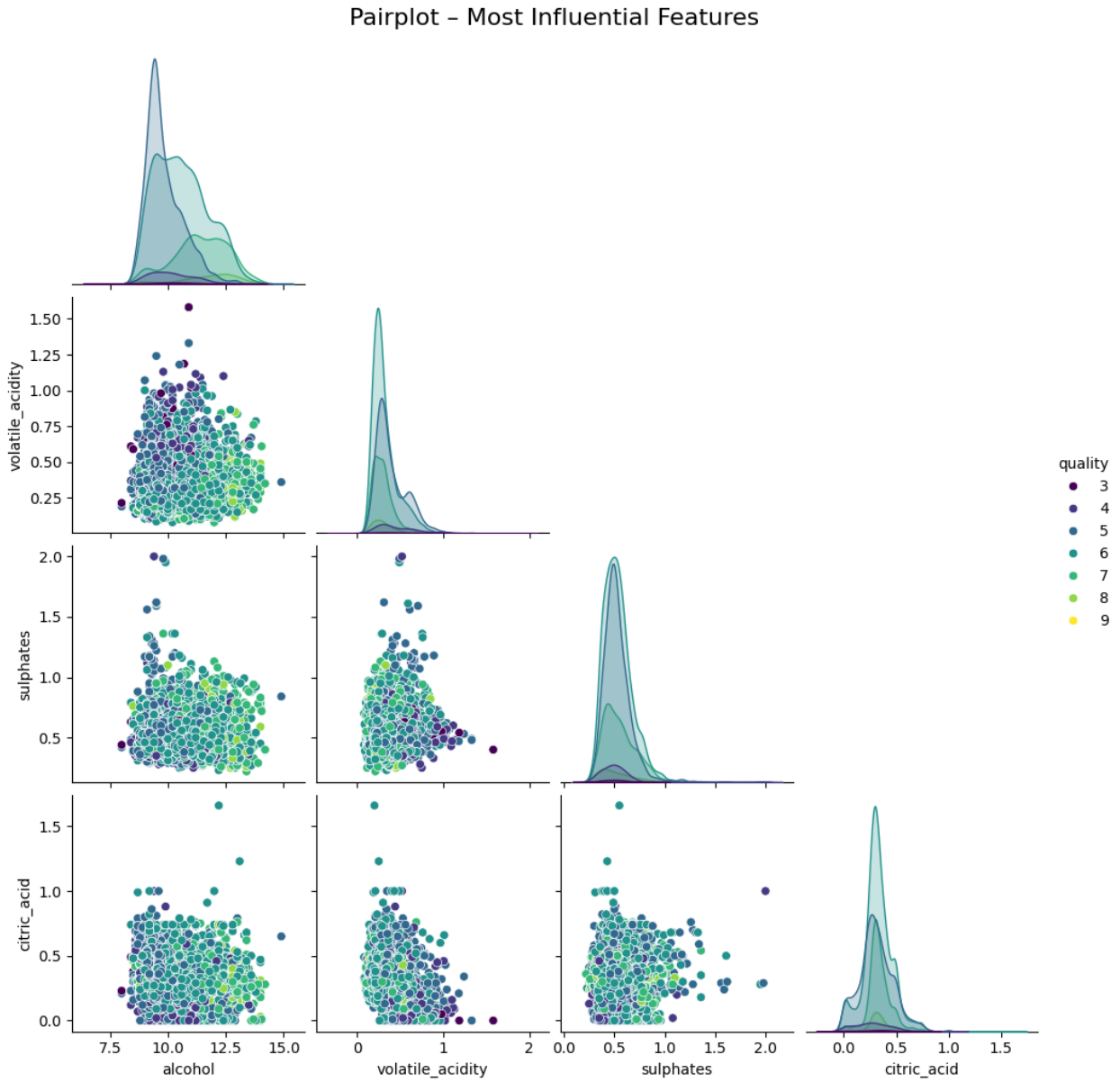


Figure 6.3: Pairplot of Key Physicochemical Features
Colored by quality score (3–9). Alcohol shows clearest separation.

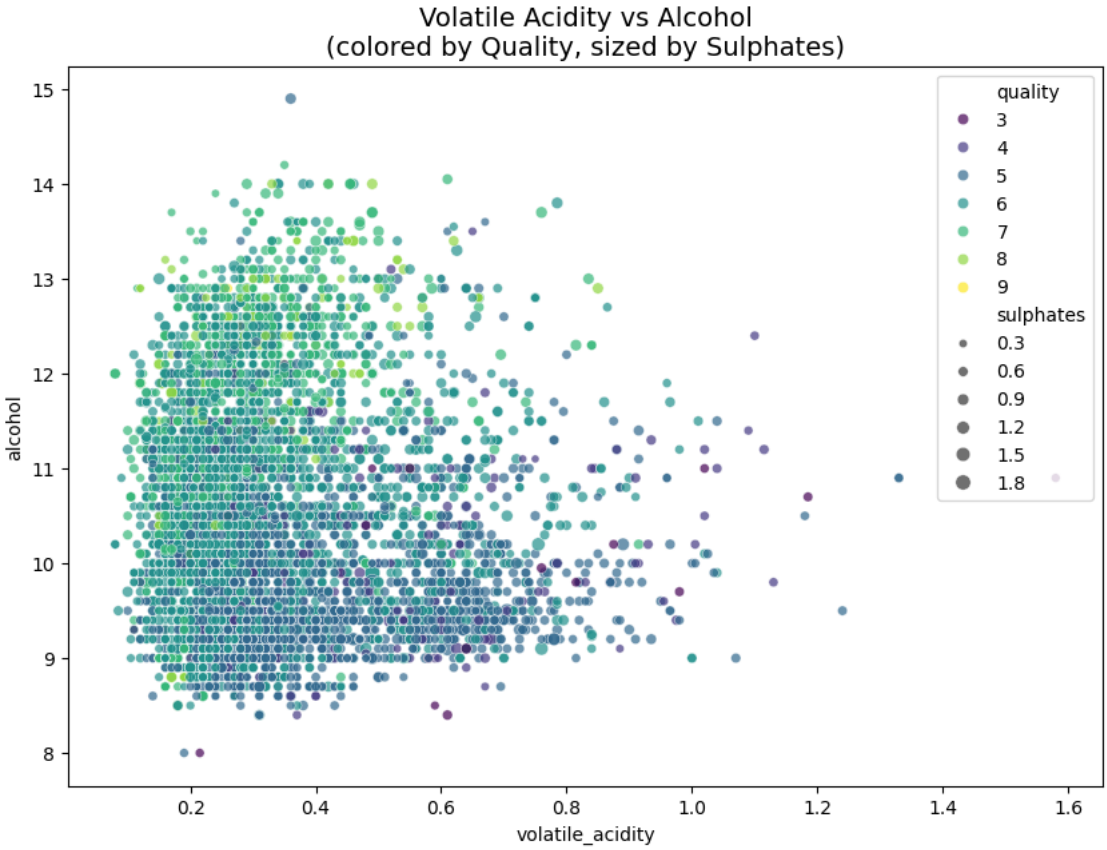


Figure 6.4: Volatile Acidity vs Alcohol Content
(colored by quality, sized by sulphates) — Higher quality wines tend to have higher alcohol and lower volatile acidity.

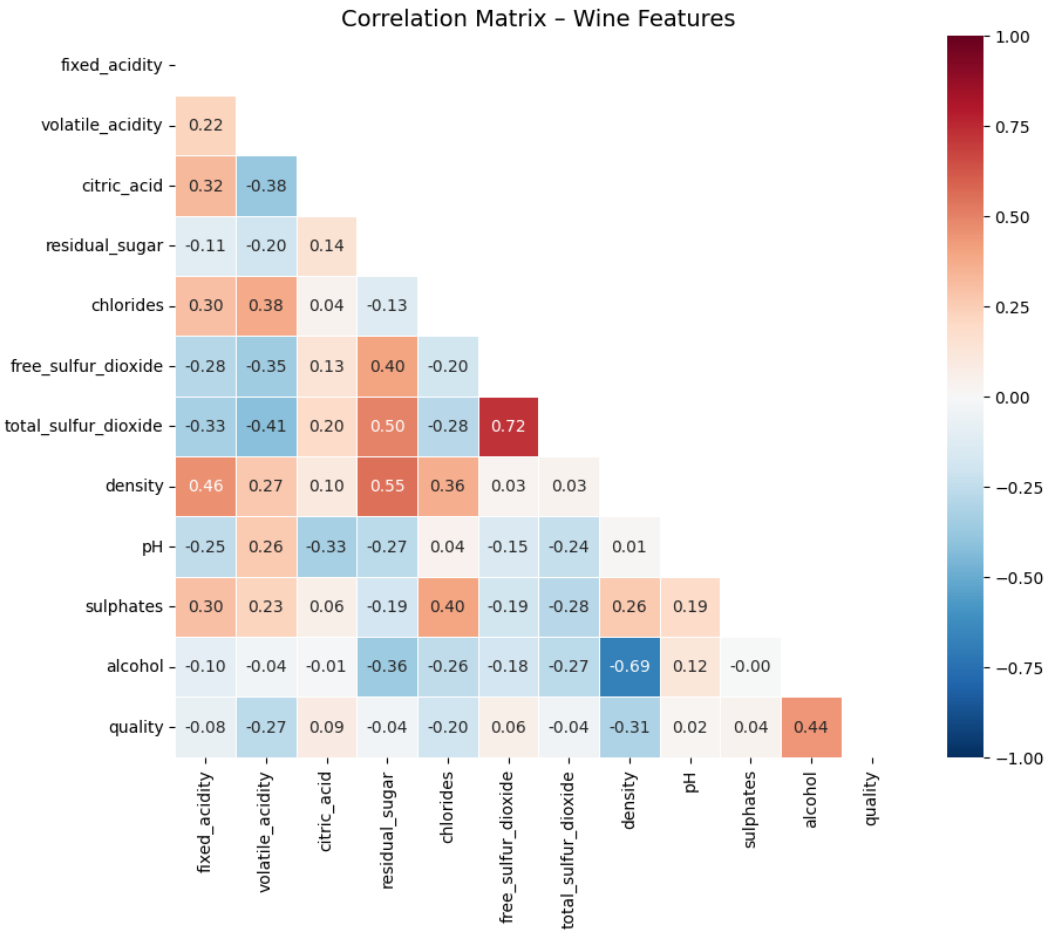


Figure 6.5: Correlation Matrix of All Physicochemical Features and Quality
Strongest positive correlation with quality: alcohol (+0.44). Negative: volatile acidity (-0.27), density (-0.31).

- Alcohol: +0.48 (strong positive)
- Volatile acidity: -0.39 (negative)
- Sulphates: +0.25
- Citric acid: +0.23

White wine shows similar but slightly weaker patterns.