GITHUB Repository link: https://github.com/HarshArya1565/Exploratory-Data-Analysis

HARSH ARYA 21BDS0098

MODULE 2

```
import pandas as pd

# Assuming the last four digits of the Roll Number (R.no) are 0098
roll_number_last4_0098 = 98  # last 4 digits of your Roll Number
(0098)

# DataFrame 1: Fname - ProductID, NameofProd, Price
HarshArya1 = {
    'ProductID': [roll_number_last4, roll_number_last4 + 1,
roll_number_last4 + 2, roll_number_last4 + 3, roll_number_last4 + 4],
    'NameofProd': ['Product_A', 'Product_B', 'Product_C', 'Product_D',
'Product_E'],
    'Price': [100, 200, 300, 400, 500]
}

df1_0098 = pd.DataFrame(HarshArya1)

# DataFrame 2: Lname - ProductID, Company_Location
HarshArya2 = {
    'ProductID': [roll_number_last4, roll_number_last4 + 1,
roll_number_last4 + 2, roll_number_last4 + 3, roll_number_last4 + 4],
    'Company_Location': ['Location_1', 'Location_2', 'Location_3',
'Location_4', 'Location_5']
}

df2_0098 = pd.DataFrame(HarshArya2)

# Show both DataFrames
print("DataFrame 1 (Fname):")
print(df1_0098)
print("\nDataFrame 2 (Lname):")
print(df2_0098)

# 1. Append (rows): Append df2 to df1 using pd.concat() (This will
```

```
work if the columns are the same in both DataFrames)
df_append_0098 = pd.concat([df1_0098, df2_0098], ignore_index=True)
print("\nDataFrame after Appending df2 to df1 using pd.concat():")
print(df_append_0098)

# 2. Concatenate: Concatenate df1 and df2 by columns (aligning them
side by side)
df_concat_0098 = pd.concat([df1_0098, df2_0098], axis=1)
print("\nDataFrame after Concatenating df1 and df2:")
print(df_concat_0098)

# 3. Merge: Merge df1 and df2 on 'ProductID' (similar to SQL join)
df_merge_0098 = pd.merge(df1_0098, df2_0098, on='ProductID',
how='inner')
print("\nDataFrame after Merging df1 and df2 on 'ProductID':")
print(df_merge_0098)

# 4. Join: Join df1 and df2 on the index (this will work only if the
indexes align)
df_join_0098 =
df1_0098.set_index('ProductID').join(df2_0098.set_index('ProductID'))
print("\nDataFrame after Joining df1 and df2 on the index:")
print(df_join_0098)
```

```
DataFrame 1 (Fname):
   ProductID NameofProd  Price
0         98  Product_A    100
1         99  Product_B    200
2        100  Product_C    300
3        101  Product_D    400
4        102  Product_E    500

DataFrame 2 (Lname):
   ProductID Company_Location
0         98       Location_1
1         99       Location_2
2        100       Location_3
3        101       Location_4
4        102       Location_5

DataFrame after Appending df2 to df1 using pd.concat():
   ProductID NameofProd  Price Company_Location
0         98  Product_A  100.0              NaN
1         99  Product_B  200.0              NaN
2        100  Product_C  300.0              NaN
3        101  Product_D  400.0              NaN
4        102  Product_E  500.0              NaN
5         98        NaN    NaN       Location_1
6         99        NaN    NaN       Location_2
7        100        NaN    NaN       Location_3
```

```
8        101        NaN    NaN       Location_4
9        102        NaN    NaN       Location_5

DataFrame after Concatenating df1 and df2:
   ProductID NameofProd  Price  ProductID Company_Location
0         98  Product_A    100         98       Location_1
1         99  Product_B    200         99       Location_2
2        100  Product_C    300        100       Location_3
3        101  Product_D    400        101       Location_4
4        102  Product_E    500        102       Location_5

DataFrame after Merging df1 and df2 on 'ProductID':
   ProductID NameofProd  Price Company_Location
0         98  Product_A    100       Location_1
1         99  Product_B    200       Location_2
2        100  Product_C    300       Location_3
3        101  Product_D    400       Location_4
4        102  Product_E    500       Location_5

DataFrame after Joining df1 and df2 on the index:
          NameofProd  Price Company_Location
ProductID
98         Product_A    100       Location_1
99         Product_B    200       Location_2
100        Product_C    300       Location_3
101        Product_D    400       Location_4
102        Product_E    500       Location_5
```

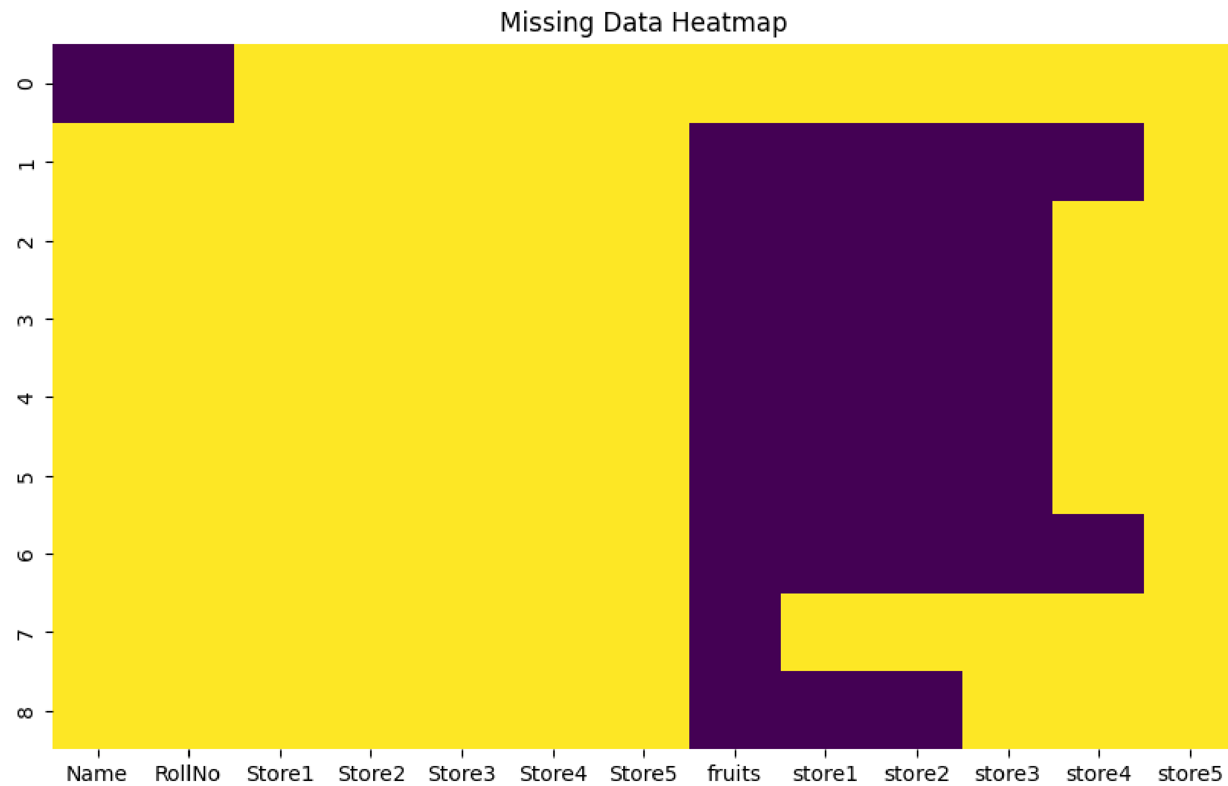
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Load the fruits.csv file into Python (make sure to specify the
correct file path)
HarshArya = pd.read_csv("fruits.csv")

# Display the first few rows of the dataframe
print("Original DataFrame:")
print(HarshArya.head())

# 2. Insert the first record as 'Your name, Rollno (last 4 digits),
roll no, roll no, roll no, NaN)
# Assuming your name is 'John Doe' and roll number last 4 digits are
0098 (change as needed)
first_record_0098 = {
    'Name': 'John Doe',
    'RollNo': 98,
    'Store1': np.nan,
    'Store2': np.nan,
```

```
    'Store3': np.nan,
    'Store4': np.nan,
    'Store5': np.nan
}

# Convert the first record to a DataFrame and concatenate it with the
existing DataFrame
first_record_df_0098 = pd.DataFrame([first_record_0098])
HarshArya = pd.concat([first_record_df_0098, HarshArya],
ignore_index=True)

# Display the updated dataframe
print("\nUpdated DataFrame with First Record:")
print(HarshArya.head())

# 3. Check if there exists any NA (missing values) in the dataset
any_na_0098 = fruits.isna().any().any()  # Check if any NaN value
exists in the dataframe
print(f"\nDoes the dataset contain any missing values? {any_na_0098}")

# 4. Finding the missing summaries of the dataset (summary of missing
values per column)
missing_summary_0098 = HarshArya.isna().sum()
print("\nMissing Values Summary (per column):")
print(missing_summary_0098)

# 5. Find the total number of NA (missing values) in the dataset
total_na_0098 = HarshArya.isna().sum().sum()  # Total missing values
in the dataframe
print(f"\nTotal number of missing values in the dataset:
{total_na_0098}")

# 6. Count the total number of complete cases in Store4 and Store5
complete_cases_0098 = HarshArya[['Store4',
'Store5']].notna().all(axis=1).sum()
print(f"\nTotal number of complete cases in Store4 and Store5:
{complete_cases_0098}")

# 7. Proportion of missing and complete data
prop_missing_0098 = HarshArya.isna().mean()  # Proportion of missing
values per column
prop_complete_0098 = 1 - prop_missing_0098  # Proportion of complete
values per column
print("\nProportion of Missing Values per Column:")
print(prop_missing_0098)
print("\nProportion of Complete Values per Column:")
print(prop_complete_0098)

# 8. Display the missing values per column for each observation
# Create a missing data heatmap
```

```python
plt.figure(figsize=(10, 6))
sns.heatmap(HarshArya.isna(), cbar=False, cmap='viridis')
plt.title("Missing Data Heatmap")
plt.show()

# 9. Performing row-wise deletion (drop rows with any missing values)
fruits_cleaned_0098 = HarshArya.dropna()

# Display the cleaned dataframe
print("\nDataFrame after Row-wise Deletion (NaN Rows Removed):")
print(fruits_cleaned_0098.head())

# Optionally, check the number of rows before and after row deletion
print(f"\nNumber of rows before deletion: {len(HarshArya)}")
print(f"Number of rows after deletion: {len(fruits_cleaned_0098)}")
```

```
Original DataFrame:
   fruits  store1  store2  store3  store4  store5
0   apple    15.0    16.0    17.0    20.0     NaN
1  banana    18.0    19.0    20.0     NaN     NaN
2    kiwi    21.0    22.0    23.0     NaN     NaN
3  grapes    24.0    25.0    26.0     NaN     NaN
4   mango    27.0    28.0    29.0     NaN     NaN

Updated DataFrame with First Record:
       Name  RollNo  Store1  Store2  Store3  Store4  Store5  fruits  
store1  \
0  John Doe    98.0     NaN     NaN     NaN     NaN     NaN     NaN  
NaN
1       NaN     NaN     NaN     NaN     NaN     NaN     NaN   apple  
15.0
2       NaN     NaN     NaN     NaN     NaN     NaN     NaN  banana  
18.0
3       NaN     NaN     NaN     NaN     NaN     NaN     NaN    kiwi  
21.0
4       NaN     NaN     NaN     NaN     NaN     NaN     NaN  grapes  
24.0

   store2  store3  store4  store5
0     NaN     NaN     NaN     NaN
1    16.0    17.0    20.0     NaN
2    19.0    20.0     NaN     NaN
3    22.0    23.0     NaN     NaN
4    25.0    26.0     NaN     NaN

Does the dataset contain any missing values? True

Missing Values Summary (per column):
Name      8
RollNo    8
```

```
Store1    9
Store2    9
Store3    9
Store4    9
Store5    9
fruits    1
store1    2
store2    2
store3    3
store4    7
store5    9
dtype: int64

Total number of missing values in the dataset: 85

Total number of complete cases in Store4 and Store5: 0

Proportion of Missing Values per Column:
Name      0.888889
RollNo    0.888889
Store1    1.000000
Store2    1.000000
Store3    1.000000
Store4    1.000000
Store5    1.000000
fruits    0.111111
store1    0.222222
store2    0.222222
store3    0.333333
store4    0.777778
store5    1.000000
dtype: float64

Proportion of Complete Values per Column:
Name      0.111111
RollNo    0.111111
Store1    0.000000
Store2    0.000000
Store3    0.000000
Store4    0.000000
Store5    0.000000
fruits    0.888889
store1    0.777778
store2    0.777778
store3    0.666667
store4    0.222222
store5    0.000000
dtype: float64
```




```
DataFrame after Row-wise Deletion (NaN Rows Removed):
Empty DataFrame
Columns: [Name, RollNo, Store1, Store2, Store3, Store4, Store5,
fruits, store1, store2, store3, store4, store5]
Index: []

Number of rows before deletion: 9
Number of rows after deletion: 0

# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the Dataset
# Load the dataset (Replace 'path_to_file.csv' with your actual file
path)
HarshArya = pd.read_csv("data(1).csv")

# Step 2: Convert all column names to lowercase
HarshArya.columns = HarshArya.columns.str.lower()

# Step 3: Replace '?' with NaN for all columns
HarshArya.replace('?', np.nan, inplace=True)
```

```
# Step 4: Display the number of rows and columns in the dataset
print("Dimensions of the dataset:", HarshArya.shape)

# Step 5: Display the header or attribute names from the dataset
print("Column names:", HarshArya.columns)

# Step 6: Display the structure of the dataset (info)
print("Structure of the dataset:")
HarshArya.info()

# Step 7: View the first 3 and last 3 rows of the dataset
print("First 3 rows of the dataset:")
print(HarshArya.head(3))

print("Last 3 rows of the dataset:")
print(HarshArya.tail(3))

# Step 8: Deleting Specific Columns (Fuel-System and Bore) by Index
and Select Function
# Deleting columns 'fuel-system' and 'bore' by column index (assuming
known column indices)
HarshArya = HarshArya.drop(HarshArya.columns[[9, 11]], axis=1)  #
Adjust indices accordingly

# Deleting columns using pandas' `drop` (select function not needed in
pandas)
HarshArya = HarshArya.drop(columns=['fuel-system', 'bore'])

# Step 9: Displaying Summary Statistics
print("Summary Statistics of the dataset:")
print(HarshArya.describe())

# Step 10: Data Cleaning

# 10.1 Find out the number of values that are not numeric in 'price',
'horsepower', and 'normalized-losses'
non_numeric_price_0098 = HarshArya['price'].apply(pd.to_numeric,
errors='coerce').isna().sum()
non_numeric_horsepower_0098 =
HarshArya['horsepower'].apply(pd.to_numeric,
errors='coerce').isna().sum()
non_numeric_normalized_losses_0098 = HarshArya['normalized-
losses'].apply(pd.to_numeric, errors='coerce').isna().sum()

print(f"Non-numeric values in 'price': {non_numeric_price_0098}")
print(f"Non-numeric values in 'horsepower':
{non_numeric_horsepower_0098}")
print(f"Non-numeric values in 'normalized-losses':
{non_numeric_normalized_losses_0098}")
```

```
# 10.2 Setting the missing value in 'price' to the mean and converting
to numeric
HarshArya['price'] = pd.to_numeric(HarshArya['price'],
errors='coerce')
mean_price_0098 = HarshArya['price'].mean()
HarshArya['price'].fillna(mean_price_0098, inplace=True)

# Step 11: Compute Measures of Central Tendency and Dispersion for
'height' Column

# 11.1 Central Tendency: Mean, Median, Mode
mean_height_0098 = HarshArya['height'].mean()
median_height_0098 = HarshArya['height'].median()
mode_height_0098 = HarshArya['height'].mode()[0]

print(f"Mean of height: {mean_height_0098}")
print(f"Median of height: {median_height_0098}")
print(f"Mode of height: {mode_height_0098}")

# 11.2 Dispersion: Standard Deviation and Variance
sd_height_0098 = HarshArya['height'].std()
var_height_0098 = HarshArya['height'].var()

print(f"Standard Deviation of height: {sd_height_0098}")
print(f"Variance of height: {var_height_0098}")

# 11.3 Quartile Ranges and IQR
height_quantiles_0098 = HarshArya['height'].quantile([0.25, 0.5,
0.75])
iqr_height_0098 = height_quantiles_0098[0.75] -
height_quantiles_0098[0.25]

print(f"Quartiles of height:\n{height_quantiles_0098}")
print(f"Interquartile Range (IQR) of height: {iqr_height_0098}")

# Step 12: Calculate Correlation Between Price and Horsepower
cor_price_hp = HarshArya[['price', 'horsepower']].corr().iloc[0, 1]
print(f"Correlation between price and horsepower: {cor_price_hp}")

# Step 13: Univariate Analysis (Plots)

# 13.1 Distribution Plot: Histogram for height
plt.figure(figsize=(8, 6))
plt.hist(HarshArya['height'], bins=20, color='lightblue',
edgecolor='black')
plt.title('21BDS0098 - Height Distribution Plot')
plt.xlabel('Height')
plt.ylabel('Frequency')
plt.show()
```

```python
# Histogram for height using Seaborn
sns.histplot(HarshArya['height'], kde=False, bins=20,
color='lightblue')
plt.title('21BDS0098 - Height Distribution Plot')
plt.xlabel('Height')
plt.ylabel('Frequency')
plt.show()

# 13.2 Distribution Plot Histogram for price
plt.figure(figsize=(8, 6))
plt.hist(HarshArya['price'], bins=20, color='lightgreen',
edgecolor='black')
plt.title('21BDS0098 - Price Distribution Plot')
plt.xlabel('price')
plt.ylabel('Frequency')
plt.show()

# Histogram for price using Seaborn
sns.histplot(HarshArya['price'], kde=False, bins=20,
color='lightgreen')
plt.title('21BDS0098 - Price Distribution Plot')
plt.xlabel('price')
plt.ylabel('Frequency')
plt.show()

# 13.3 Distribution Plot Density for price
plt.figure(figsize=(8, 6))
sns.kdeplot(HarshArya['price'], shade=True, color='purple')
plt.title('21BDS0098 - Price Density Plot')
plt.xlabel('price')
plt.ylabel('Density')
plt.show()

# 13.4 Distribution Plot (Histogram + Density for price)
plt.figure(figsize=(8, 6))
sns.histplot(HarshArya['price'], kde=True, bins=20,
color='lightgreen', line_kws={'color': 'purple'})
plt.title('21BDS0098 - Price Histogram and Density Plot')
plt.xlabel('price')
plt.ylabel('Frequency/Density')
plt.show()

# 13.5 Boxplot for price
plt.figure(figsize=(8, 6))
sns.boxplot(x=HarshArya['price'], color='lightblue')
plt.title('21BDS0098 - Price Boxplot')
plt.xlabel('price')
plt.show()
```

```
# 13.6 Display a Barplot for 'no-of-cylinders' (Vertical and
Horizontal)

# Check if 'no-of-cylinders' exists in the column names
if 'no-of-cylinders' in HarshArya.columns:
    # Vertical Barplot
    plt.figure(figsize=(8, 6))
    sns.countplot(x='no-of-cylinders', HarshArya=HarshArya,
palette='Blues')
    plt.title('21BDS0098 - Barplot of Number of Cylinders')
    plt.xlabel('Number of Cylinders')
    plt.ylabel('Count')
    plt.show()

    # Horizontal Barplot
    plt.figure(figsize=(8, 6))
    sns.countplot(y='no-of-cylinders', HarshArya=HarshArya,
palette='Blues')
    plt.title('21BDS0098 - Barplot of Number of Cylinders')
    plt.xlabel('Count')
    plt.ylabel('Number of Cylinders')
    plt.show()

else:
    print("'no-of-cylinders' column not found in the dataset.")
    # You can check other columns that might be similar or adjust the
code accordingly
    print("Available columns:", HarshArya.columns)

# 13.7 Display Pie Plot for Drive-Wheel
drive_wheel_counts = HarshArya['drive-wheels'].value_counts()
plt.figure(figsize=(8, 6))
plt.pie(drive_wheel_counts, labels=drive_wheel_counts.index,
autopct='%1.1f%%', startangle=90, colors=sns.color_palette('Set2'))
plt.title('21BDS0098 - Pie Chart for Drive-Wheel')
plt.axis('equal')  # Equal aspect ratio ensures the pie is drawn as a
circle.
plt.show()

# 13.8 Display Dot Plot for 'price'
plt.figure(figsize=(8, 6))
sns.stripplot(x=HarshArya['price'], color='purple', jitter=True,
size=6)
plt.title('21BDS0098 - Dot Plot for Price')
plt.xlabel('Price')
plt.show()
```

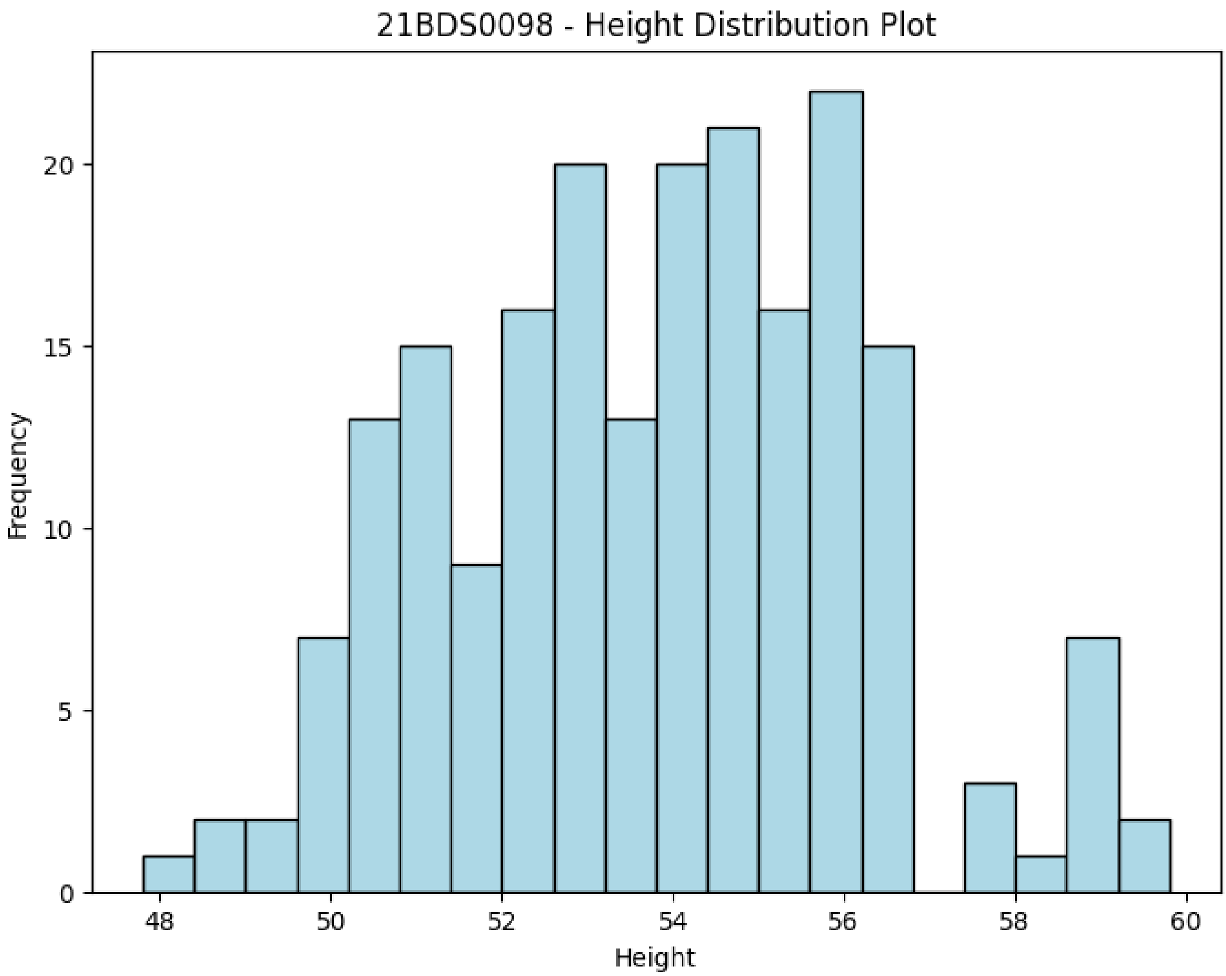
```
Dimensions of the dataset: (205, 26)
Column names: Index(['symboling', 'normalized-losses', 'make', 'fuel-
type', 'aspiration',
```

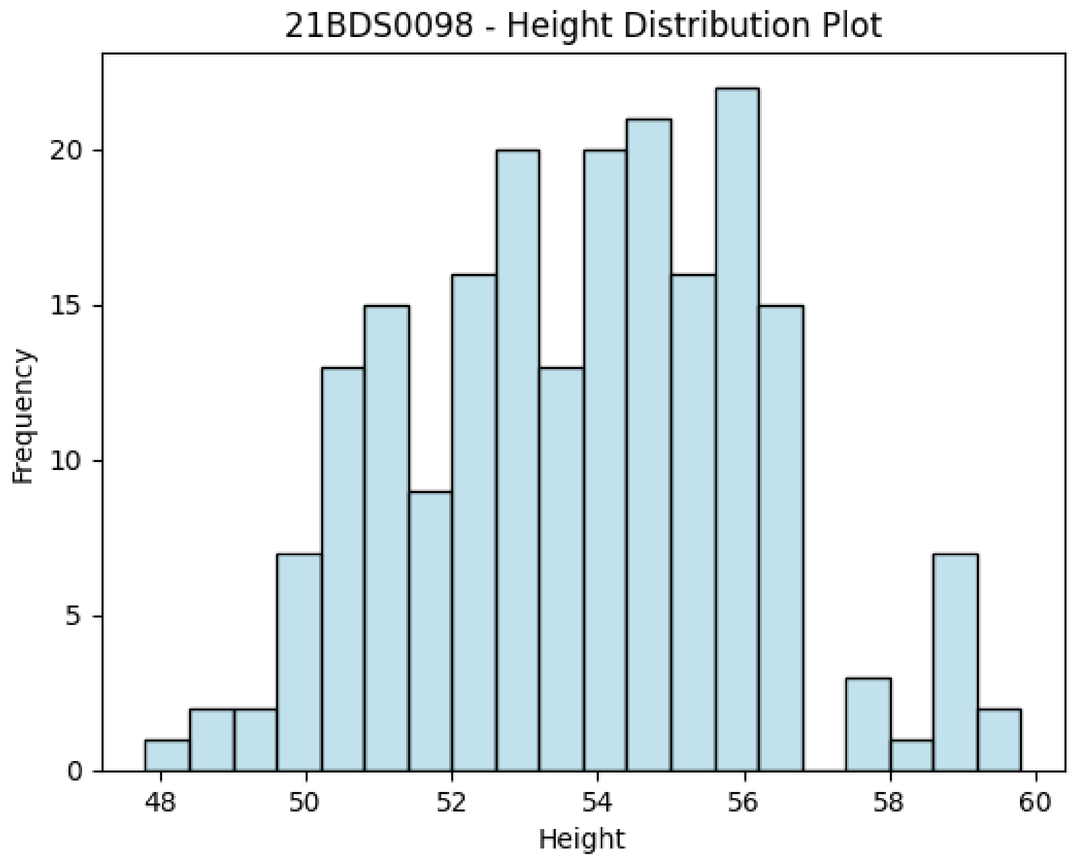
```
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-
location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight',
'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore',
'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price'],
      dtype='object')
Structure of the dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          205 non-null    int64  
 1   normalized-losses  164 non-null    object 
 2   make               205 non-null    object 
 3   fuel-type          205 non-null    object 
 4   aspiration         205 non-null    object 
 5   num-of-doors       203 non-null    object 
 6   body-style         205 non-null    object 
 7   drive-wheels       205 non-null    object 
 8   engine-location    205 non-null    object 
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64  
 14  engine-type        205 non-null    object 
 15  num-of-cylinders   205 non-null    object 
 16  engine-size        205 non-null    int64  
 17  fuel-system        205 non-null    object 
 18  bore               201 non-null    object 
 19  stroke             201 non-null    object 
 20  compression-ratio  205 non-null    float64
 21  horsepower         203 non-null    object 
 22  peak-rpm           203 non-null    object 
 23  city-mpg           205 non-null    int64  
 24  highway-mpg        205 non-null    int64  
 25  price              201 non-null    object 
dtypes: float64(5), int64(5), object(16)
memory usage: 41.8+ KB
First 3 rows of the dataset:
   symboling normalized-losses         make fuel-type aspiration num-
of-doors  \
0          3                NaN  alfa-romero       gas        std
two
1          3                NaN  alfa-romero       gas        std
```

```
two
2          1               NaN  alfa-romero       gas        std
two

    body-style drive-wheels engine-location  wheel-base  ...  engine-
size  \
0  convertible          rwd           front        88.6  ...
130
1  convertible          rwd           front        88.6  ...
130
2    hatchback          rwd           front        94.5  ...
152

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm
city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000
21
1         mpfi  3.47    2.68               9.0        111      5000
21
2         mpfi  2.68    3.47               9.0        154      5000
19

  highway-mpg  price
0          27  13495
1          27  16500
2          26  16500

[3 rows x 26 columns]
Last 3 rows of the dataset:
     symboling normalized-losses   make fuel-type aspiration num-of-
doors  \
202         -1                95  volvo       gas        std
four
203         -1                95  volvo    diesel      turbo
four
204         -1                95  volvo       gas      turbo
four

    body-style drive-wheels engine-location  wheel-base  ...  engine-
size  \
202      sedan          rwd           front       109.1  ...
173
203      sedan          rwd           front       109.1  ...
145
204      sedan          rwd           front       109.1  ...
141

     fuel-system  bore  stroke compression-ratio horsepower  peak-rpm
\
202         mpfi  3.58    2.87               8.8        134      5500
```

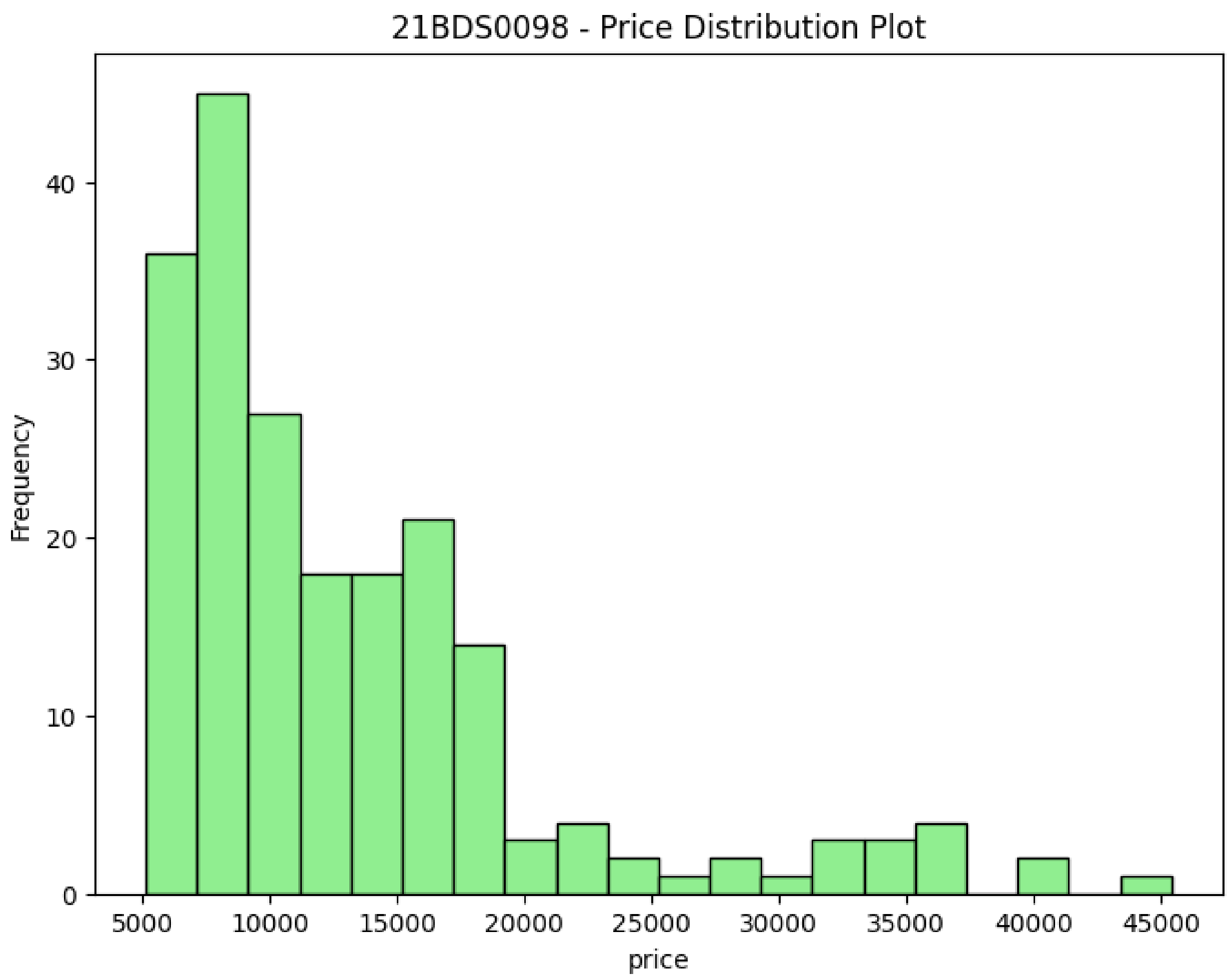
```
203           idi  3.01     3.4                23.0        106      4800

204          mpfi  3.78    3.15                 9.5        114      5400


    city-mpg highway-mpg  price
202       18          23  21485
203       26          27  22470
204       19          25  22625

[3 rows x 26 columns]
Summary Statistics of the dataset:
        symboling      length      height  curb-weight  engine-size  \
count  205.000000  205.000000  205.000000   205.000000   205.000000
mean     0.834146  174.049268   53.724878  2555.565854   126.907317
std      1.245307   12.337289    2.443522   520.680204    41.642693
min     -2.000000  141.100000   47.800000  1488.000000    61.000000
25%      0.000000  166.300000   52.000000  2145.000000    97.000000
50%      1.000000  173.200000   54.100000  2414.000000   120.000000
75%      2.000000  183.100000   55.500000  2935.000000   141.000000
max      3.000000  208.100000   59.800000  4066.000000   326.000000

       compression-ratio    city-mpg  highway-mpg
count         205.000000  205.000000   205.000000
mean           10.142537   25.219512    30.751220
std             3.972040    6.542142     6.886443
min             7.000000   13.000000    16.000000
25%             8.600000   19.000000    25.000000
50%             9.000000   24.000000    30.000000
75%             9.400000   30.000000    34.000000
max            23.000000   49.000000    54.000000
Non-numeric values in 'price': 4
Non-numeric values in 'horsepower': 2
Non-numeric values in 'normalized-losses': 41
Mean of height: 53.72487804878049
Median of height: 54.1
Mode of height: 50.8
Standard Deviation of height: 2.4435219699049044
Variance of height: 5.970799617407946
Quartiles of height:
0.25    52.0
0.50    54.1
0.75    55.5
Name: height, dtype: float64
Interquartile Range (IQR) of height: 3.5
Correlation between price and horsepower: 0.7587142204139695

<ipython-input-73-8282478dba2e>:59: FutureWarning:
```

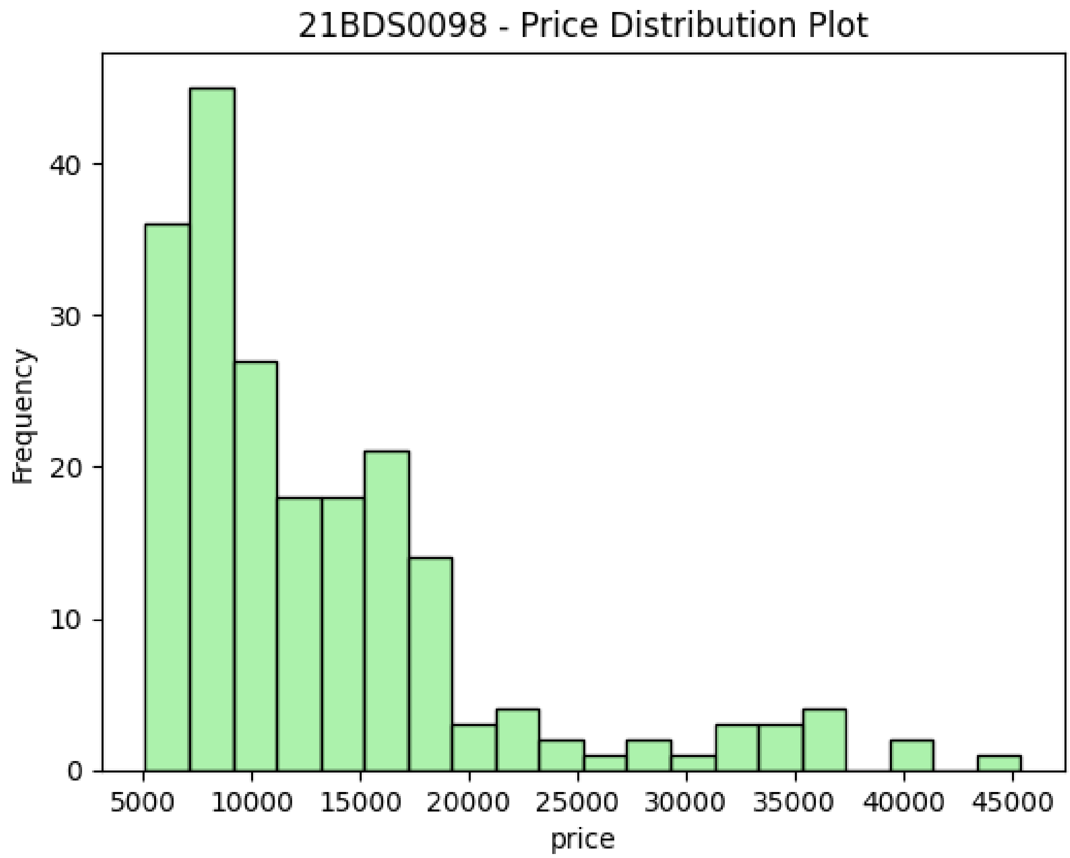
```
A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.
```



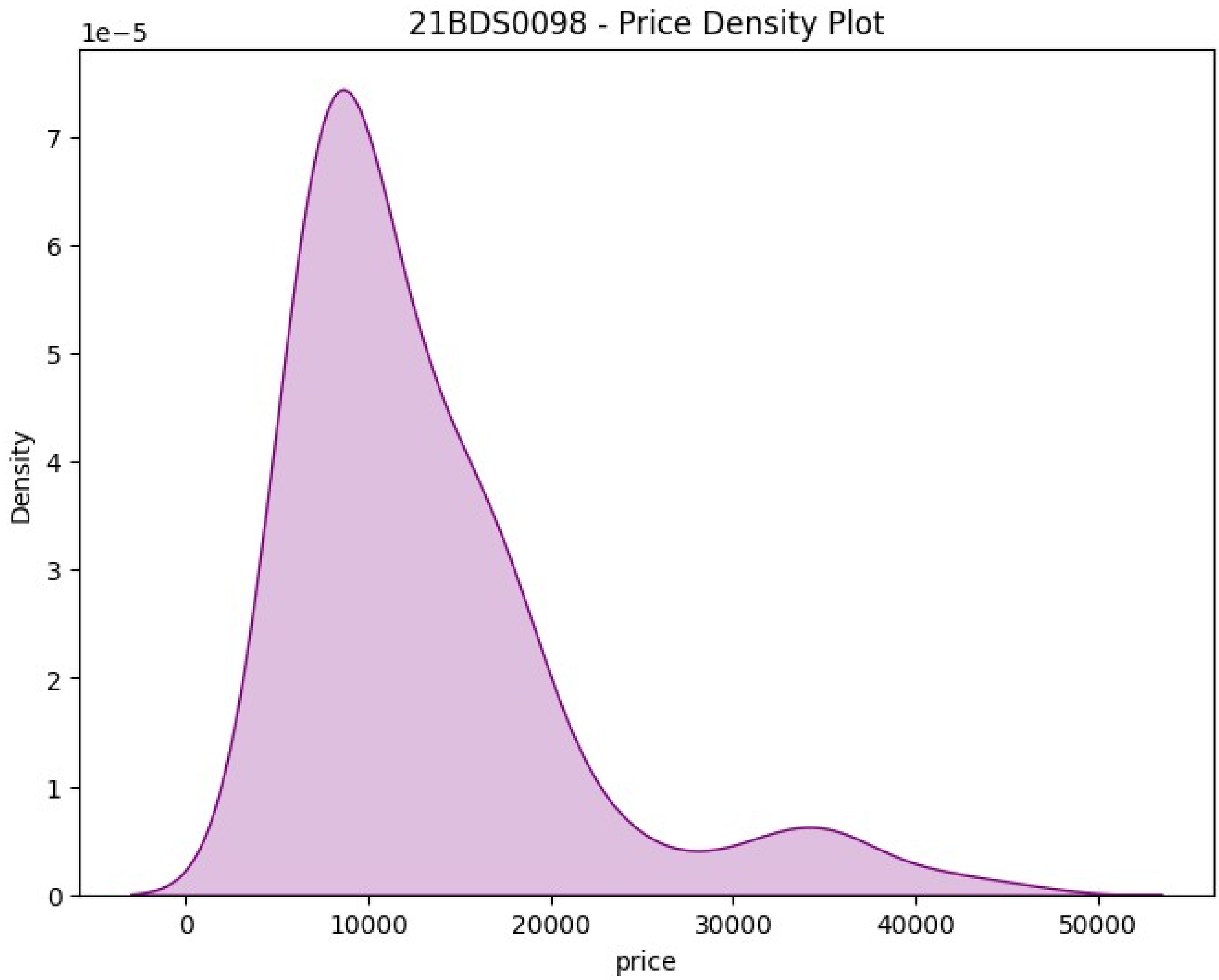


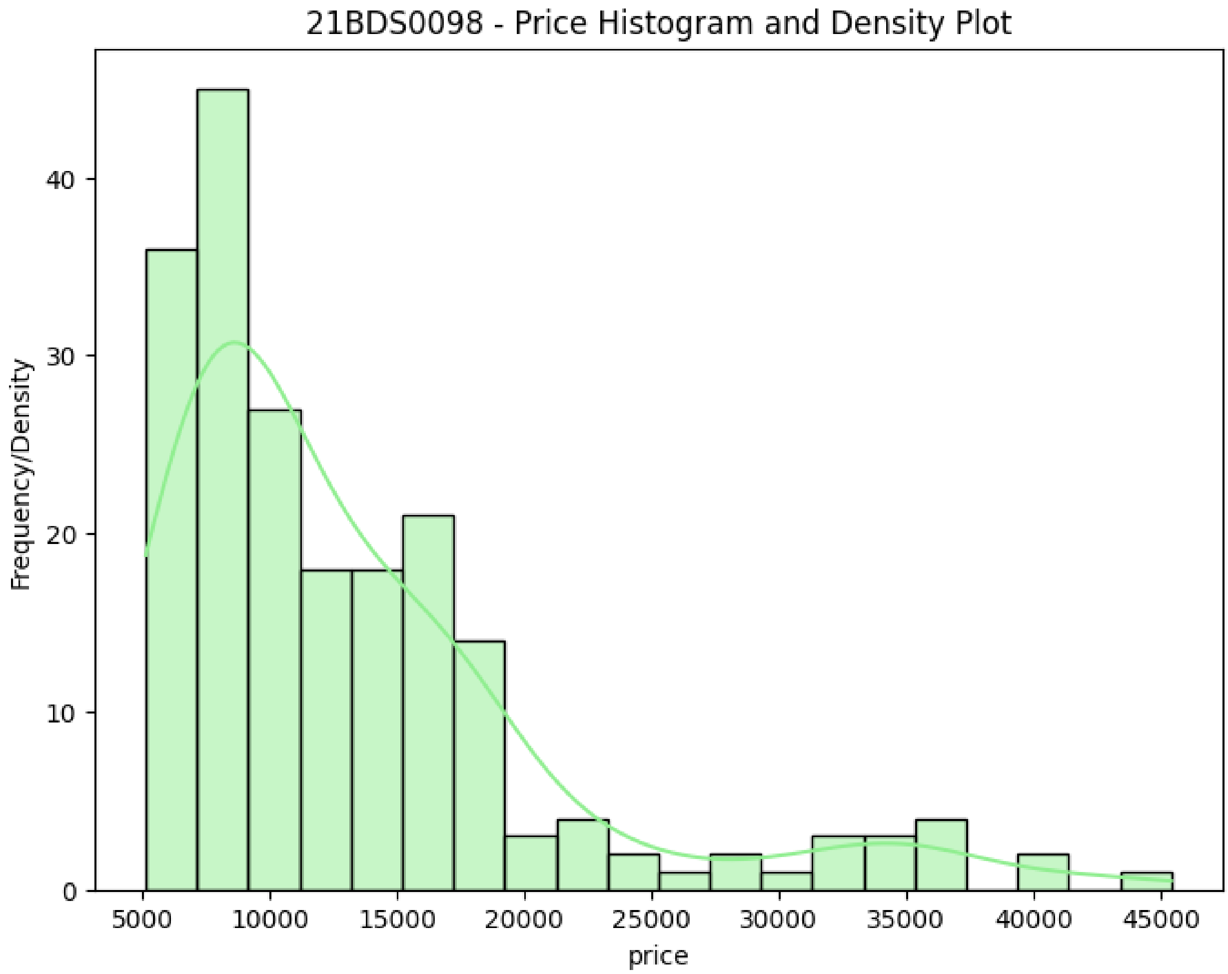
21BDS0098 - Height Distribution Plot
Frequency
Height
0
5
10
15
20
48
50
52
54
56
58
60


21BDS0098 - Price Distribution Plot
Frequency
0
10
20
30
40
5000
10000
15000
20000
25000
30000
35000
40000
45000
price

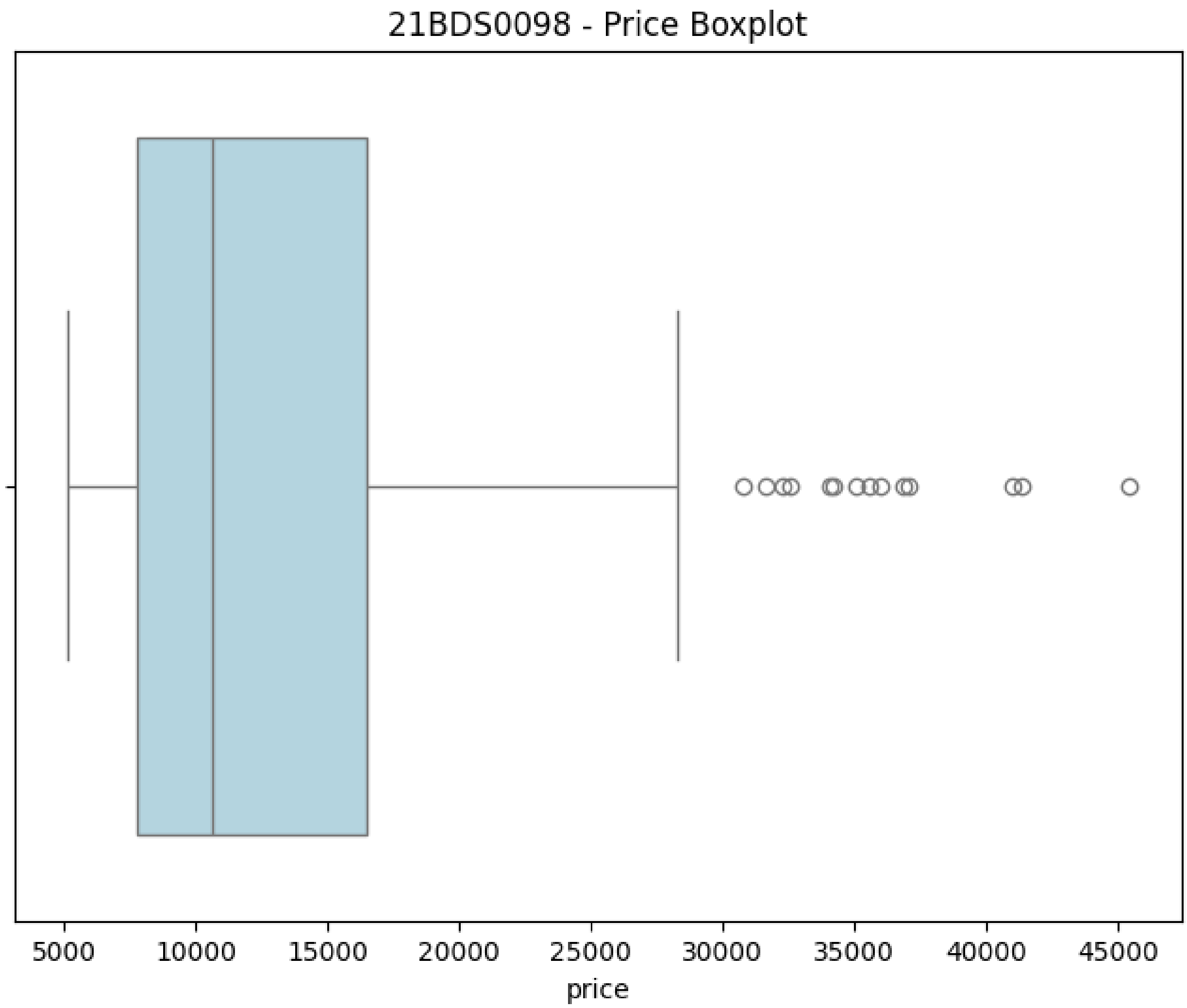



```
<ipython-input-73-8282478dba2e>:124: FutureWarning:



`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.
```

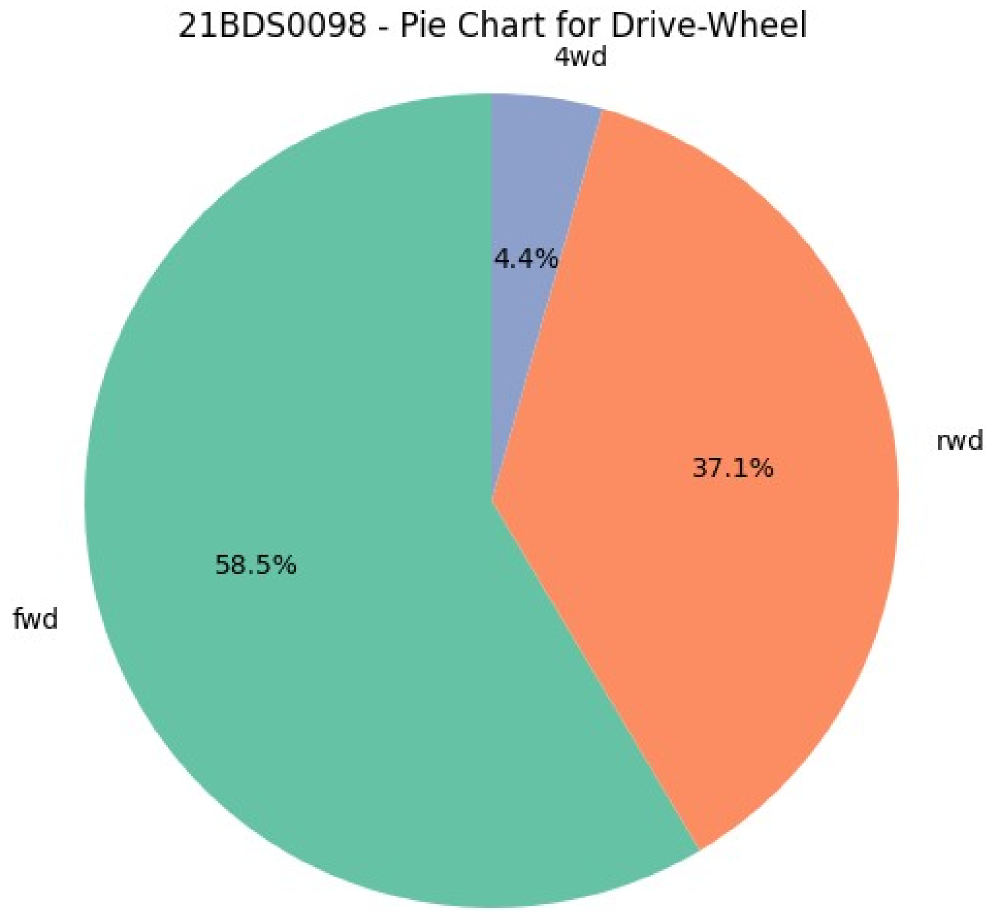

21BDS0098 - Price Density Plot
1e−5
Density
0
1
2
3
4
5
6
7
0
10000
20000
30000
40000
50000
price

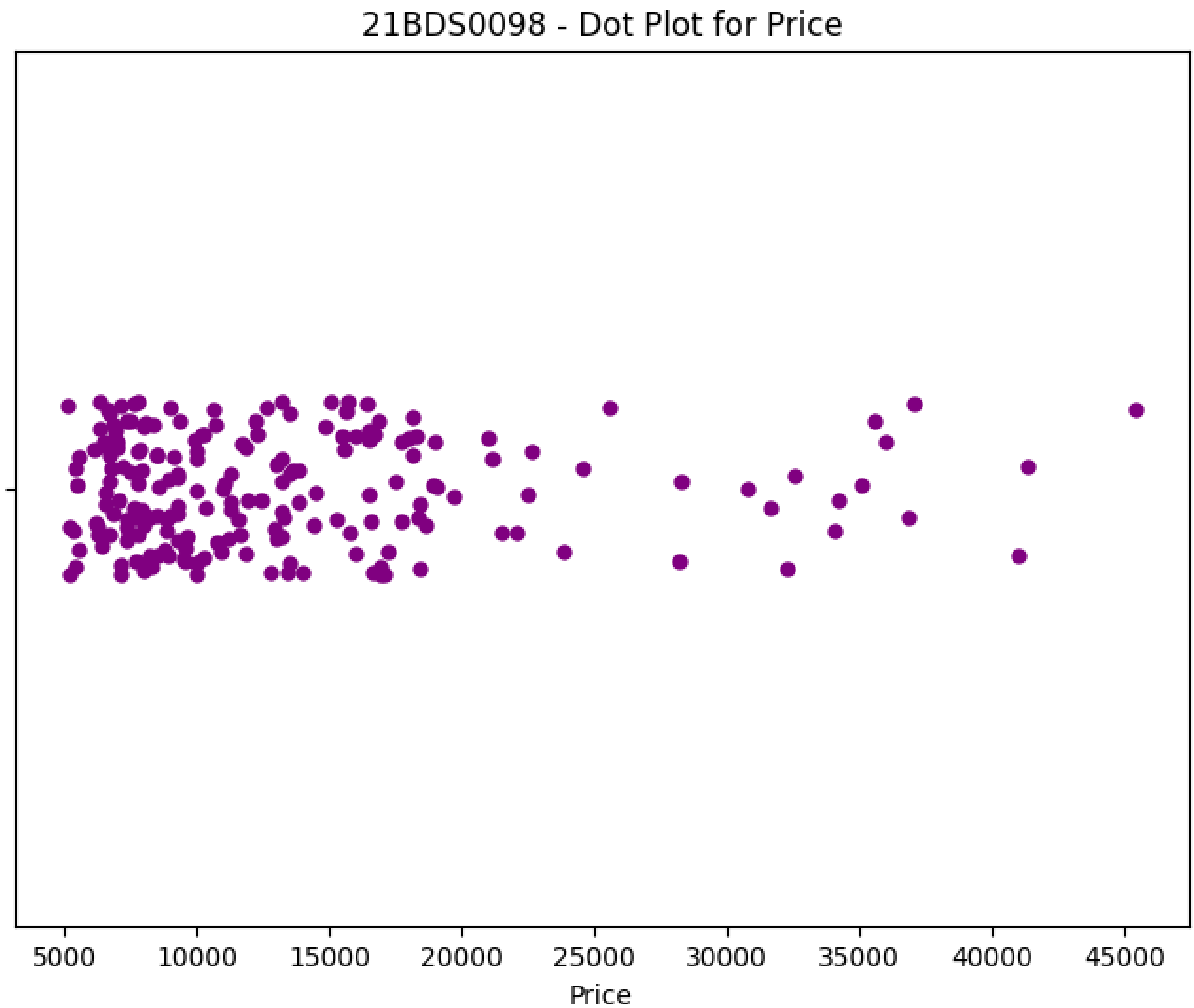


21BDS0098 - Price Histogram and Density Plot
Frequency/Density
0
10
20
30
40
5000
10000
15000
20000
25000
30000
35000
40000
45000
price




```
'no-of-cylinders' column not found in the dataset.
Available columns: Index(['symboling', 'normalized-losses', 'make',
'fuel-type', 'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-
location',
       'length', 'height', 'curb-weight', 'engine-type', 'num-of-
cylinders',
       'engine-size', 'stroke', 'compression-ratio', 'horsepower',
'peak-rpm',
       'city-mpg', 'highway-mpg', 'price'],
      dtype='object')
```


21BDS0098 - Pie Chart for Drive-Wheel
4wd
4.4%
rwd
37.1%
fwd
58.5%



MODULE 3

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset (replace 'data.csv' with the actual path to your
CSV file)
HarshArya = pd.read_csv('data(1).csv')

# Step 1: Replace '?' with NaN (if applicable)
HarshArya.replace('?', np.nan, inplace=True)

# Step 2: Convert columns that should be numeric to numeric, coercing
errors to NaN
```

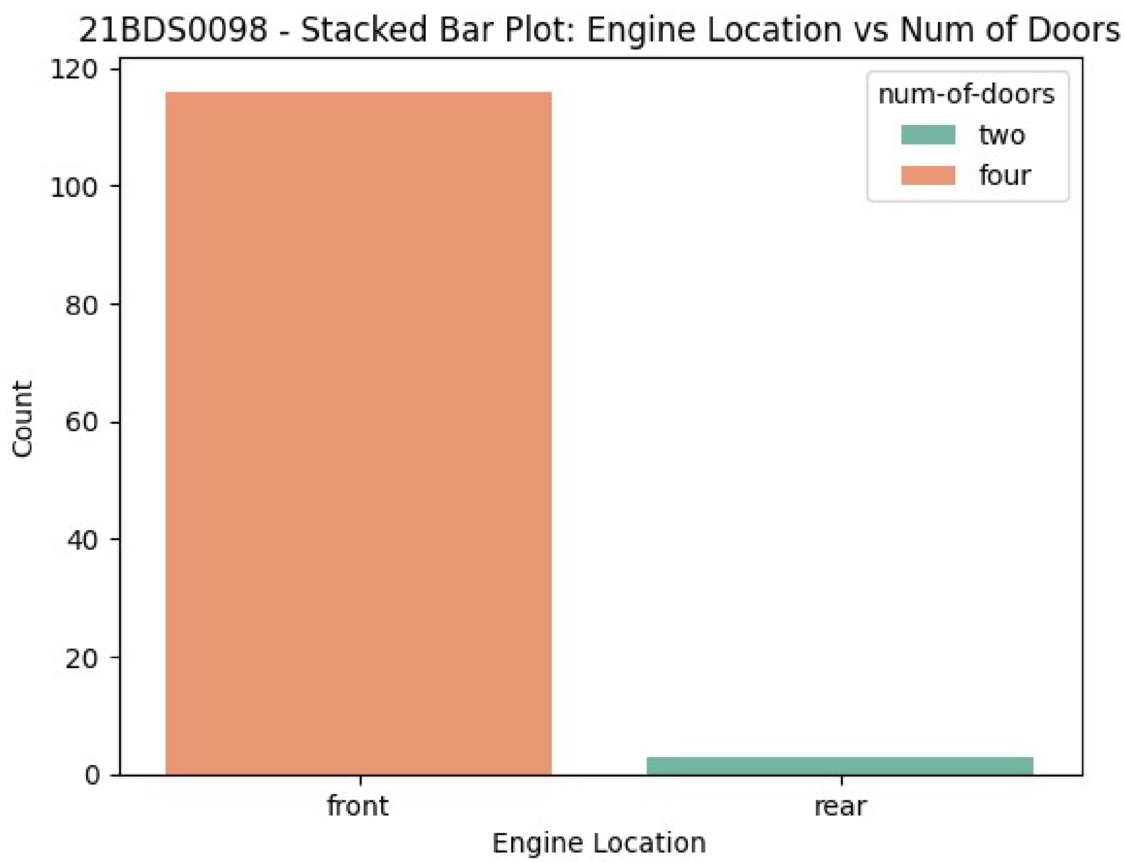
```
# Example: Convert 'horsepower' and 'price' columns to numeric
HarshArya['horsepower'] = pd.to_numeric(HarshArya['horsepower'],
errors='coerce')
HarshArya['price'] = pd.to_numeric(HarshArya['price'],
errors='coerce')

# Step 3: Handle missing values after conversion to numeric (impute
missing values)
# Fill missing values for numeric columns with the mean
numeric_cols_0098 = HarshArya.select_dtypes(include=['float64',
'int64']).columns
HarshArya[numeric_cols_0098] =
HarshArya[numeric_cols_0098].fillna(HarshArya[numeric_cols_0098].mean(
))

# Step 4: Handle missing values in non-numeric columns by using the
mode (most frequent value)
non_numeric_cols_0098 = HarshArya.select_dtypes(exclude=['float64',
'int64']).columns
for col in non_numeric_cols_0098:
    HarshArya[col] = HarshArya[col].fillna(HarshArya[col].mode()[0])

# Step 5: Check for remaining missing values
print("Missing values after imputation:\n", HarshArya.isna().sum())

# Bivariate and Multivariate Analysis

# 1. Categorical vs Categorical: Stacked Bar Plot (engine-location vs
num-of-doors)
sns.countplot(data=HarshArya, x="engine-location", hue='num-of-doors',
dodge=False, palette='Set2')
plt.title("21BDS0098 - Stacked Bar Plot: Engine Location vs Num of
Doors")
plt.xlabel('Engine Location')
plt.ylabel('Count')
plt.show()

# 2. Categorical vs Quantitative: Bar Plot (Price vs Engine Location)
sns.barplot(data=HarshArya, x='engine-location', y='price',
palette='Set2')
plt.title("21BDS0098 - Bar Chart: Price vs Engine Location")
plt.xlabel('Engine Location')
plt.ylabel('Price')
plt.show()

# 3. Quantitative vs Quantitative: Scatter Plot (Price vs Horsepower)
sns.scatterplot(data=HarshArya, x='price', y='horsepower',
hue='engine-location', palette='Set2')
plt.title("21BDS0098 - Scatter Plot: Price vs Horsepower")
plt.xlabel('Price')
```

```
plt.ylabel('Horsepower')
plt.show()

# 4. Quantitative vs Quantitative: Heatmap (Correlation matrix)
numeric_data_0098 = HarshArya.select_dtypes(include=['number'])
corr_matrix_0098 = numeric_data_0098.corr()   # Calculate correlation
matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix_0098, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)
plt.title("21BDS0098 - Heatmap of Correlation Matrix")
plt.show()

# 5. Categorical vs Quantitative: Density Plot (Price vs Engine
Location)
sns.kdeplot(data=HarshArya, x='price', hue='engine-location',
fill=True, palette='Set2')
plt.title("21BDS0098 - Density Plot: Price vs Engine Location")
plt.xlabel('Price')
plt.ylabel('Density')
plt.show()

# 6. Categorical vs Quantitative: Box Plot (Price vs Engine Location)
sns.boxplot(data=HarshArya, x='engine-location', y='price',
palette='Set2')
plt.title("21BDS0098 - Box Plot: Price vs Engine Location")
plt.xlabel('Engine Location')
plt.ylabel('Price')
plt.show()

# 7. Categorical vs Quantitative: Violin Plot (Price vs Engine
Location)
sns.violinplot(data=HarshArya, x='engine-location', y='price',
palette='Set2')
plt.title("21BDS0098 - Violin Plot: Price vs Engine Location")
plt.xlabel('Engine Location')
plt.ylabel('Price')
plt.show()

# 8. Multivariate: Scatter Plot (using color as third variable)
sns.scatterplot(data=HarshArya, x='price', y='horsepower',
hue='engine-location', size='curb-weight', palette='Set2', sizes=(20,
200))
plt.title("21BDS0098 - Scatter Plot: Price vs Horsepower (With Size
and Color)")
plt.xlabel('Price')
plt.ylabel('Horsepower')
plt.show()

# 9. Bubble Plot (with x, y, and size)
```

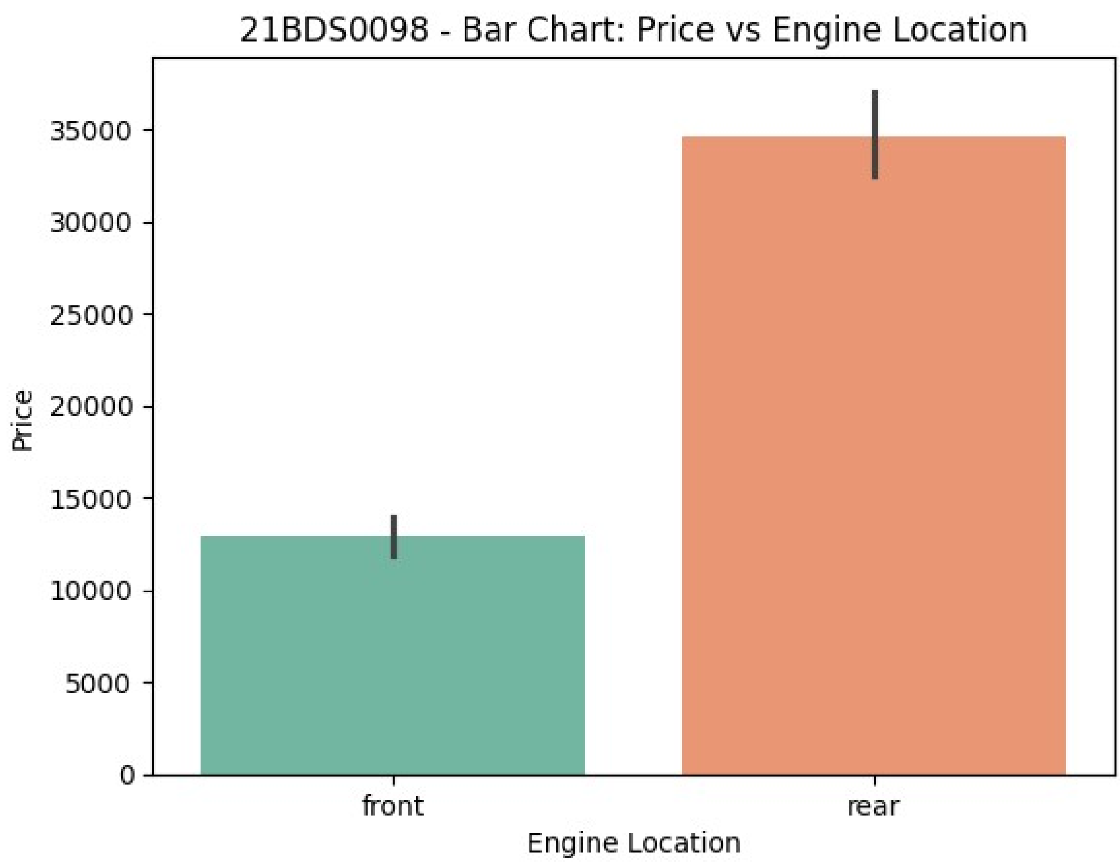
```python
sns.scatterplot(data=HarshArya, x='price', y='horsepower',
hue='engine-location', size='curb-weight', sizes=(20, 200),
palette='Set2')
plt.title("21BDS0098 - Bubble Plot: Price vs Horsepower")
plt.xlabel('Price')
plt.ylabel('Horsepower')
plt.show()

# 10. Display a graph into sub-graphs (Faceting)
sns.displot(data=HarshArya, x='price', col='engine-location',
kde=True, facet_kws={'margin_titles': True})
plt.suptitle("21BDS0098 - Distribution of Price by Engine Location")
plt.show()

# 11. Display a graph into sub-graphs (Facet Grid)
sns.FacetGrid(HarshArya, col='engine-location').map(sns.histplot,
'price', kde=True)
plt.suptitle("21BDS0098 - Facet Grid: Price Distribution by Engine
Location")
plt.show()

# Bivariate Analysis - Contingency Table (Categorical vs Categorical)

# Create a contingency table for "engine-location" and "num-of-doors"
contingency_table = pd.crosstab(HarshArya['engine-
location'],HarshArya['num-of-doors'])
print("Contingency Table for Engine Location vs Num of Doors:")
print(contingency_table)

# Stacked bar chart for engine-location vs num-of-doors
# 1. Using Matplotlib
contingency_table.plot(kind='bar', stacked=True, color=['skyblue',
'lightgreen'])
plt.title("21BDS0098 - Stacked Bar Chart: Engine Location vs Num of
Doors")
plt.xlabel('Engine Location')
plt.ylabel('Count')
plt.show()

# 2. Using Seaborn
sns.barplot(data=HarshArya, x='engine-location', hue='num-of-doors',
dodge=False, palette='Set2')
plt.title("21BDS0098 - Stacked Bar Plot: Engine Location vs Num of
Doors")
plt.xlabel('Engine Location')
plt.ylabel('Count')
plt.show()

# 3. Grouped bar plot (side-by-side plot)
sns.barplot(data=HarshArya, x='engine-location', hue='num-of-doors',
```

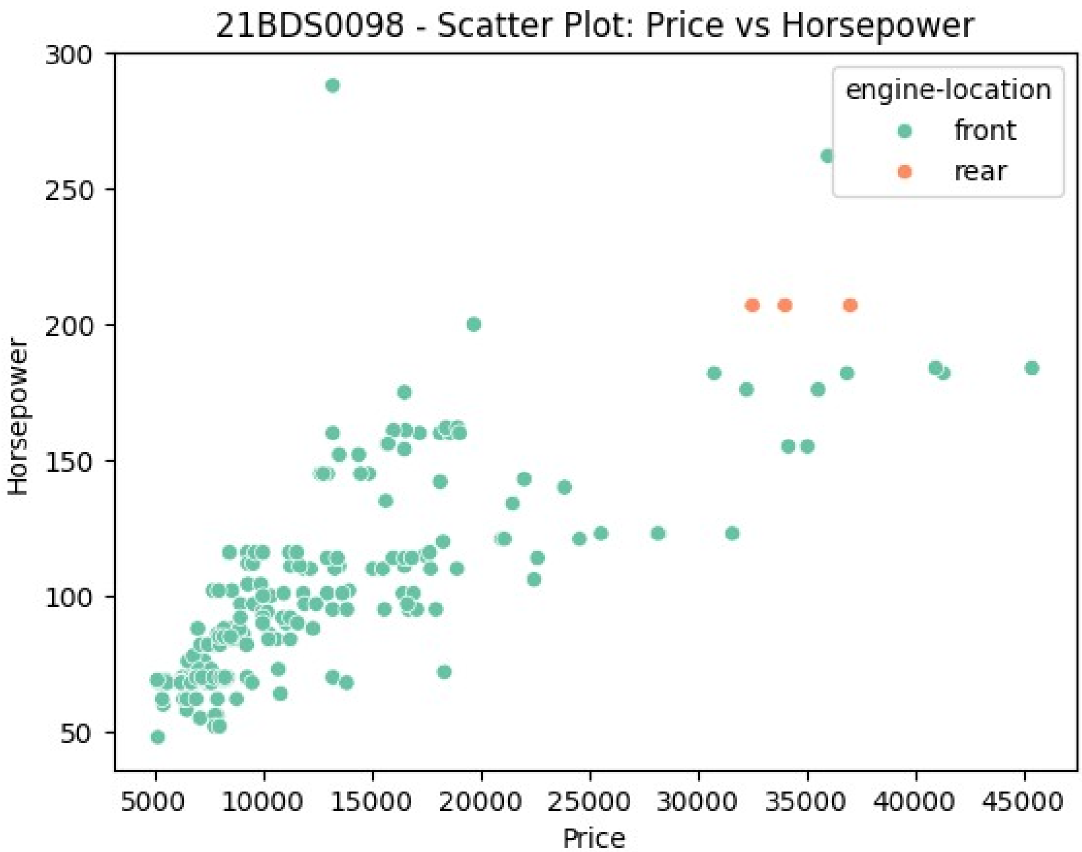
```
palette='Set2', ci=None)
plt.title("21BDS0098 - Grouped Bar Plot: Engine Location vs Num of
Doors")
plt.xlabel('Engine Location')
plt.ylabel('Count')
plt.show()
```

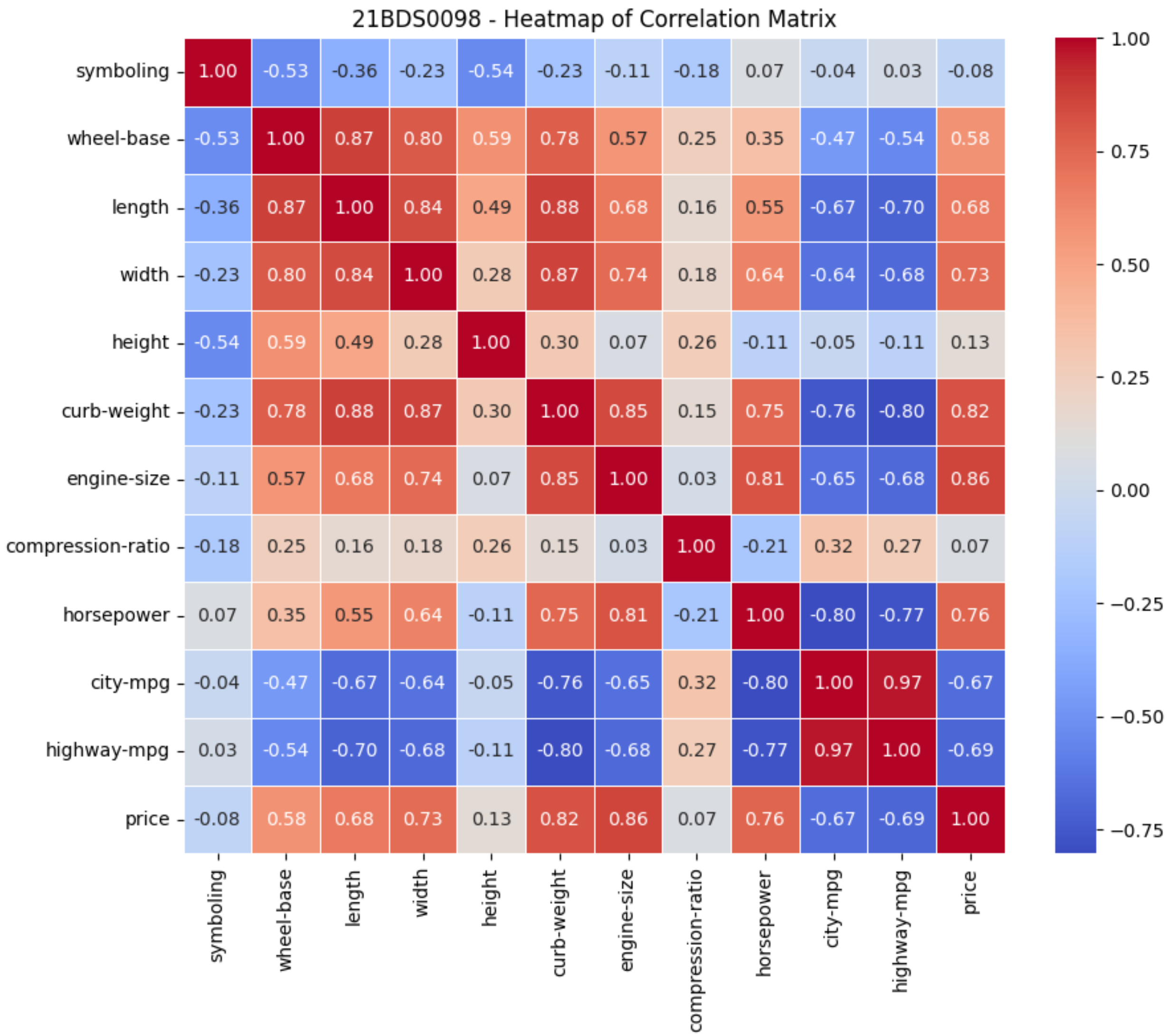
```
Missing values after imputation:
 symboling            0
normalized-losses    0
make                 0
fuel-type            0
aspiration           0
num-of-doors         0
body-style           0
drive-wheels         0
engine-location      0
wheel-base           0
length               0
width                0
height               0
curb-weight          0
engine-type          0
num-of-cylinders     0
engine-size          0
fuel-system          0
bore                 0
stroke               0
compression-ratio    0
horsepower           0
peak-rpm             0
city-mpg             0
highway-mpg          0
price                0
dtype: int64
```

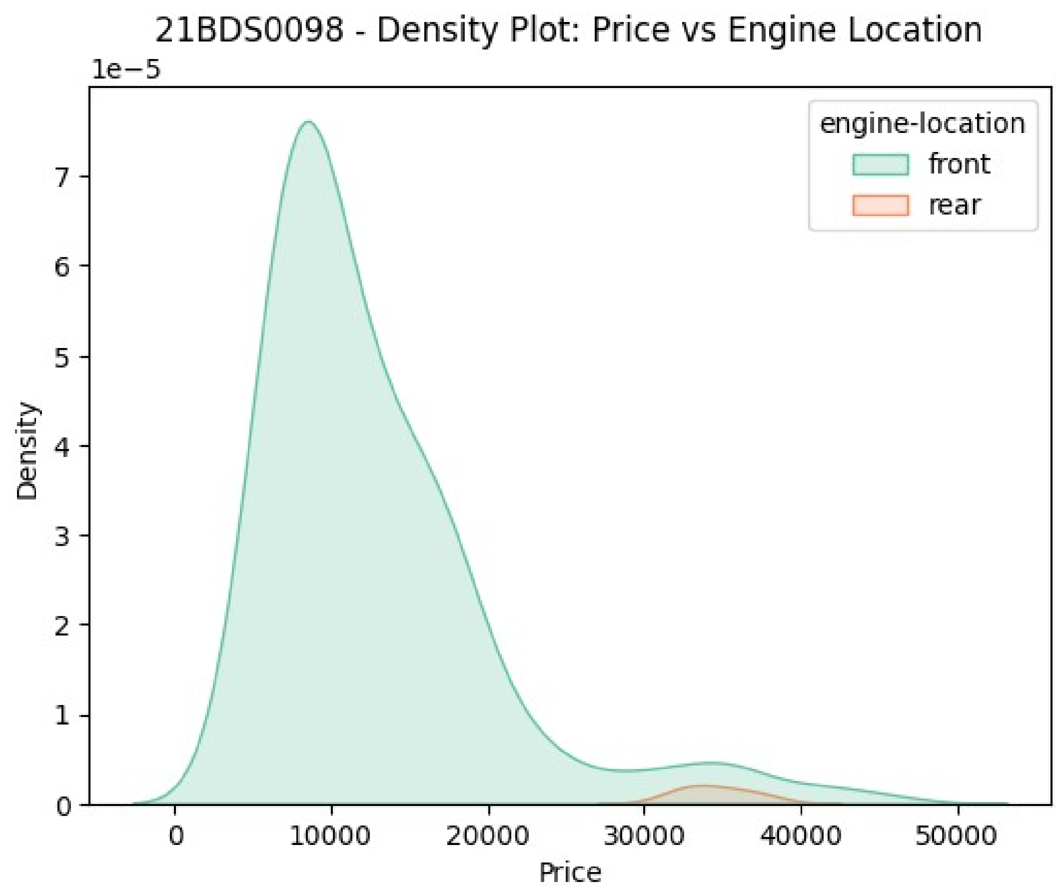



```
<ipython-input-79-daa2eab70c16>:41: FutureWarning:



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

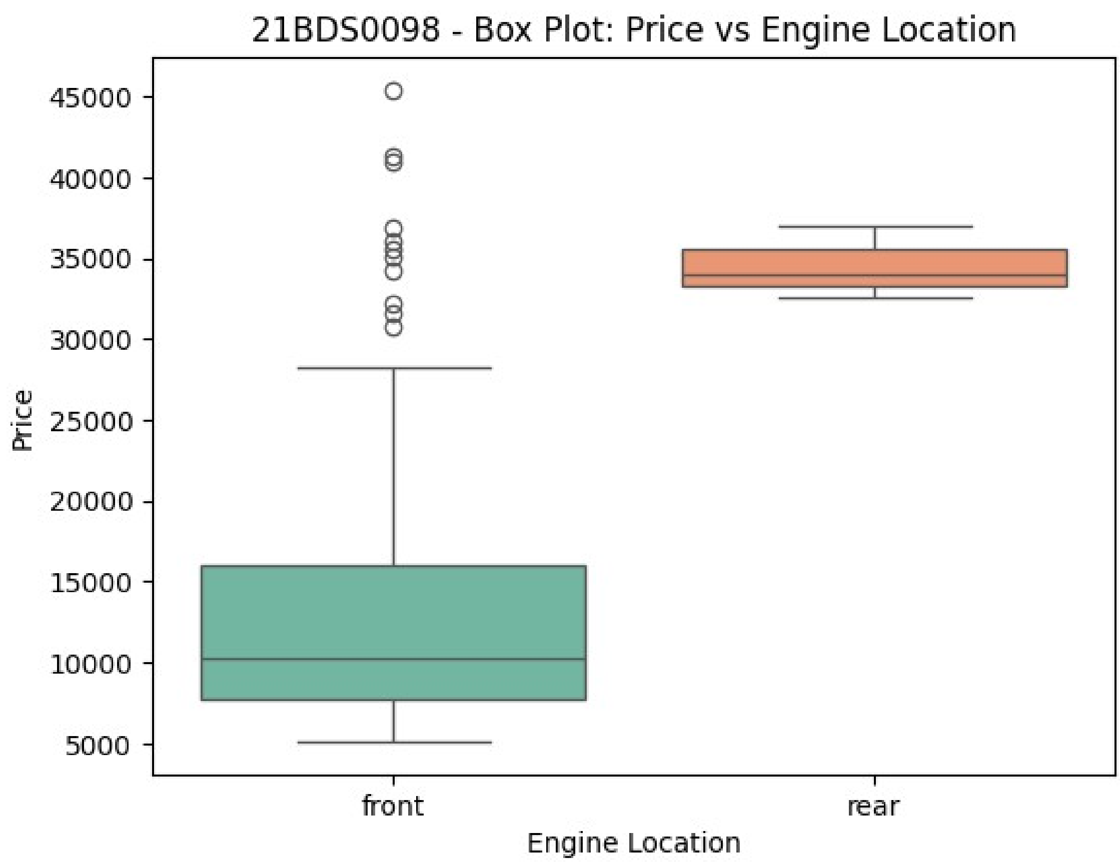

21BDS0098 - Bar Chart: Price vs Engine Location
Price
35000
30000
25000
20000
15000
10000
5000
0
front
rear
Engine Location


21BDS0098 - Scatter Plot: Price vs Horsepower
engine-location
front
rear
Horsepower
Price
300
250
200
150
100
50
5000
10000
15000
20000
25000
30000
35000
40000
45000

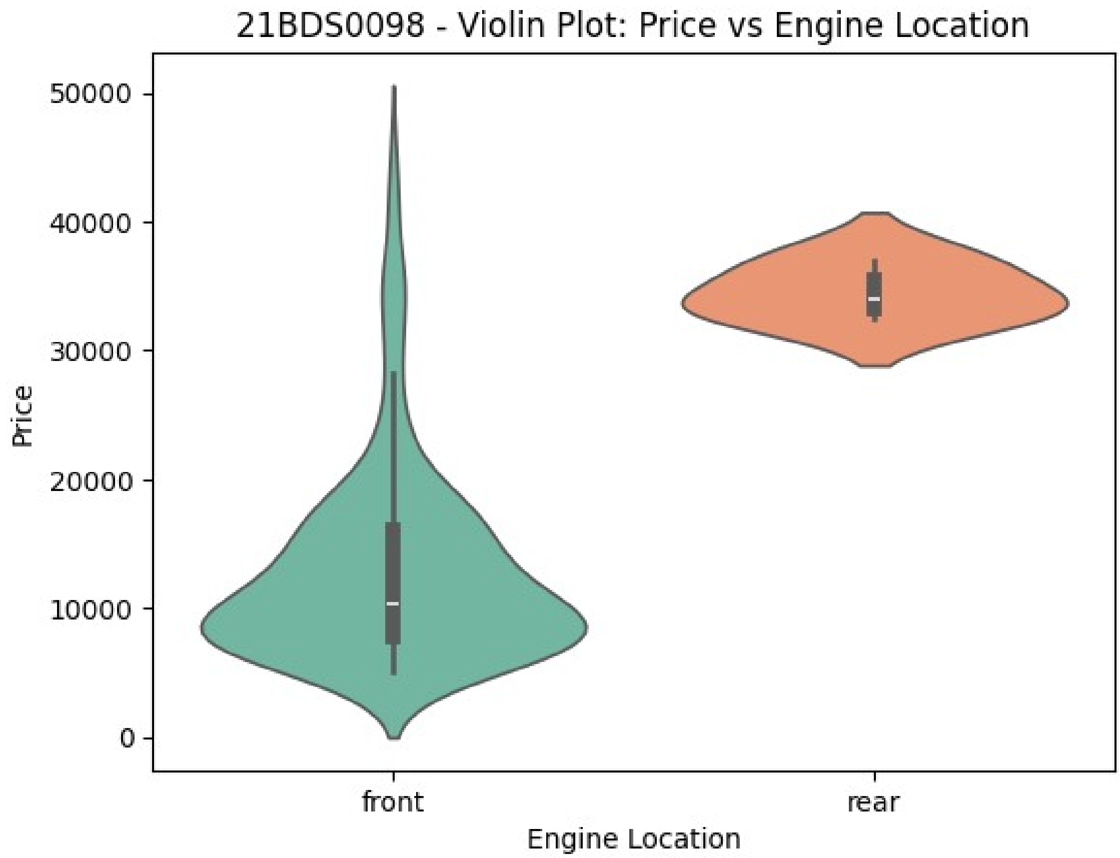

21BDS0098 - Heatmap of Correlation Matrix
symboling
wheel-base
length
width
height
curb-weight
engine-size
compression-ratio
horsepower
city-mpg
highway-mpg
price
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

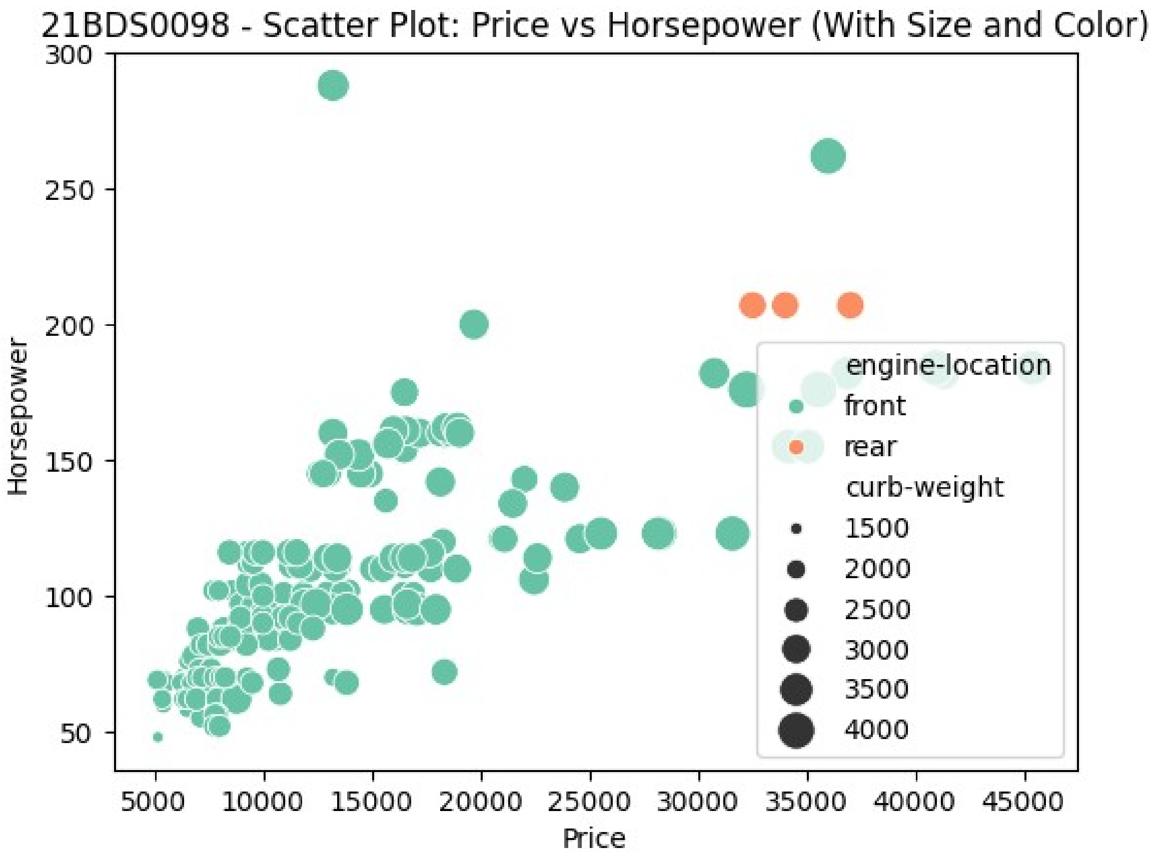


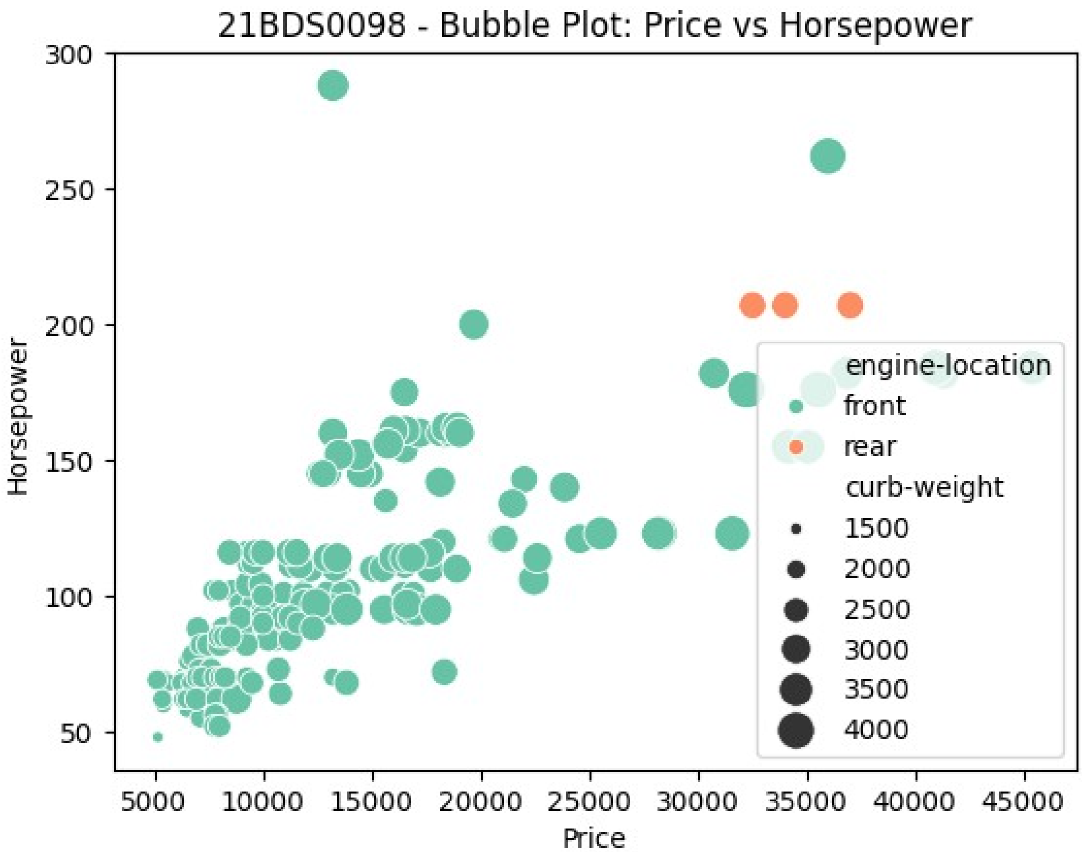

```
<ipython-input-79-daa2eab70c16>:70: FutureWarning:



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

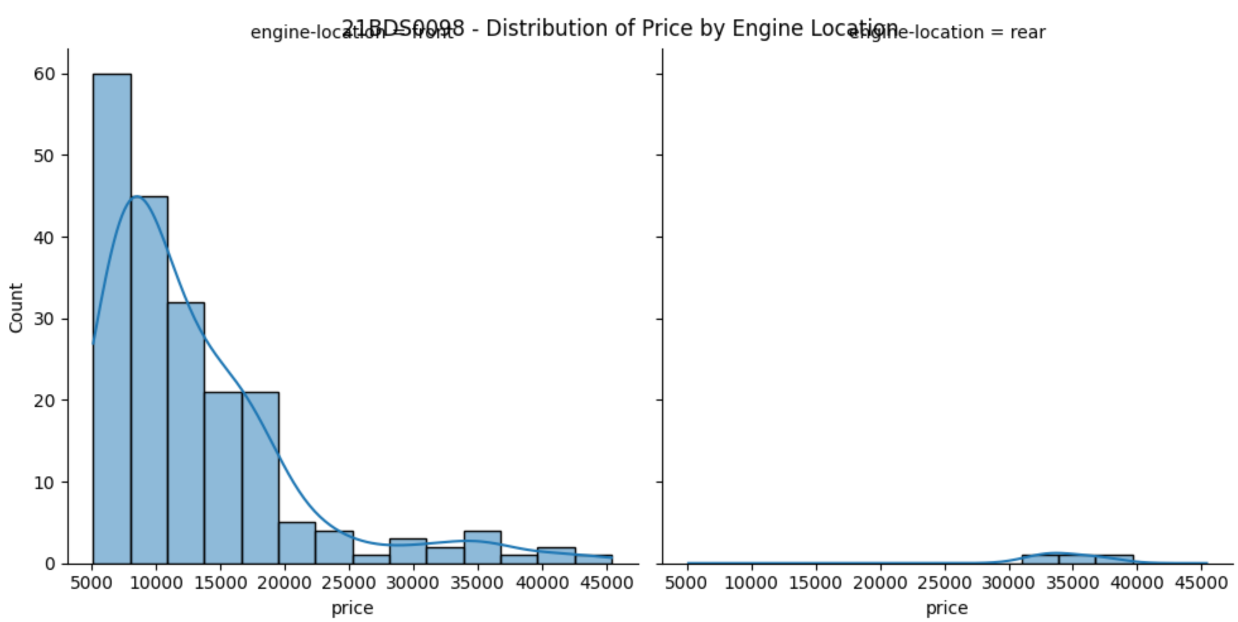


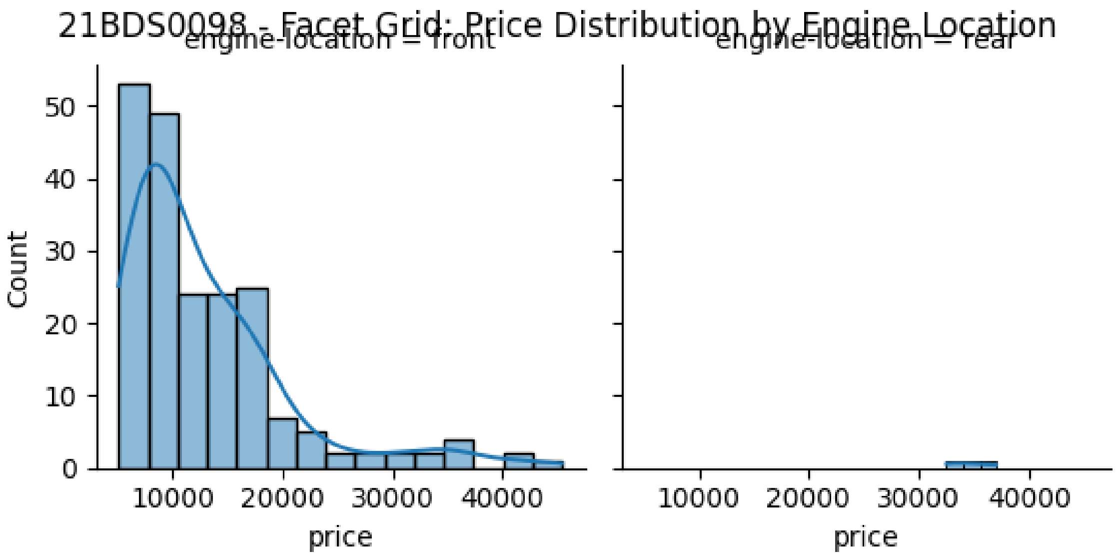

```
<ipython-input-79-daa2eab70c16>:77: FutureWarning:


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


21BDS0098 - Violin Plot: Price vs Engine Location
50000
40000
30000
20000
10000
0
Price
front
rear
Engine Location

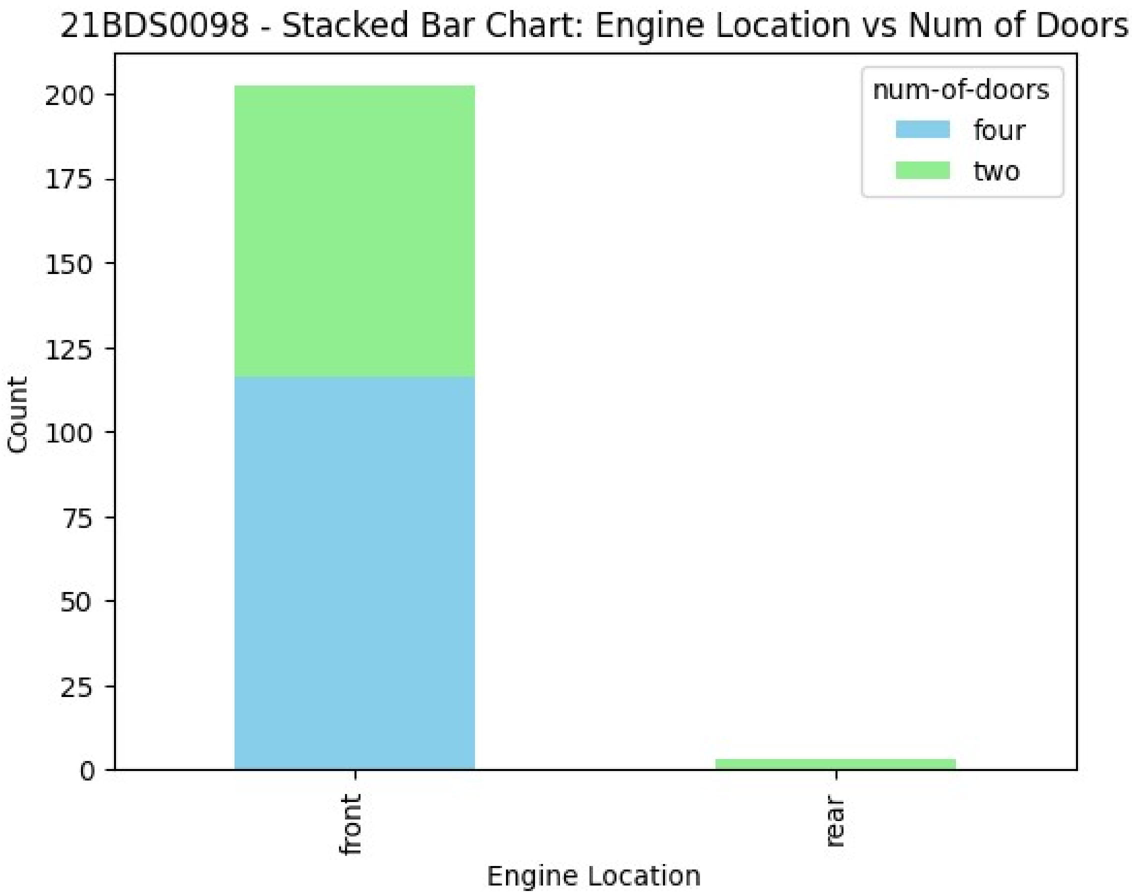

21BDS0098 - Scatter Plot: Price vs Horsepower (With Size and Color)
Horsepower
Price
300
250
200
150
100
50
5000
10000
15000
20000
25000
30000
35000
40000
45000
engine-location
front
rear
curb-weight
1500
2000
2500
3000
3500
4000

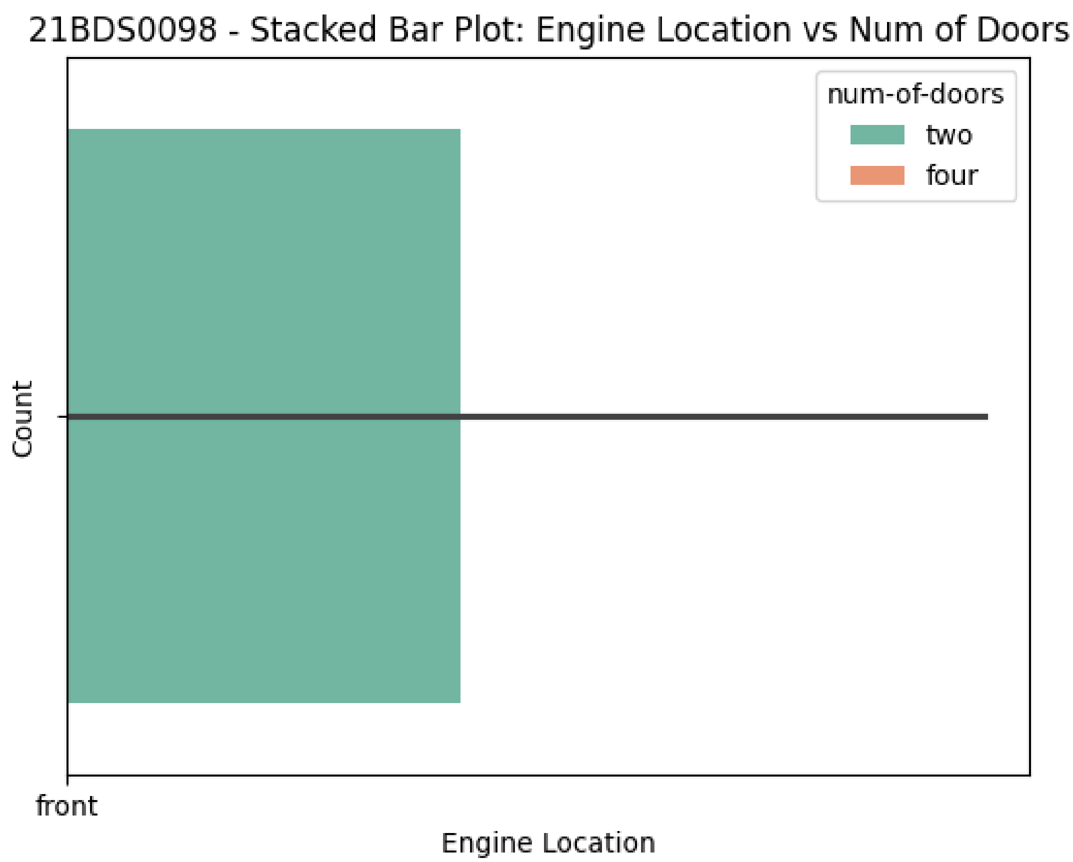

21BDS0098 - Bubble Plot: Price vs Horsepower
Horsepower
Price
engine-location
front
rear
curb-weight
1500
2000
2500
3000
3500
4000
50
100
150
200
250
300
5000
10000
15000
20000
25000
30000
35000
40000
45000


21BDS0098 - Distribution of Price by Engine Location
engine-location = front
engine-location = rear
Count
price
price
0
10
20
30
40
50
60
5000
10000
15000
20000
25000
30000
35000
40000
45000

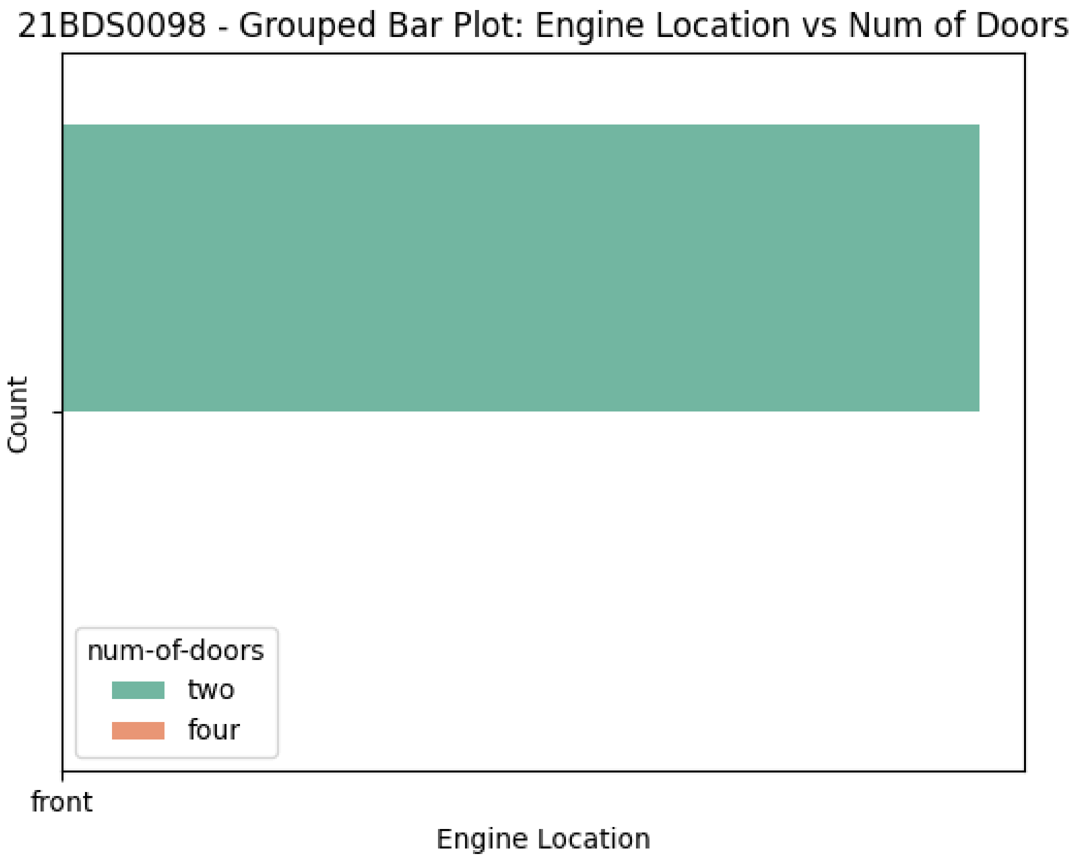



```
Contingency Table for Engine Location vs Num of Doors:
num-of-doors     four  two
engine-location
front             116   86
rear                0    3
```


21BDS0098 - Stacked Bar Chart: Engine Location vs Num of Doors
num-of-doors
four
two
Count
0
25
50
75
100
125
150
175
200
front
rear
Engine Location




```
<ipython-input-79-daa2eab70c16>:130: FutureWarning:



The `ci` parameter is deprecated. Use `errorbar=None` for the same
effect.
```




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

# 1. Simulate the AirPassengers dataset
# Let's create the dataset with monthly passenger data from 1949 to
1960

date_range = pd.date_range(start='1949-01-01', end='1960-12-01',
freq='MS')
airpassengers_data = [
    112, 118, 132, 129, 121, 135, 148, 148, 136, 119, 104, 118,  #
1949
    115, 126, 141, 135, 125, 149, 170, 170, 158, 133, 114, 140,  #
1950
    135, 148, 178, 163, 158, 182, 209, 208, 191, 164, 149, 163,  #
1951
    172, 188, 214, 209, 195, 220, 246, 253, 227, 200, 170, 171,  #
1952
    190, 207, 230, 220, 210, 236, 258, 266, 245, 213, 188, 188,  #
1953
```

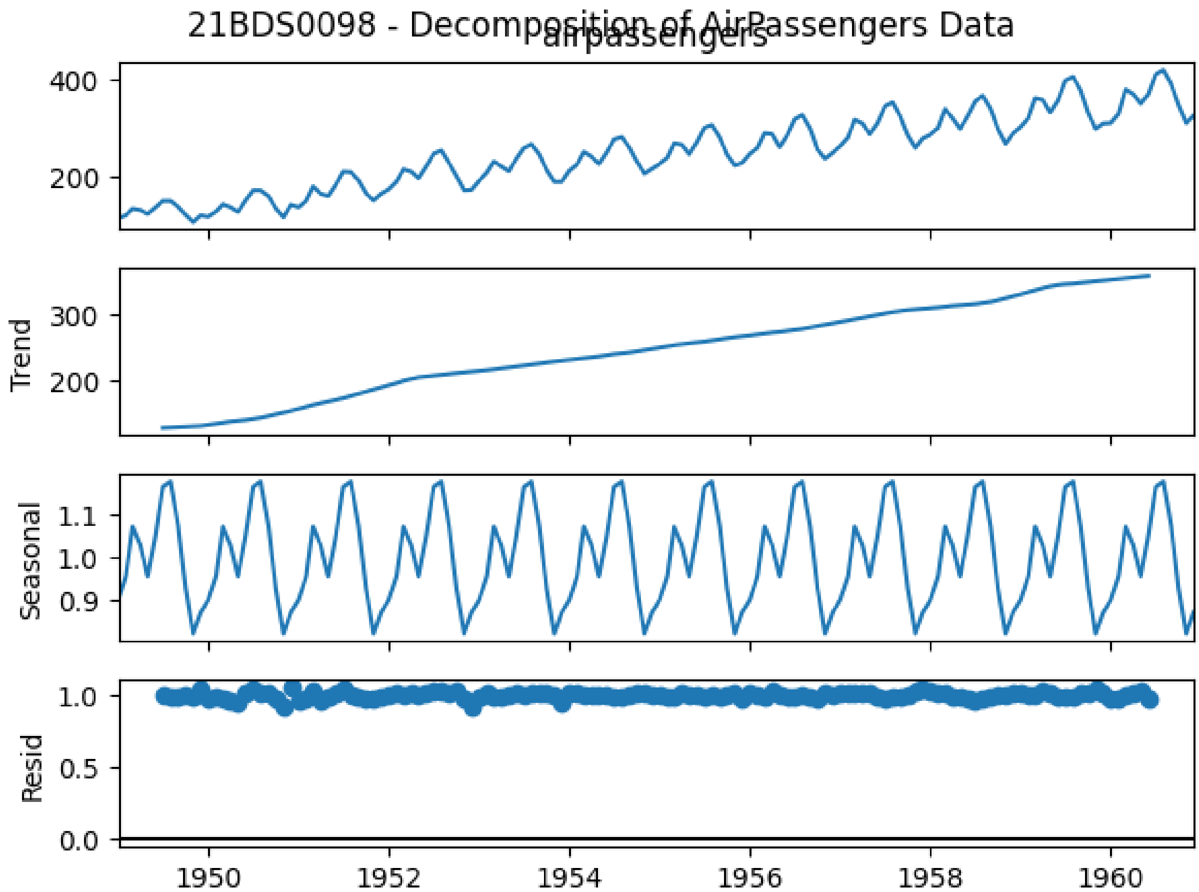
```
    211, 224, 250, 240, 225, 249, 276, 281, 259, 230, 205, 215,  #
1954
    225, 238, 268, 265, 245, 270, 300, 306, 280, 245, 222, 227,  #
1955
    246, 258, 289, 287, 260, 285, 318, 327, 299, 255, 236, 248,  #
1956
    262, 280, 317, 309, 287, 310, 345, 353, 324, 285, 258, 278,  #
1957
    286, 300, 339, 320, 298, 324, 355, 367, 340, 298, 267, 288,  #
1958
    302, 320, 362, 358, 332, 357, 398, 406, 378, 331, 298, 309,  #
1959
    310, 329, 380, 370, 350, 370, 411, 420, 391, 348, 310, 325   #
1960
]

# Create DataFrame with Date and Air Passengers
HarshArya = pd.DataFrame({'airpassengers': airpassengers_data},
index=date_range)

# 2. Check the structure and data type of AirPassengers
print("Structure and Data type of the dataset:")
print(HarshArya.info())  # Structure and data types

# 3. Check for missing values in the dataset
print("\nMissing values in the dataset:")
print(HarshArya.isna().sum())  # Missing values

# 4. Check for the starting date and ending date
print("\nStarting date and Ending date of the dataset:")
print("Start Date: ", HarshArya.index[0])
print("End Date: ", HarshArya.index[-1])

# 5. Check the frequency of the dataset
print("\nFrequency of the dataset:")
print(HarshArya.index.freq)  # Frequency of the dataset

# 6. Check for the summary of the dataset
print("\nSummary of the dataset:")
print(HarshArya.describe())  # Summary statistics

# 7. Plot the decomposition of the dataset
# Decompose the time series using statsmodels
decomposition_0098 =
sm.tsa.seasonal_decompose(HarshArya['airpassengers'],
model='multiplicative', period=12)

# Plot the decomposition
decomposition_0098.plot()
plt.suptitle("21BDS0098 - Decomposition of AirPassengers Data")
```

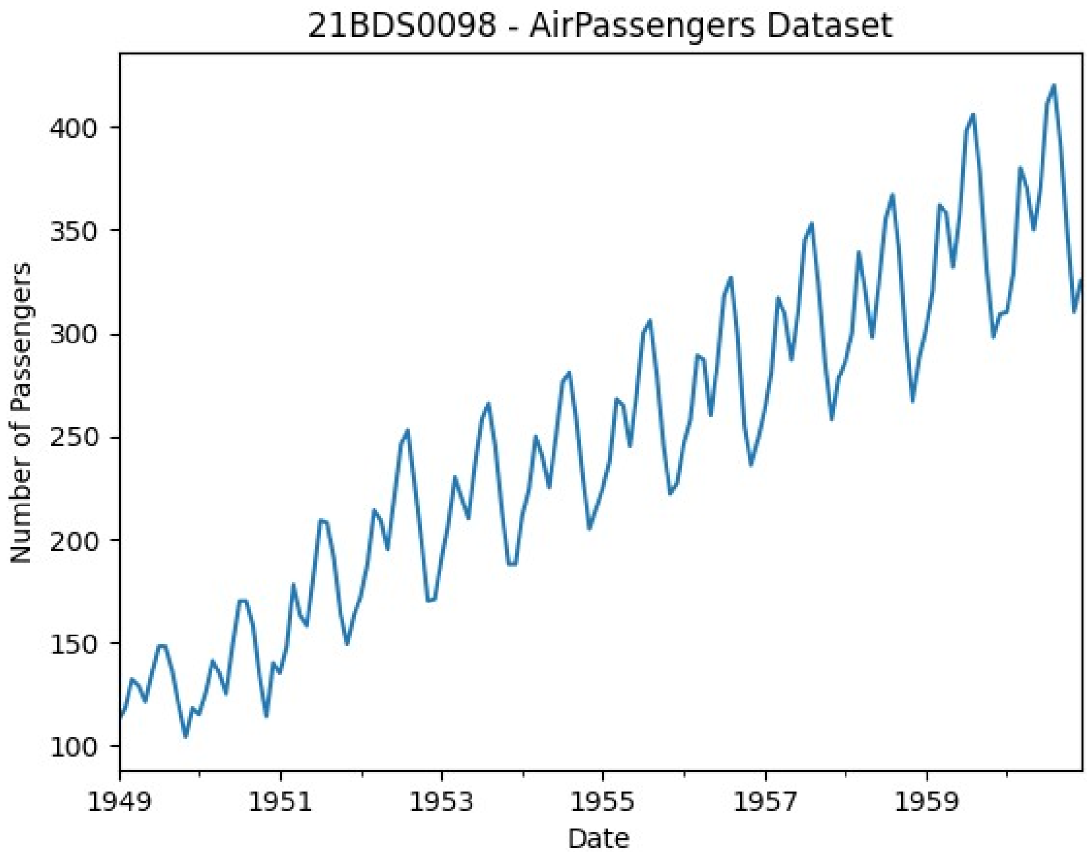
```python
plt.show()

# 8. Plot the dataset
HarshArya['airpassengers'].plot(title="21BDS0098 - AirPassengers
Dataset")
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.show()

# 9. Plot the time-series of the dataset (plot.ts equivalent)
plt.figure(figsize=(10, 6))
plt.plot(HarshArya.index, HarshArya['airpassengers'])
plt.title("21BDS0098 - Time Series Plot of AirPassengers")
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.show()

# 10. Draw the regressor line (Linear regression)
# Fit linear model
from sklearn.linear_model import LinearRegression

# Prepare the data for the regression line
HarshArya['time'] = np.arange(len(HarshArya))  # Create a time
variable
X = HarshArya['time'].values.reshape(-1, 1)
y = HarshArya['airpassengers']

# Create and fit the model
model = LinearRegression()
model.fit(X, y)

# Predict the values using the model
y_pred = model.predict(X)

# Plot the data and the regression line
plt.figure(figsize=(10, 6))
plt.plot(HarshArya.index, HarshArya['airpassengers'], label='Air
Passengers')
plt.plot(HarshArya.index, y_pred, color='red', label='Linear Trend
Line')
plt.title("21BDS0098 - AirPassengers with Linear Trend Line")
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.legend()
plt.show()

# 11. Print the cycle across the years for the dataset
print("\nCycle across the years:")
print(HarshArya.index.to_period('M').month)  # Cycle (months)
```

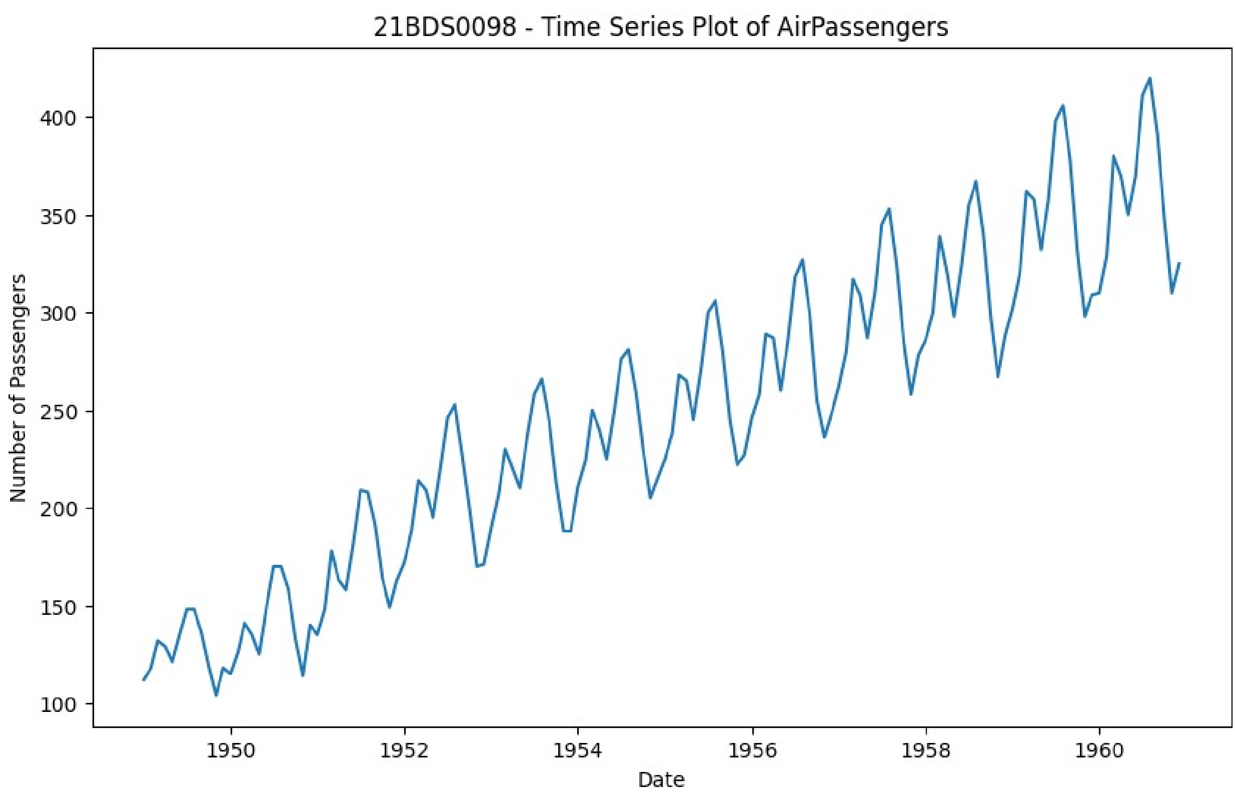
```python
# 12. Make the dataset stationary
# a. Log transformation
HarshArya['log_airpassengers'] = np.log(HarshArya['airpassengers'])

# Plot the log-transformed data
plt.figure(figsize=(10, 6))
plt.plot(HarshArya.index, HarshArya['log_airpassengers'])
plt.title("21BDS0098 - Log Transformation of AirPassengers")
plt.xlabel('Date')
plt.ylabel('Log of Number of Passengers')
plt.show()

# b. Differencing to make the data stationary
HarshArya['stationary'] =
HarshArya['log_airpassengers'].diff().dropna()

# Plot the stationary data
plt.figure(figsize=(10, 6))
plt.plot(HarshArya.index[1:], HarshArya['stationary'][1:])
plt.title("21BDS0098 - Stationary Series (Differenced Log)")
plt.xlabel('Date')
plt.ylabel('Differenced Log of Passengers')
plt.show()

# 13. Plot a box plot across months for seasonal effect
plt.figure(figsize=(10, 6))
sns.boxplot(x=HarshArya.index.month, y=HarshArya['airpassengers'])
plt.title("21BDS0098 - Box Plot Across Months for Seasonal Effect")
plt.xlabel('Month')
plt.ylabel('Number of Passengers')
plt.show()
```

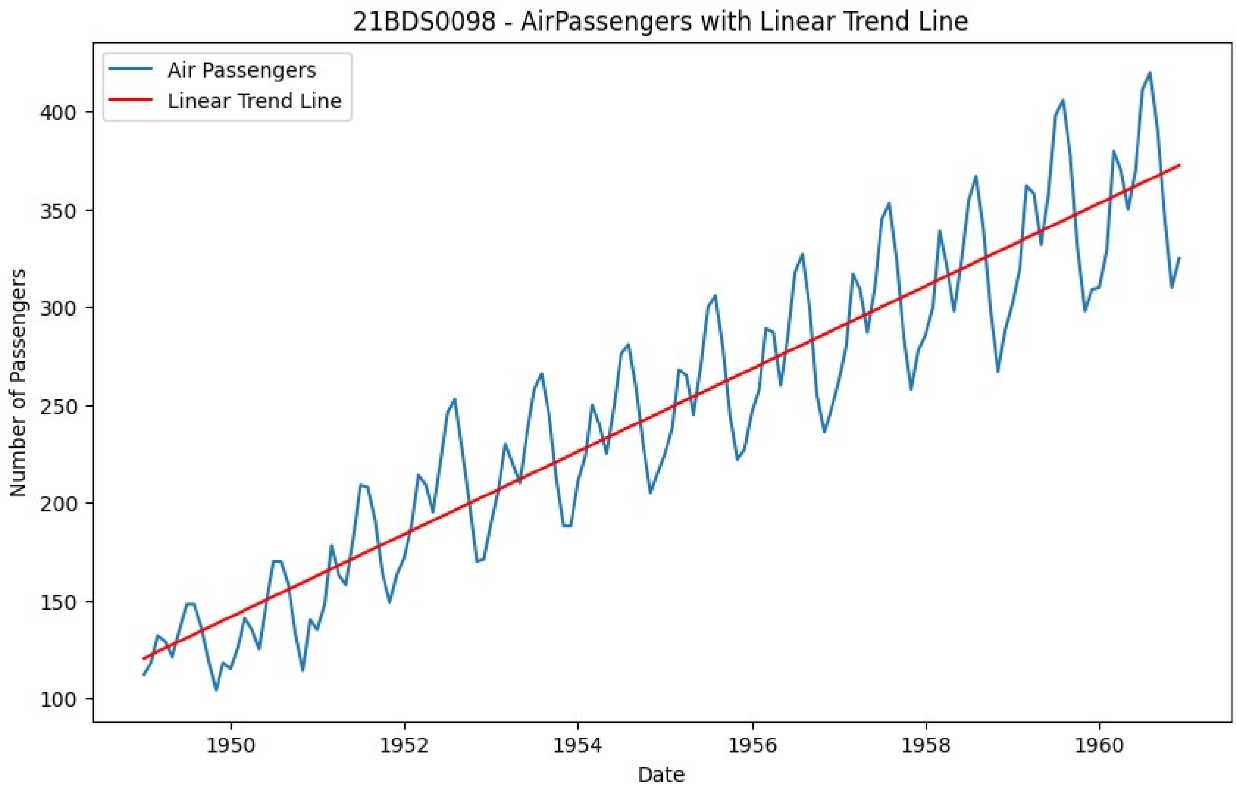
```
Structure and Data type of the dataset:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 144 entries, 1949-01-01 to 1960-12-01
Freq: MS
Data columns (total 1 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   airpassengers  144 non-null    int64
dtypes: int64(1)
memory usage: 2.2 KB
None

Missing values in the dataset:
airpassengers    0
dtype: int64

Starting date and Ending date of the dataset:
Start Date:  1949-01-01 00:00:00
```

```
End Date:  1960-12-01 00:00:00

Frequency of the dataset:
<MonthBegin>

Summary of the dataset:
       airpassengers
count     144.000000
mean      246.381944
std        78.742654
min       104.000000
25%       186.500000
50%       247.000000
75%       306.750000
max       420.000000
```

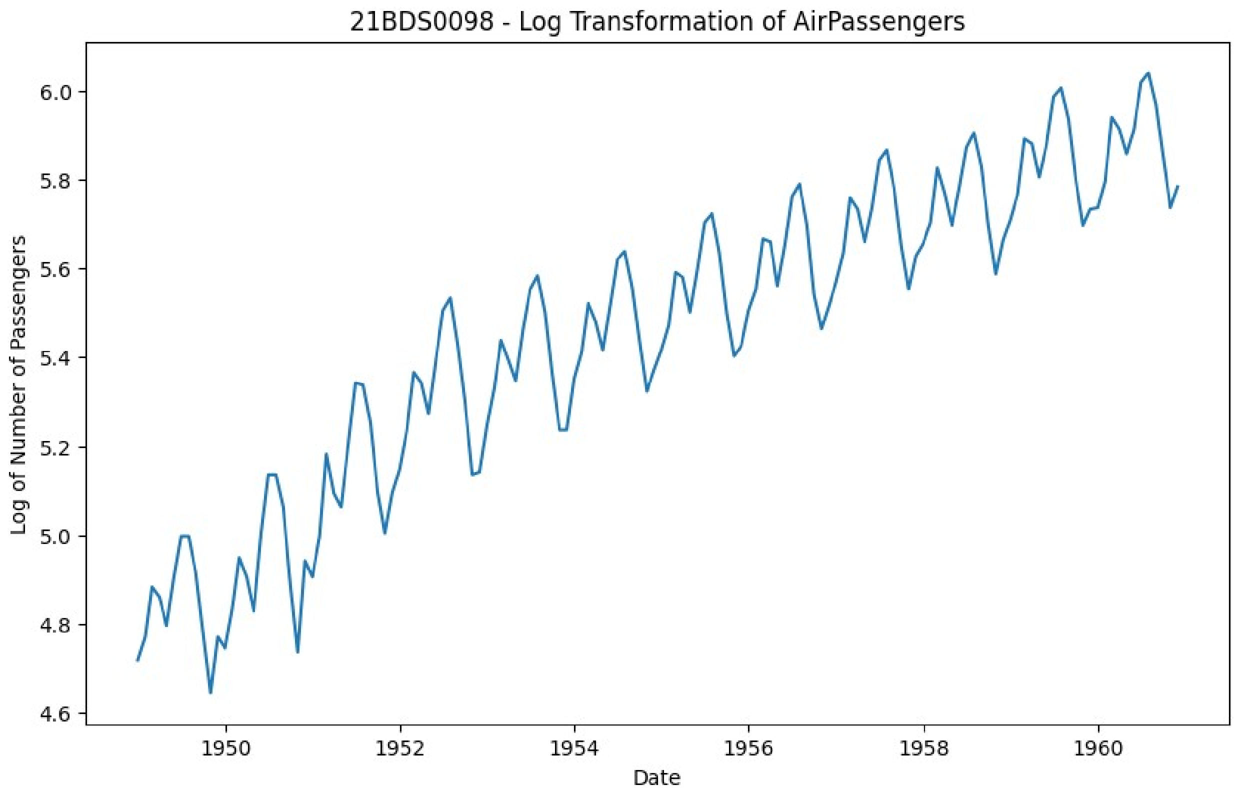


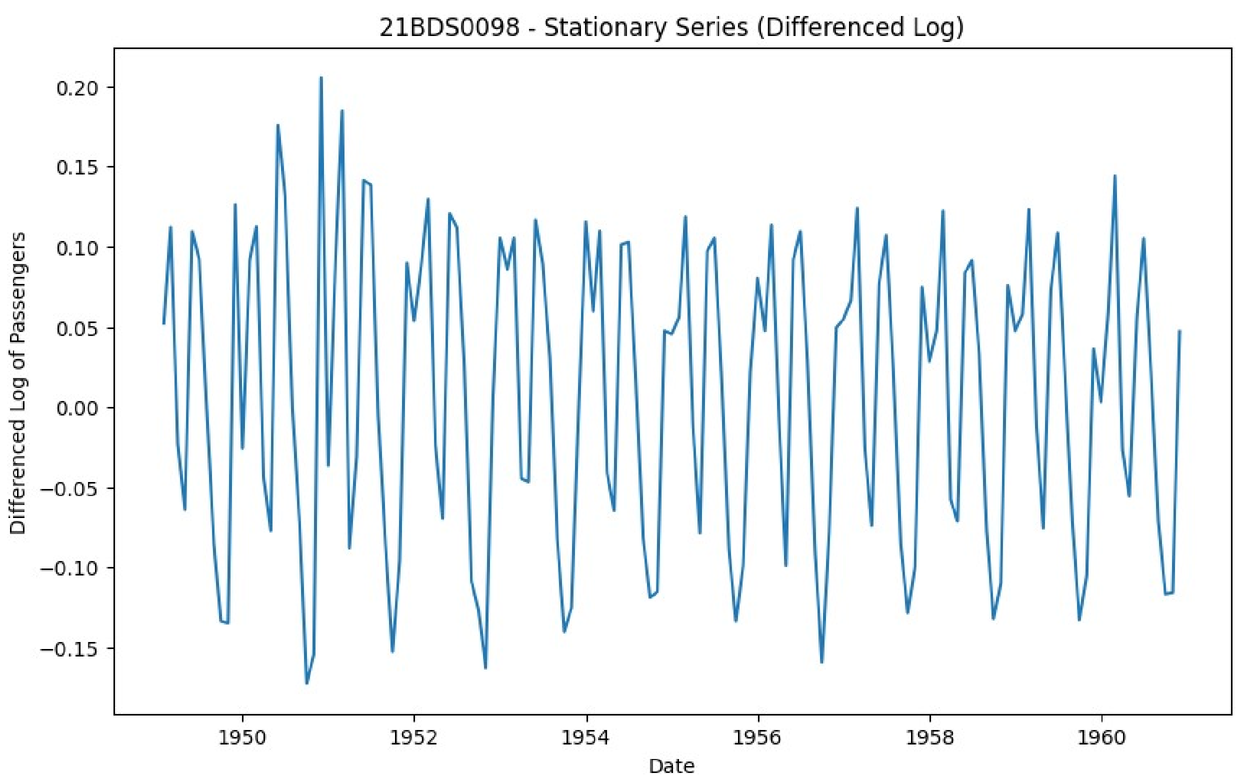

21BDS0098 - AirPassengers Dataset
Number of Passengers
100
150
200
250
300
350
400
1949
1951
1953
1955
1957
1959
Date

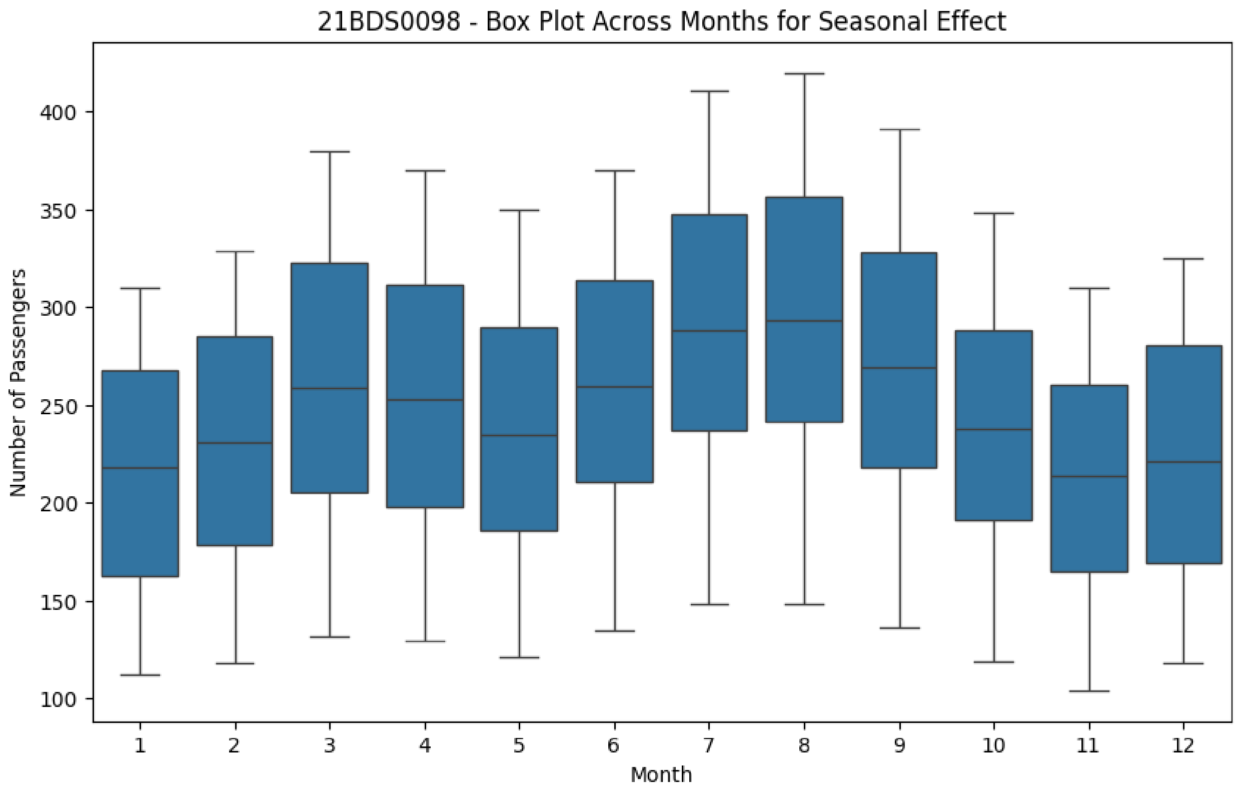

21BDS0098 - Time Series Plot of AirPassengers
Number of Passengers
100
150
200
250
300
350
400
1950
1952
1954
1956
1958
1960
Date




```
Cycle across the years:
Index([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10,
       ...
        3,  4,  5,  6,  7,  8,  9, 10, 11, 12],
      dtype='int64', length=144)
```


21BDS0098 - Log Transformation of AirPassengers
Log of Number of Passengers
Date
6.0
5.8
5.6
5.4
5.2
5.0
4.8
4.6
1950
1952
1954
1956
1958
1960


21BDS0098 - Stationary Series (Differenced Log)
Differenced Log of Passengers
Date
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
1950
1952
1954
1956
1958
1960



MODULE 4

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew, kurtosis

# Load the 'mpg' dataset from seaborn (which is a common dataset,
similar to mtcars)
HarshArya = sns.load_dataset('mpg').dropna()  # Load 'mpg' dataset and
remove rows with NaN values

# Display the first few rows of the dataset
print(HarshArya.head())

# Ensure the columns are numeric, checking for non-numeric values.
HarshArya = HarshArya.apply(pd.to_numeric, errors='coerce')

# Check column names to confirm the exact name of 'cyl' or equivalent
print(HarshArya.columns)

# 1. Measure of Central Tendency
```

```
## Mean (Arithmetic Mean)
mean_values_0098 = HarshArya.mean()
print("Mean for each variable:")
print(mean_values_0098)

## Median
median_values_0098 = HarshArya.median()
print("Median for each variable:")
print(median_values_0098)

## Quantiles (25%, 50%, 75%)
quantile_values_0098 = HarshArya.quantile([0.25, 0.5, 0.75])
print("Quantiles (25%, 50%, 75%) for each variable:")
print(quantile_values_0098)

## Deciles (using pd.qcut)
deciles_0098 = pd.qcut(HarshArya['mpg'], 10, labels=False) + 1  #
Create deciles for 'mpg'
print("Deciles for mpg variable:")
print(deciles_0098.value_counts())

## Percentiles (10%, 50%, 90%)
percentile_values_0098 = HarshArya.quantile([0.1, 0.5, 0.9])
print("Percentiles (10%, 50%, 90%) for each variable:")
print(percentile_values_0098)

# 2. Measure of Dispersions

## Range (max - min)
range_values_0098 = HarshArya.max() - HarshArya.min()
print("Range for each variable:")
print(range_values_0098)

## Interquartile Range (IQR)
iqr_values_0098 = HarshArya.quantile(0.75) - HarshArya.quantile(0.25)
print("Interquartile Range (IQR) for each variable:")
print(iqr_values_0098)

## Interdecile Range (90th - 10th percentile)
interdecile_values_0098 = HarshArya.quantile(0.9) -
HarshArya.quantile(0.1)
print("Interdecile Range for each variable:")
print(interdecile_values_0098)

## Standard Deviation
sd_values_0098 = HarshArya.std()
print("Standard Deviation for each variable:")
print(sd_values_0098)
```

```python
## Variance
variance_values_0098 = HarshArya.var()
print("Variance for each variable:")
print(variance_values_0098)

## Skewness
skewness_values_0098 = HarshArya.apply(skew)
print("Skewness for each variable:")
print(skewness_values_0098)

## Kurtosis
kurtosis_values_0098 = HarshArya.apply(kurtosis)
print("Kurtosis for each variable:")
print(kurtosis_values_0098)

# 3. Frequency Distribution

## Frequency Distribution (Table)
freq_table_0098 = HarshArya.apply(pd.value_counts)
print("Frequency distribution for each variable:")
print(freq_table_0098)

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

# Plot histogram for key numeric variables
plt.subplot(2, 2, 1)
HarshArya['mpg'].hist(color='skyblue', edgecolor='black')
plt.title('21BDS0098 - Histogram of MPG')
plt.xlabel('Miles per Gallon')
plt.ylabel('Frequency')

plt.subplot(2, 2, 2)
HarshArya['horsepower'].hist(color='lightgreen', edgecolor='black')
plt.title('21BDS0098 - Histogram of Horsepower')
plt.xlabel('Horsepower')
plt.ylabel('Frequency')

plt.subplot(2, 2, 3)
HarshArya['weight'].hist(color='lightcoral', edgecolor='black')
plt.title('21BDS0098 - Histogram of Weight')
plt.xlabel('Weight')
plt.ylabel('Frequency')

plt.subplot(2, 2, 4)
HarshArya['acceleration'].hist(color='lightyellow', edgecolor='black')
plt.title('21BDS0098 - Histogram of Acceleration')
plt.xlabel('Acceleration')
plt.ylabel('Frequency')
```

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

## Relative Frequency Distribution
relative_freq_mpg_0098 = HarshArya['mpg'].value_counts(normalize=True)
print("Relative Frequency Distribution for MPG:")
print(relative_freq_mpg_0098)

## Cumulative Frequency Distribution
cumulative_freq_mpg_0098 = HarshArya['mpg'].value_counts().cumsum()
print("Cumulative Frequency Distribution for MPG:")
print(cumulative_freq_mpg_0098)

# 4. Categorical Variable Analysis

## Check if 'cylinders' column is available
if 'cylinders' in HarshArya.columns:
    # Pie Plot for number of cylinders (cylinders)
    cyl_counts_0098 = HarshArya['cylinders'].value_counts()
    plt.figure(figsize=(7, 7))
    cyl_counts_0098.plot.pie(autopct='%1.1f%%', colors=['lightblue',
'lightgreen', 'lightcoral'], startangle=90)
    plt.title('21BDS0098 - Pie Plot for Number of Cylinders')
    plt.ylabel('')
    plt.show()

    # Stacked Bar Plot for cylinders vs origin
    plt.figure(figsize=(8, 6))
    sns.countplot(x='cylinders', hue='origin', data=HarshArya,
dodge=False, palette="Set2")
    plt.title("21BDS0098 - Stacked Bar Plot for Cylinders and Origin")
    plt.xlabel("Number of Cylinders")
    plt.ylabel("Count")
    plt.show()
else:
    print("Column 'cylinders' not found in the dataset.")

# 5. Summary of all measures (Mean, Median, etc.)
summary_stats_0098 = pd.DataFrame({
    'Mean': mean_values_0098,
    'Median': median_values_0098,
    'IQR': iqr_values_0098,
    'Standard Deviation': sd_values_0098,
    'Skewness': skewness_values_0098,
    'Kurtosis': kurtosis_values_0098
})
print("Summary Statistics for the dataset:")
print(summary_stats_0098)
```

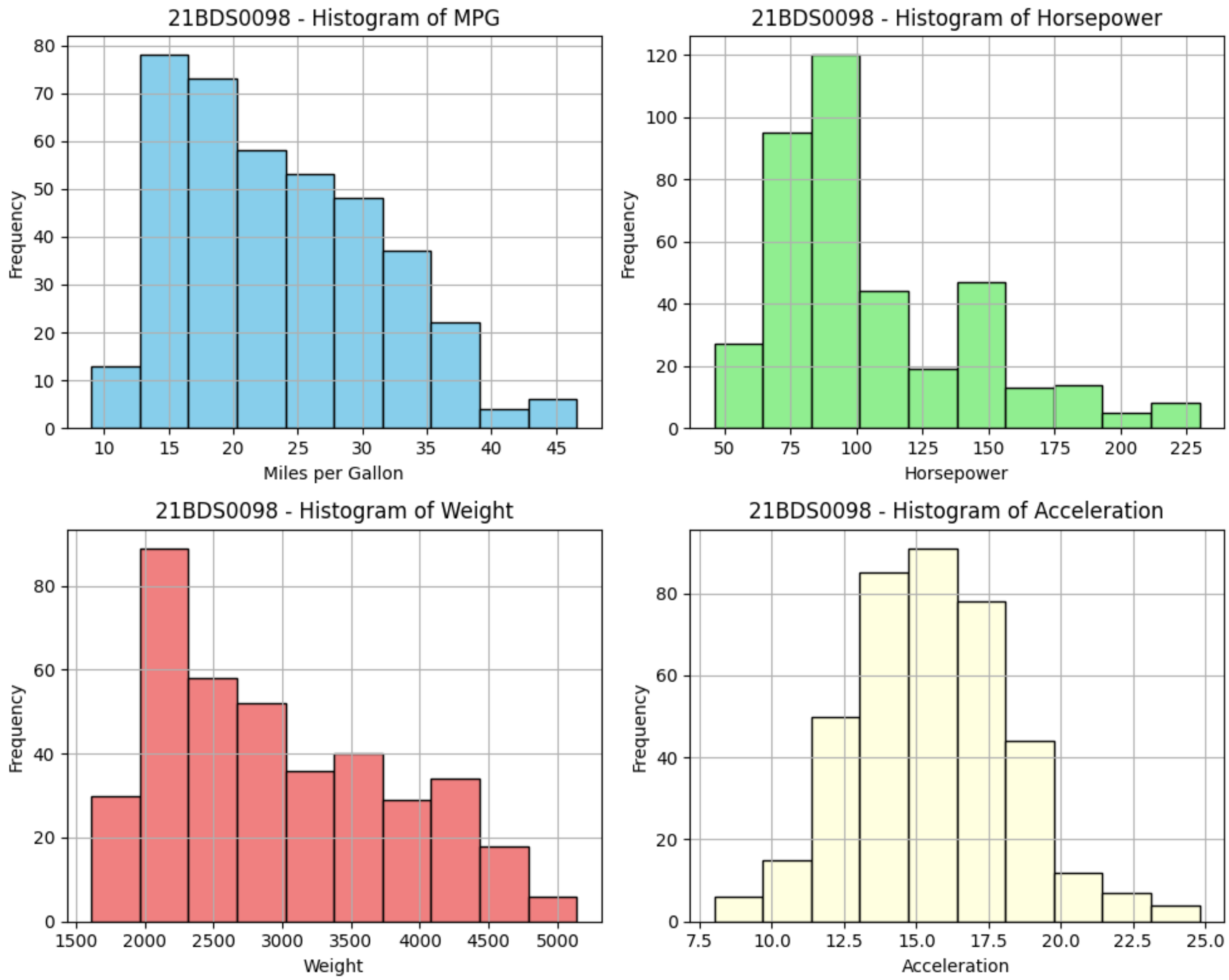
```
    mpg  cylinders  displacement  horsepower  weight  acceleration  \
0  18.0          8         307.0       130.0    3504          12.0
1  15.0          8         350.0       165.0    3693          11.5
2  18.0          8         318.0       150.0    3436          11.0
3  16.0          8         304.0       150.0    3433          12.0
4  17.0          8         302.0       140.0    3449          10.5

   model_year origin                       name
0          70    usa  chevrolet chevelle malibu
1          70    usa          buick skylark 320
2          70    usa         plymouth satellite
3          70    usa              amc rebel sst
4          70    usa                ford torino
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model_year', 'origin', 'name'],
      dtype='object')
Mean for each variable:
mpg               23.445918
cylinders          5.471939
displacement     194.411990
horsepower       104.469388
weight          2977.584184
acceleration      15.541327
model_year        75.979592
origin                  NaN
name                    NaN
dtype: float64
Median for each variable:
mpg               22.75
cylinders          4.00
displacement     151.00
horsepower        93.50
weight          2803.50
acceleration      15.50
model_year        76.00
origin              NaN
name                NaN
dtype: float64
Quantiles (25%, 50%, 75%) for each variable:
        mpg  cylinders  displacement  horsepower   weight
acceleration  \
0.25  17.00        4.0        105.00        75.0  2225.25
13.775
0.50  22.75        4.0        151.00        93.5  2803.50
15.500
0.75  29.00        8.0        275.75       126.0  3614.75
17.025

      model_year  origin  name
0.25        73.0     NaN   NaN
```

```
0.50        76.0     NaN   NaN
0.75        79.0     NaN   NaN
Deciles for mpg variable:
mpg
1     52
6     40
10    40
8     39
9     39
7     38
3     37
5     36
4     36
2     35
Name: count, dtype: int64
Percentiles (10%, 50%, 90%) for each variable:
       mpg  cylinders  displacement  horsepower  weight  acceleration  \
0.1  14.00        4.0          90.0        67.0  1990.0          12.0   

0.5  22.75        4.0         151.0        93.5  2803.5          15.5   

0.9  34.19        8.0         350.0       157.7  4277.6          19.0   


     model_year  origin  name
0.1        71.0     NaN   NaN
0.5        76.0     NaN   NaN
0.9        81.0     NaN   NaN
Range for each variable:
mpg               37.6
cylinders          5.0
displacement     387.0
horsepower       184.0
weight          3527.0
acceleration      16.8
model_year        12.0
origin             NaN
name               NaN
dtype: float64
Interquartile Range (IQR) for each variable:
mpg               12.00
cylinders          4.00
displacement     170.75
horsepower        51.00
weight          1389.50
acceleration       3.25
model_year         6.00
origin              NaN
name                NaN
```

```
dtype: float64
Interdecile Range for each variable:
mpg               20.19
cylinders          4.00
displacement     260.00
horsepower        90.70
weight          2287.60
acceleration       7.00
model_year        10.00
origin              NaN
name                NaN
dtype: float64
Standard Deviation for each variable:
mpg               7.805007
cylinders         1.705783
displacement    104.644004
horsepower       38.491160
weight          849.402560
acceleration      2.758864
model_year        3.683737
origin                 NaN
name                   NaN
dtype: float64
Variance for each variable:
mpg                 60.918142
cylinders            2.909696
displacement     10950.367554
horsepower        1481.569393
weight          721484.709008
acceleration         7.611331
model_year          13.569915
origin                    NaN
name                      NaN
dtype: float64
Skewness for each variable:
mpg             0.455341
cylinders       0.506163
displacement    0.698981
horsepower      1.083161
weight          0.517595
acceleration    0.290470
model_year      0.019613
origin               NaN
name                 NaN
dtype: float64
Kurtosis for each variable:
mpg            -0.524703
cylinders      -1.395695
displacement   -0.783692
horsepower      0.672822
```

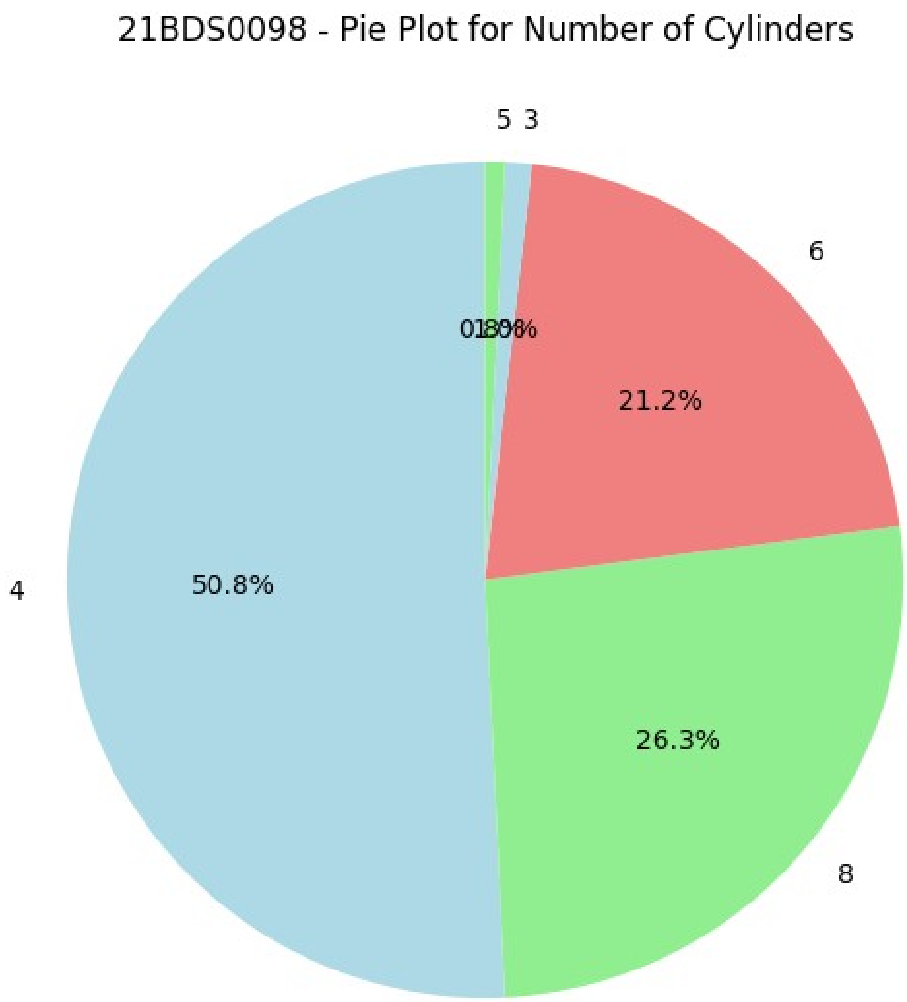
```
weight          -0.814241
acceleration     0.423320
model_year      -1.167876
origin                NaN
name                  NaN
dtype: float64
Frequency distribution for each variable:
        mpg  cylinders  displacement  horsepower  weight  acceleration
\
3.0     NaN        4.0           NaN         NaN     NaN           NaN

4.0     NaN      199.0           NaN         NaN     NaN           NaN

5.0     NaN        3.0           NaN         NaN     NaN           NaN

6.0     NaN       83.0           NaN         NaN     NaN           NaN

8.0     NaN      103.0           NaN         NaN     NaN           1.0

...     ...        ...           ...         ...     ...           ...

4951.0  NaN        NaN           NaN         NaN     1.0           NaN

4952.0  NaN        NaN           NaN         NaN     1.0           NaN

4955.0  NaN        NaN           NaN         NaN     1.0           NaN

4997.0  NaN        NaN           NaN         NaN     1.0           NaN

5140.0  NaN        NaN           NaN         NaN     1.0           NaN


        model_year  origin  name
3.0            NaN     NaN   NaN
4.0            NaN     NaN   NaN
5.0            NaN     NaN   NaN
6.0            NaN     NaN   NaN
8.0            NaN     NaN   NaN
...            ...     ...   ...
4951.0         NaN     NaN   NaN
4952.0         NaN     NaN   NaN
4955.0         NaN     NaN   NaN
4997.0         NaN     NaN   NaN
5140.0         NaN     NaN   NaN

[679 rows x 9 columns]

<ipython-input-93-e74e52ec1767>:87: FutureWarning:

pandas.value_counts is deprecated and will be removed in a future
```

```
version. Use pd.Series(obj).value_counts() instead.
```



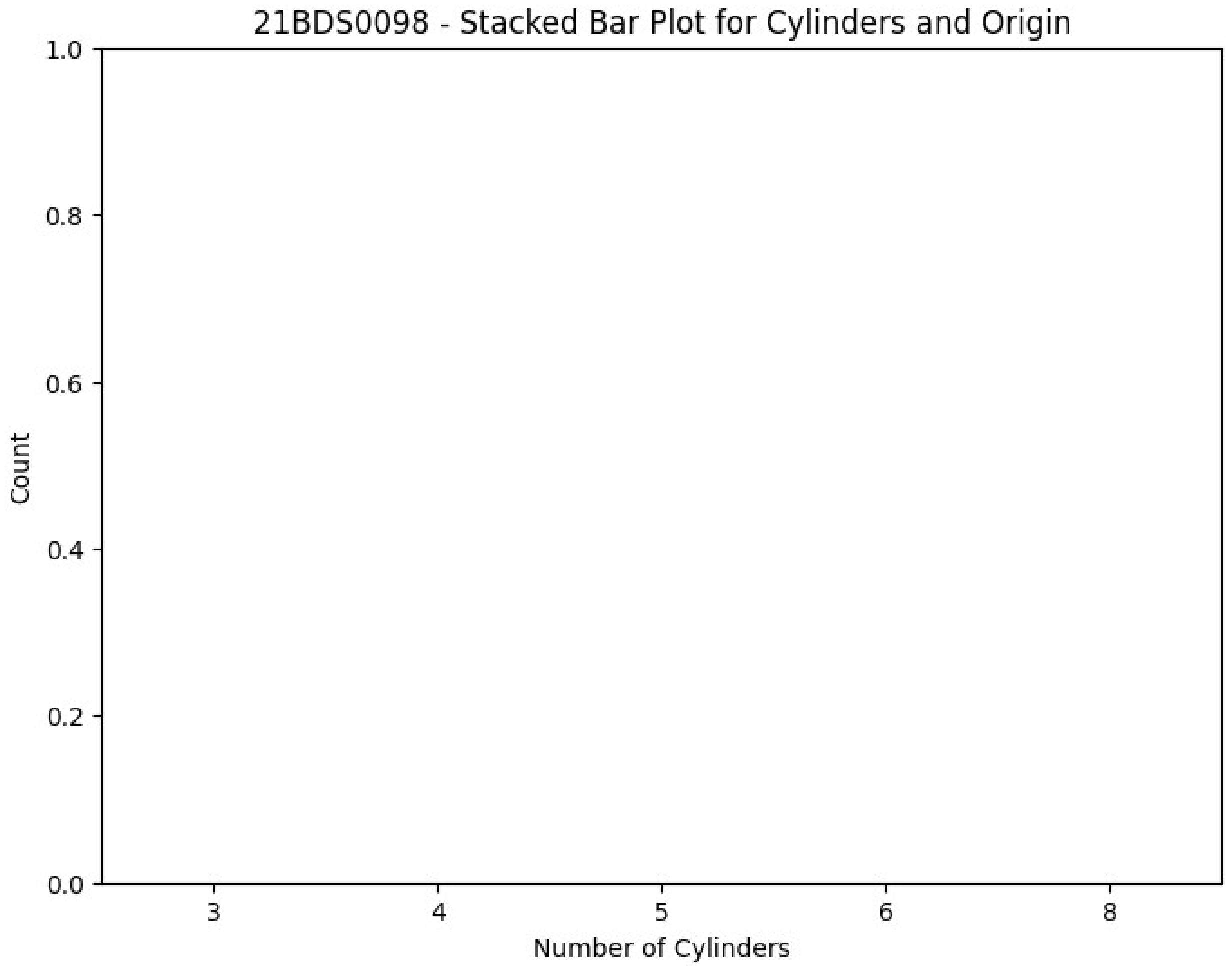

```
Relative Frequency Distribution for MPG:
mpg
13.0    0.051020
14.0    0.048469
18.0    0.043367
15.0    0.040816
26.0    0.035714
          ...
31.9    0.002551
16.9    0.002551
18.2    0.002551
22.3    0.002551
44.0    0.002551
Name: proportion, Length: 127, dtype: float64
Cumulative Frequency Distribution for MPG:
mpg
```

```
13.0     20
14.0     39
18.0     56
15.0     72
26.0     86
       ...
31.9    388
16.9    389
18.2    390
22.3    391
44.0    392
Name: count, Length: 127, dtype: int64
```




```
<ipython-input-93-e74e52ec1767>:146: UserWarning:
```

```
Ignoring `palette` because no `hue` variable has been assigned.
```




```
Summary Statistics for the dataset:
                     Mean   Median      IQR  Standard Deviation
Skewness  \
mpg             23.445918    22.75    12.00            7.805007
0.455341
cylinders        5.471939     4.00     4.00            1.705783
0.506163
displacement   194.411990   151.00   170.75          104.644004
0.698981
horsepower     104.469388    93.50    51.00           38.491160
1.083161
weight        2977.584184  2803.50  1389.50          849.402560
0.517595
acceleration    15.541327    15.50     3.25            2.758864
0.290470
model_year      75.979592    76.00     6.00            3.683737
```

```
0.019613
origin                NaN      NaN      NaN                 NaN
NaN
name                  NaN      NaN      NaN                 NaN
NaN

              Kurtosis
mpg          -0.524703
cylinders    -1.395695
displacement -0.783692
horsepower    0.672822
weight       -0.814241
acceleration  0.423320
model_year   -1.167876
origin             NaN
name               NaN

# Import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
from mpl_toolkits.mplot3d import Axes3D
import plotly.express as px

# Load the Titanic dataset from seaborn (or use your own dataset)
url =
"https://raw.githubusercontent.com/datasciencedojo/datasets/master/
titanic.csv"
HarshArya_0098 = pd.read_csv(url)

# Check the first few rows of the dataset to verify column names
print(HarshArya_0098.head())

# Check the column names to make sure 'Survived' is present
print("\nColumns in the dataset:")
print(HarshArya_0098.columns)

# Data Cleaning
# Fill missing values in 'Age' with the mean and in 'Embarked' with
the mode (most frequent value)
HarshArya_0098['Age'].fillna(HarshArya_0098['Age'].mean(),
inplace=True)
HarshArya_0098['Embarked'].fillna(HarshArya_0098['Embarked'].mode()
[0], inplace=True)

# 3. Create a 2-way contingency table (Categorical vs Categorical)
Contingency_Table_0098 = pd.crosstab(HarshArya_0098['Sex'],
HarshArya_0098['Survived'])
```

```
print("\nContingency Table (Categorical vs Categorical):")
print(Contingency_Table_0098)

# 4. Create a 3-way contingency table (Categorical vs Categorical vs
Categorical)
Contingency_Table_3way_0098 = pd.crosstab([HarshArya_0098['Sex'],
HarshArya_0098['Embarked']], HarshArya_0098['Survived'])
print("\n3-Way Contingency Table:")
print(Contingency_Table_3way_0098)

# 5. Apply row profile, column profile, and chi-square on one of the
contingency tables
Chi2_0098, P_0098, Dof_0098, Expected_0098 =
chi2_contingency(Contingency_Table_0098)
print(f"\nChi-Square Test Result:\nChi2: {Chi2_0098}, P-value:
{P_0098}, Degrees of Freedom: {Dof_0098}")
print("Expected Frequencies:")
print(Expected_0098)

# Row profile
Row_Profile_0098 =
Contingency_Table_0098.div(Contingency_Table_0098.sum(axis=1), axis=0)
print("\nRow Profile (Proportions per Row):")
print(Row_Profile_0098)

# Column profile
Column_Profile_0098 =
Contingency_Table_0098.div(Contingency_Table_0098.sum(axis=0), axis=1)
print("\nColumn Profile (Proportions per Column):")
print(Column_Profile_0098)

# Relative Frequency
Relative_Frequency_0098 = Contingency_Table_0098 /
Contingency_Table_0098.sum().sum()
print("\nRelative Frequency Table:")
print(Relative_Frequency_0098)

# 6. Scatter Plot (Categorical vs Numerical)
sns.scatterplot(x='Age', y='Fare', hue='Survived',
data=HarshArya_0098)
plt.title('Scatter Plot: Age vs Fare with Survival Status')
plt.show()

# 7. Scatter Plot for 3 Variables (Age, Fare, and Pclass)
sns.scatterplot(x='Age', y='Fare', hue='Survived', style='Pclass',
data=HarshArya_0098)
plt.title('3D Scatter Plot: Age, Fare, and Pclass')
plt.show()

# 8. Change color, shape, and add horizontal bars to the scatter plot
```

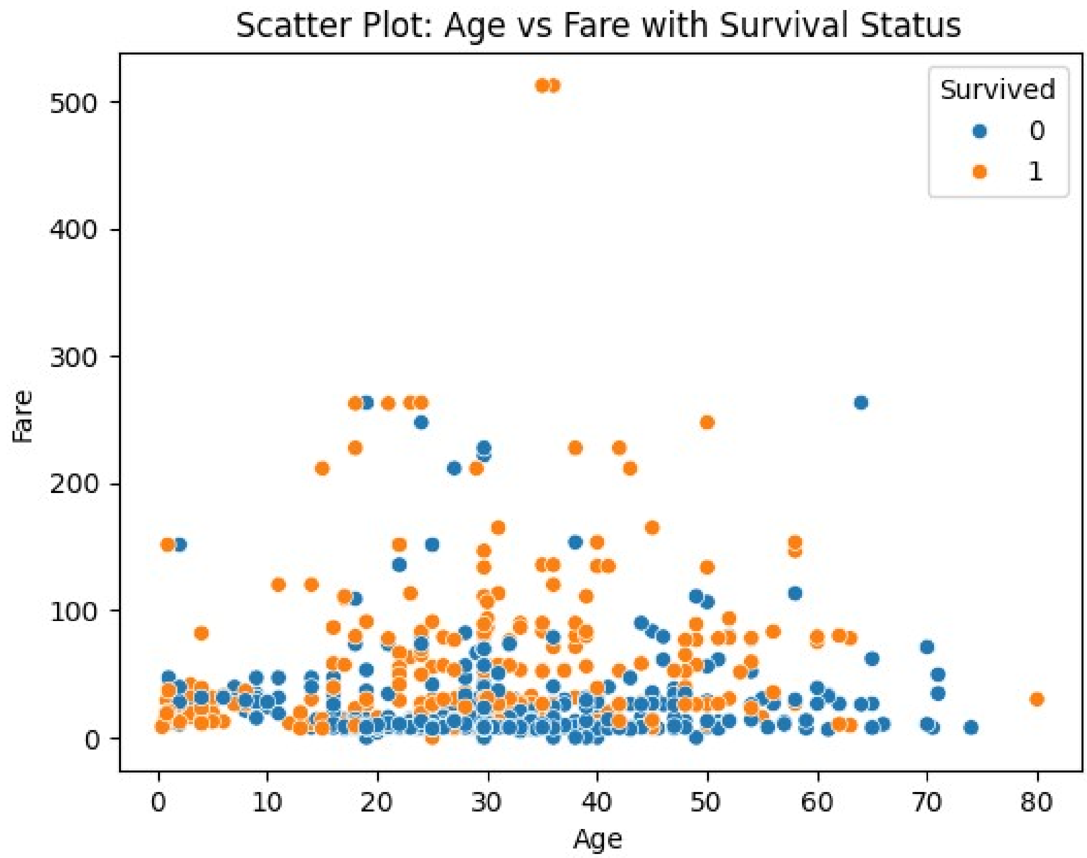
```
sns.scatterplot(x='Age', y='Fare', hue='Survived', style='Sex',
data=HarshArya_0098)
plt.title('Scatter Plot: Age vs Fare with Survival and Gender')
plt.show()

# 9. 3D Scatter Plot (Age, Fare, and Pclass as 3D Axes)
fig_0098 = plt.figure()
ax_0098 = fig_0098.add_subplot(111, projection='3d')
ax_0098.scatter(HarshArya_0098['Age'], HarshArya_0098['Fare'],
HarshArya_0098['Pclass'], c=HarshArya_0098['Survived'],
cmap='coolwarm')
ax_0098.set_xlabel('Age')
ax_0098.set_ylabel('Fare')
ax_0098.set_zlabel('Pclass')
plt.title('3D Scatter Plot: Age, Fare, and Pclass')
plt.show()

# 10. 2D Boxplot (Categorical vs Numerical)
sns.boxplot(x='Survived', y='Age', data=HarshArya_0098)
plt.title('Boxplot: Survival vs Age')
plt.show()

# 11. Radar Chart (Sunray Plot) using 'Pclass', 'Age', 'Fare',
'SibSp', 'Parch'
Categories_0098 = ['Pclass', 'Age', 'Fare', 'SibSp', 'Parch']  #
Example categories
Values_0098 = [
    HarshArya_0098['Pclass'].mode()[0],   # Mode of Pclass (most
frequent class)
    HarshArya_0098['Age'].mean(),         # Mean Age
    HarshArya_0098['Fare'].mean(),        # Mean Fare
    HarshArya_0098['SibSp'].mean(),       # Mean of SibSp
(siblings/spouses aboard)
    HarshArya_0098['Parch'].mean()        # Mean of Parch
(parents/children aboard)
]

# To close the radar chart, append the first value to the end of the
list
Values_0098.append(Values_0098[0])  # Append first value to make the
list a circle

# Calculate angle for each axis (360 degrees divided by number of
categories)
Angles_0098 = np.linspace(0, 2 * np.pi, len(Categories_0098),
endpoint=False).tolist()

# Append the first angle to close the circle (it must match the first
value of 'Values')
Angles_0098.append(Angles_0098[0])  # Ensure the loop is closed by
```

```
adding the first angle to the end of Angles

# Create Radar chart data setup
fig_0098 = plt.figure(figsize=(6, 6))
ax_0098 = fig_0098.add_subplot(111, polar=True)

# Plot the data
ax_0098.fill(Angles_0098, Values_0098, color='skyblue', alpha=0.25)  #
Fill the area
ax_0098.plot(Angles_0098, Values_0098, color='blue', linewidth=2)  #
Line around the plot

# Set the y-axis labels (empty because we're not showing radial
values)
ax_0098.set_yticklabels([])

# Set the x-ticks to be the category labels
ax_0098.set_xticks(Angles_0098[:-1])  # Exclude last angle to avoid
repetition
ax_0098.set_xticklabels(Categories_0098)

# Title and showing the plot
plt.title("21BDS0098 - Sunray Plot (Radar Chart)")
plt.show()
```

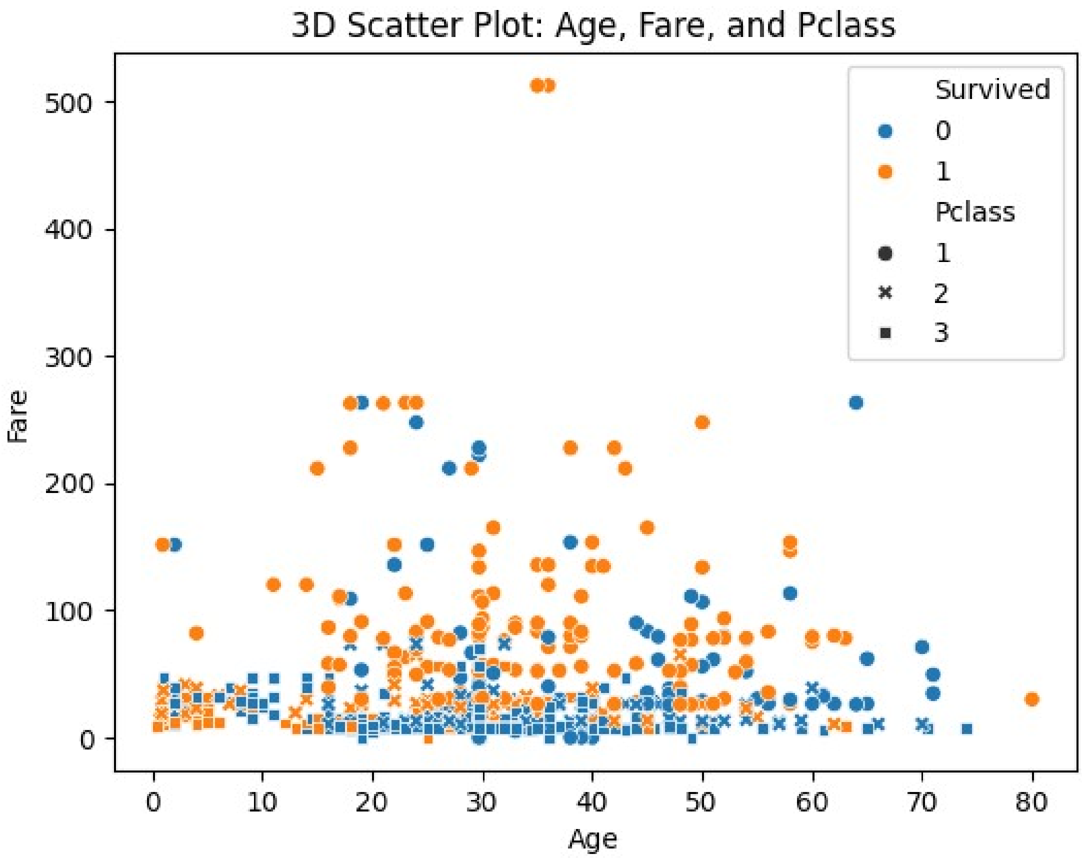
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age
SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0
1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                             Heikkinen, Miss. Laina  female  26.0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                           Allen, Mr. William Henry    male  35.0
0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
```

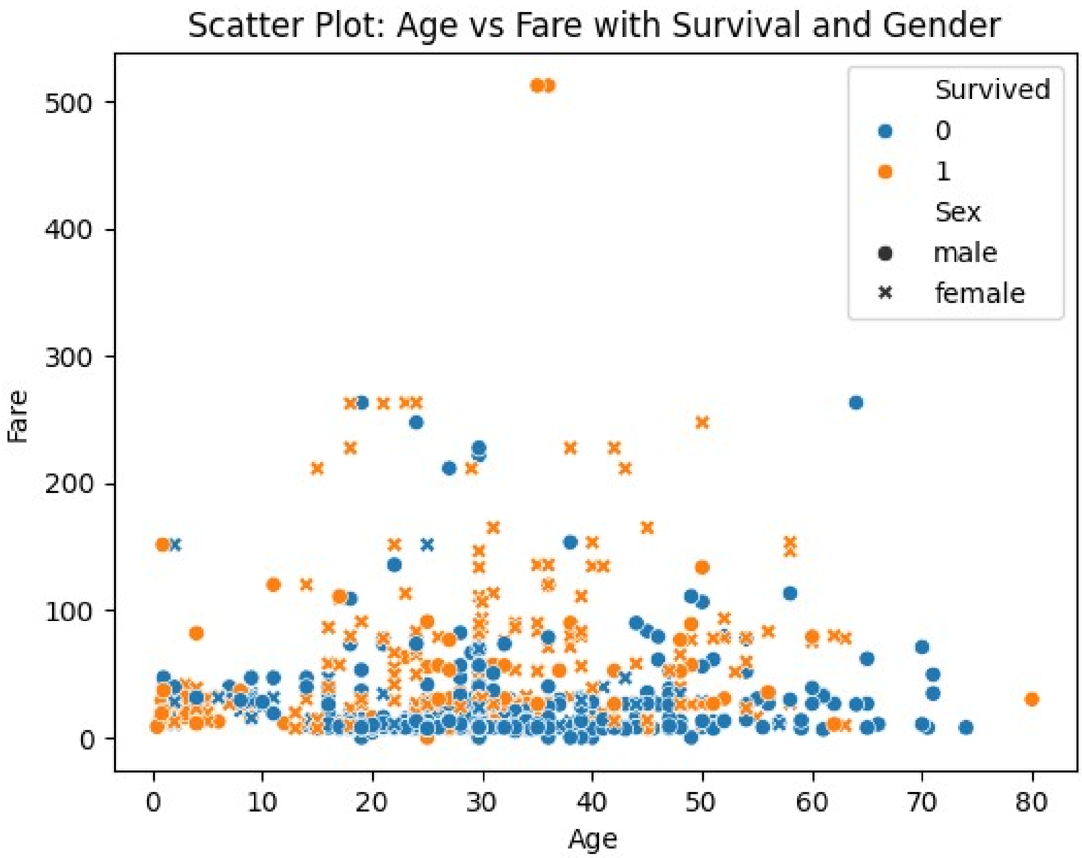
```
4      0            373450   8.0500   NaN        S

Columns in the dataset:
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age',
'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')

Contingency Table (Categorical vs Categorical):
Survived    0    1
Sex
female     81  233
male      468  109

3-Way Contingency Table:
Survived           0    1
Sex    Embarked
female C           9   64
       Q           9   27
       S          63  142
male   C          66   29
       Q          38    3
       S         364   77

Chi-Square Test Result:
Chi2: 260.71702016732104, P-value: 1.1973570627755645e-58, Degrees of
Freedom: 1
Expected Frequencies:
[[193.47474747 120.52525253]
 [355.52525253 221.47474747]]

Row Profile (Proportions per Row):
Survived         0         1
Sex
female    0.257962  0.742038
male      0.811092  0.188908

Column Profile (Proportions per Column):
Survived         0         1
Sex
female    0.147541  0.681287
male      0.852459  0.318713

Relative Frequency Table:
Survived         0         1
Sex
female    0.090909  0.261504
male      0.525253  0.122334
```

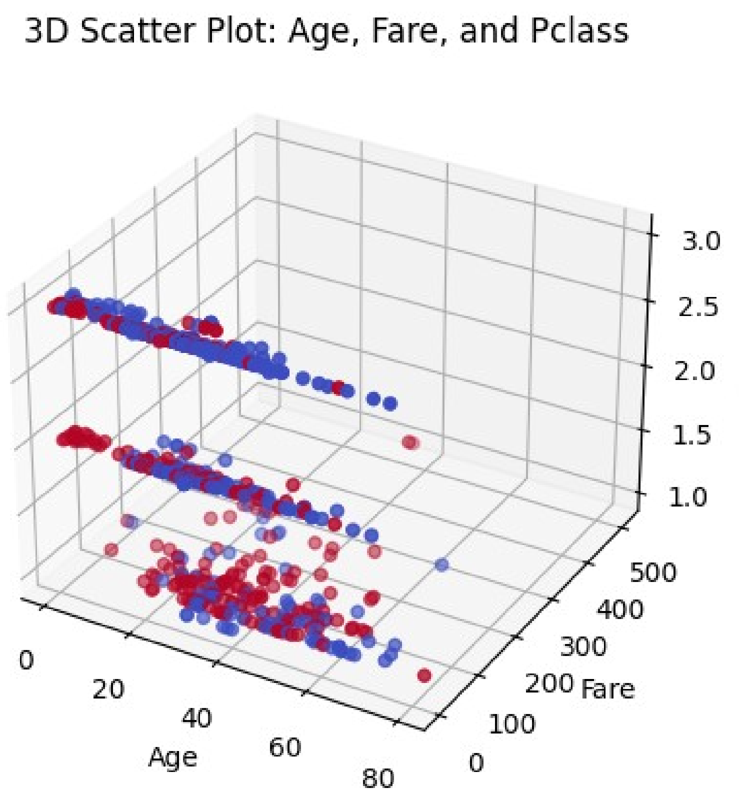
```
<ipython-input-94-c0ad1c9e4e64>:23: FutureWarning:

A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.



<ipython-input-94-c0ad1c9e4e64>:24: FutureWarning:

A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.
```

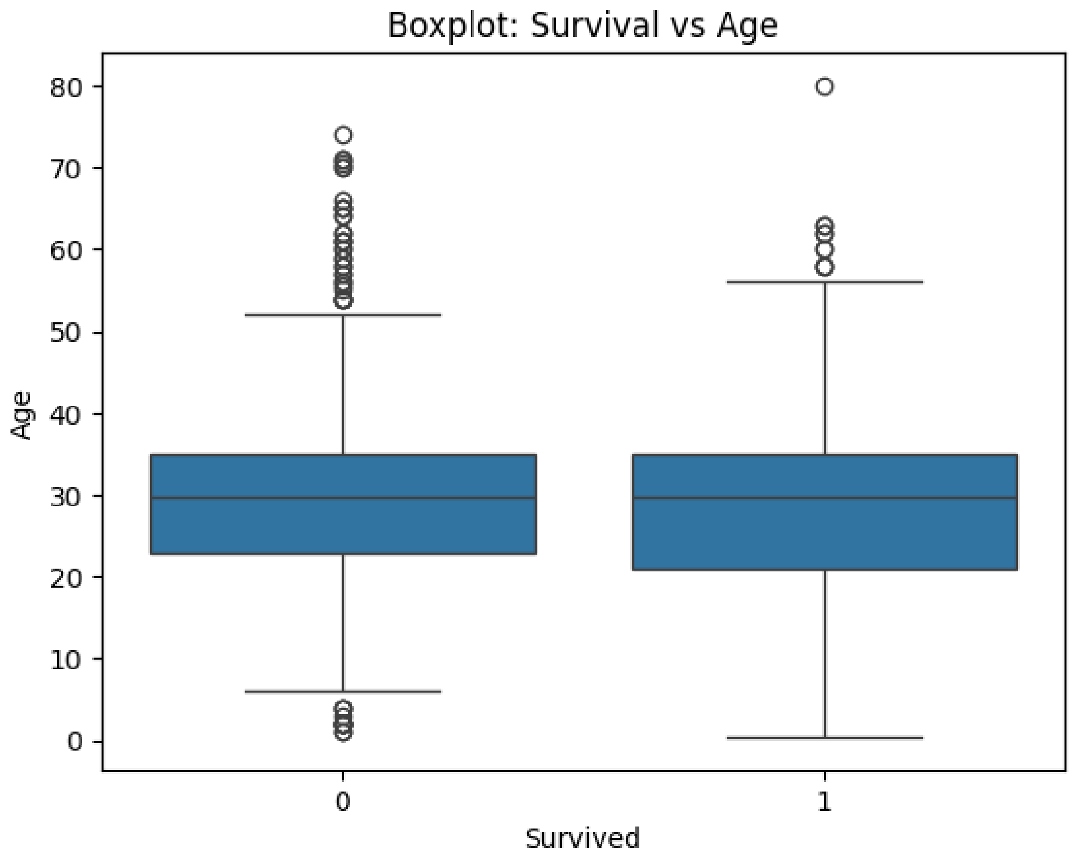


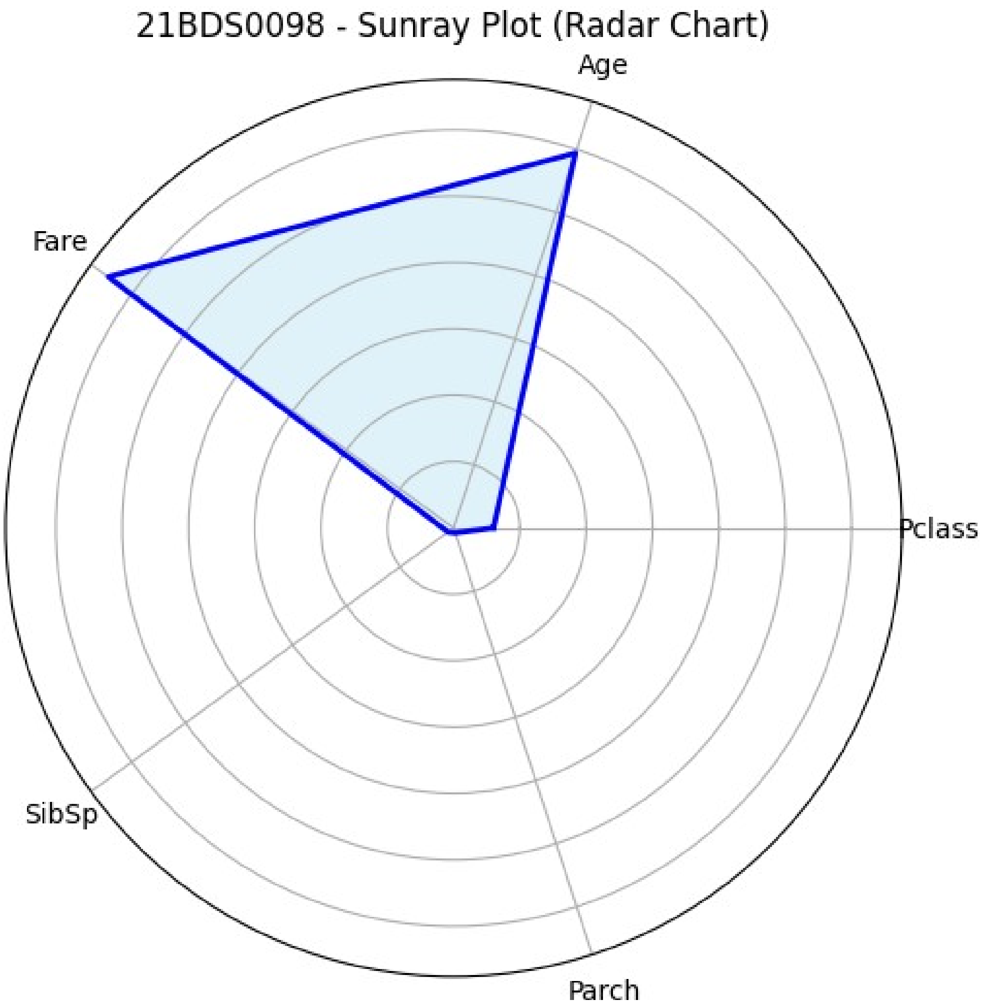
Scatter Plot: Age vs Fare with Survival Status
Survived
0
1
Fare
Age
0
100
200
300
400
500
0
10
20
30
40
50
60
70
80


3D Scatter Plot: Age, Fare, and Pclass
Survived
0
1
Pclass
1
2
3
Fare
500
400
300
200
100
0
Age
0
10
20
30
40
50
60
70
80


Scatter Plot: Age vs Fare with Survival and Gender
Survived
0
1
Sex
male
female
Fare
0
100
200
300
400
500
Age
0
10
20
30
40
50
60
70
80


3D Scatter Plot: Age, Fare, and Pclass
Age
0
20
40
60
80
Fare
0
100
200
300
400
500
1.0
1.5
2.0
2.5
3.0


Boxplot: Survival vs Age
Age
80
70
60
50
40
30
20
10
0
0
1
Survived



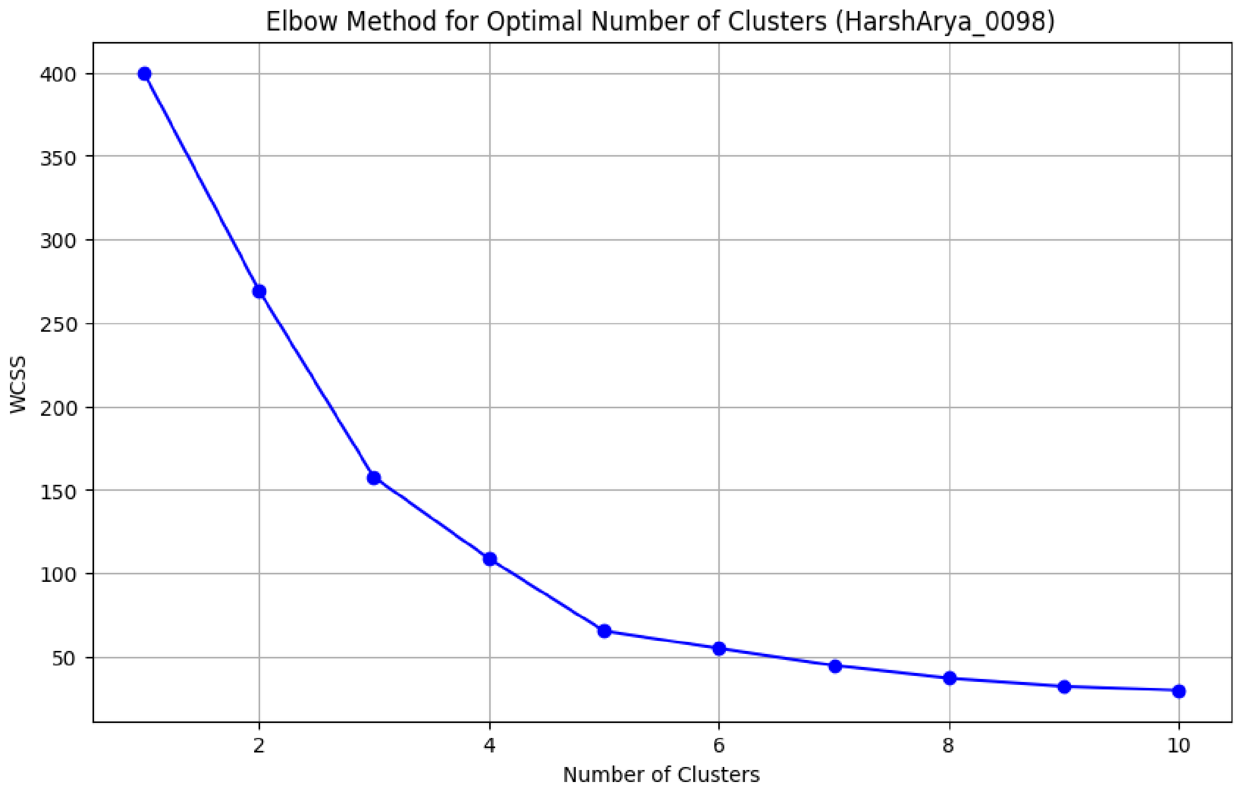
MODULE 5

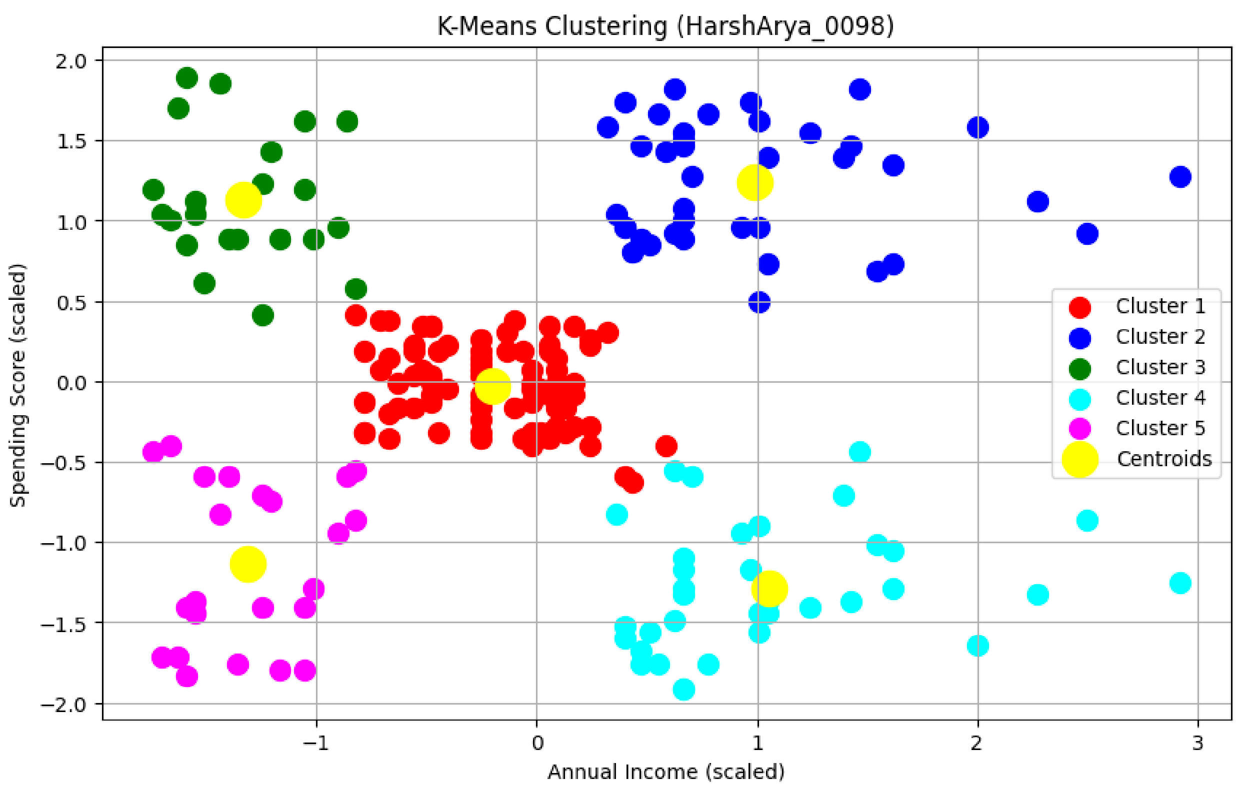
```
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Load the dataset
# For demonstration, let's assume the dataset is available as
'Mall_Customers.csv'
# Replace this path with the actual file path
HarshArya_0098 = pd.read_csv('Mall_Customers.csv')

# Displaying the first few rows of the dataset to understand its
structure
```

```
print(HarshArya_0098.head())

# Extracting 4th and 5th columns (assuming the columns are 'Annual
Income' and 'Spending Score')
X_0098 = HarshArya_0098.iloc[:, [3, 4]].values  # Select 4th and 5th
columns as features

# Feature Scaling - Standardize the data before clustering
scaler_0098 = StandardScaler()
X_scaled_0098 = scaler_0098.fit_transform(X_0098)

# Elbow Method to find the optimal number of clusters
wcss_0098 = []  # List to store the within-cluster sum of squares
(WCSS)
for i in range(1, 11):  # Try from 1 to 10 clusters
    kmeans_0098 = KMeans(n_clusters=i, init='k-means++', max_iter=300,
n_init=10, random_state=42)
    kmeans_0098.fit(X_scaled_0098)
    wcss_0098.append(kmeans_0098.inertia_)  # WCSS is the inertia
value

# Plotting the elbow graph to determine the optimal number of clusters
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), wcss_0098, marker='o', color='blue')
plt.title('Elbow Method for Optimal Number of Clusters
(HarshArya_0098)')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.grid(True)
plt.show()

# From the plot, you can visually determine the optimal number of
clusters.
# Let's assume it's 5 (you should adjust based on the elbow plot
observation).
optimal_clusters_0098 = 5

# Fitting KMeans to the dataset with the optimal number of clusters
kmeans_0098 = KMeans(n_clusters=optimal_clusters_0098, init='k-means+
+', max_iter=300, n_init=10, random_state=42)
y_kmeans_0098 = kmeans_0098.fit_predict(X_scaled_0098)

# Visualizing the clusters

# Plotting the clusters
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled_0098[y_kmeans_0098 == 0, 0],
X_scaled_0098[y_kmeans_0098 == 0, 1], s=100, c='red', label='Cluster
1')
plt.scatter(X_scaled_0098[y_kmeans_0098 == 1, 0],
```

```
X_scaled_0098[y_kmeans_0098 == 1, 1], s=100, c='blue', label='Cluster
2')
plt.scatter(X_scaled_0098[y_kmeans_0098 == 2, 0],
X_scaled_0098[y_kmeans_0098 == 2, 1], s=100, c='green', label='Cluster
3')
plt.scatter(X_scaled_0098[y_kmeans_0098 == 3, 0],
X_scaled_0098[y_kmeans_0098 == 3, 1], s=100, c='cyan', label='Cluster
4')
plt.scatter(X_scaled_0098[y_kmeans_0098 == 4, 0],
X_scaled_0098[y_kmeans_0098 == 4, 1], s=100, c='magenta',
label='Cluster 5')

# Plot the centroids
plt.scatter(kmeans_0098.cluster_centers_[:, 0],
kmeans_0098.cluster_centers_[:, 1], s=300, c='yellow',
label='Centroids')
plt.title('K-Means Clustering (HarshArya_0098)')
plt.xlabel('Annual Income (scaled)')
plt.ylabel('Spending Score (scaled)')
plt.legend()
plt.grid(True)
plt.show()
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40


Elbow Method for Optimal Number of Clusters (HarshArya_0098)
WCSS
Number of Clusters


K-Means Clustering (HarshArya_0098)
Spending Score (scaled)
Annual Income (scaled)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroids

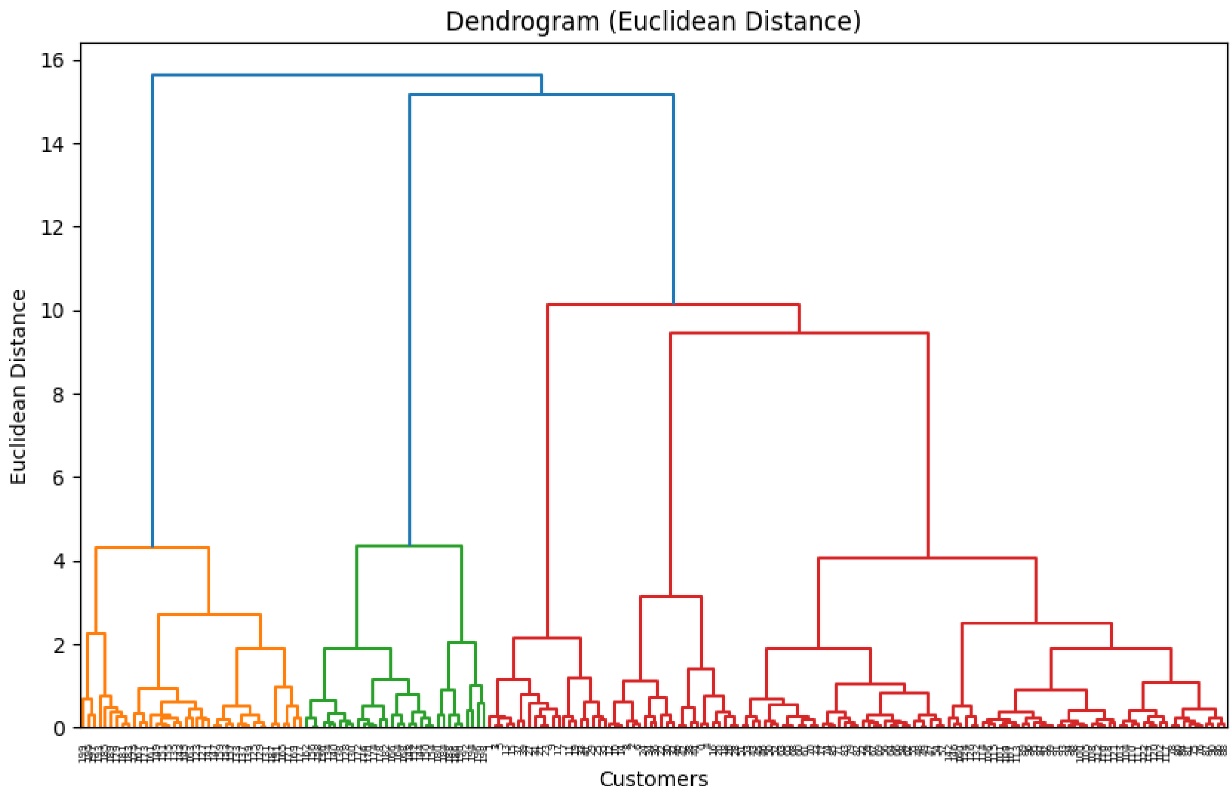
```python
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.spatial.distance import pdist, squareform
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import StandardScaler

# Load the dataset (Make sure to replace this path with the correct
path)
HarshArya_0098 = pd.read_csv('Mall_Customers.csv')

# Display the first few rows to understand the structure of the
dataset
print(HarshArya_0098.head())

# Check for missing values
print(HarshArya_0098.isnull().sum())

# Data Cleaning - Drop rows with missing values (if any)
HarshArya_0098 = HarshArya_0098.dropna()

# Select relevant columns - 'Annual Income' and 'Spending Score'
HarshArya_0098_dataset = HarshArya_0098[['Annual Income (k$)',
'Spending Score (1-100)']]

# Standardize the data before clustering
scaler_0098 = StandardScaler()
HarshArya_0098_dataset_scaled =
scaler_0098.fit_transform(HarshArya_0098_dataset)

# Statistical summary
print("Statistical Summary:\n")
print(HarshArya_0098_dataset.describe())

# Compute the distance matrix using Euclidean method
distance_matrix_euclidean_0098 = pdist(HarshArya_0098_dataset_scaled,
metric='euclidean')
distance_matrix_euclidean_0098 =
squareform(distance_matrix_euclidean_0098)
print("Distance Matrix (Euclidean):\n",
distance_matrix_euclidean_0098)

# Perform Hierarchical Clustering using Euclidean distance and ward.D
method
Z_euclidean_0098 = linkage(HarshArya_0098_dataset_scaled,
method='ward', metric='euclidean')

# Plotting the Dendrogram for Euclidean distance
plt.figure(figsize=(10, 6))
```

```
dendrogram(Z_euclidean_0098)
plt.title('Dendrogram (Euclidean Distance)')
plt.xlabel('Customers')
plt.ylabel('Euclidean Distance')
plt.show()

# Compute the distance matrix using Manhattan method
distance_matrix_manhattan_0098 = pdist(HarshArya_0098_dataset_scaled,
metric='cityblock')
distance_matrix_manhattan_0098 =
squareform(distance_matrix_manhattan_0098)
print("Distance Matrix (Manhattan):\n",
distance_matrix_manhattan_0098)

# Perform Hierarchical Clustering using Manhattan distance and average
method
Z_manhattan_0098 = linkage(HarshArya_0098_dataset_scaled,
method='average', metric='cityblock')

# Plotting the Dendrogram for Manhattan distance
plt.figure(figsize=(10, 6))
dendrogram(Z_manhattan_0098)
plt.title('Dendrogram (Manhattan Distance)')
plt.xlabel('Customers')
plt.ylabel('Manhattan Distance')
plt.show()

# Compute the distance matrix using Maximum method
distance_matrix_maximum_0098 = pdist(HarshArya_0098_dataset_scaled,
metric='chebyshev')
distance_matrix_maximum_0098 =
squareform(distance_matrix_maximum_0098)
print("Distance Matrix (Maximum):\n", distance_matrix_maximum_0098)

# Perform Hierarchical Clustering using Maximum distance and average
method
Z_maximum_0098 = linkage(HarshArya_0098_dataset_scaled,
method='average', metric='chebyshev')

# Plotting the Dendrogram for Maximum distance
plt.figure(figsize=(10, 6))
dendrogram(Z_maximum_0098)
plt.title('Dendrogram (Maximum Distance)')
plt.xlabel('Customers')
plt.ylabel('Maximum Distance')
plt.show()

# Compute the distance matrix using Canberra method
distance_matrix_canberra_0098 = pdist(HarshArya_0098_dataset_scaled,
metric='canberra')
```

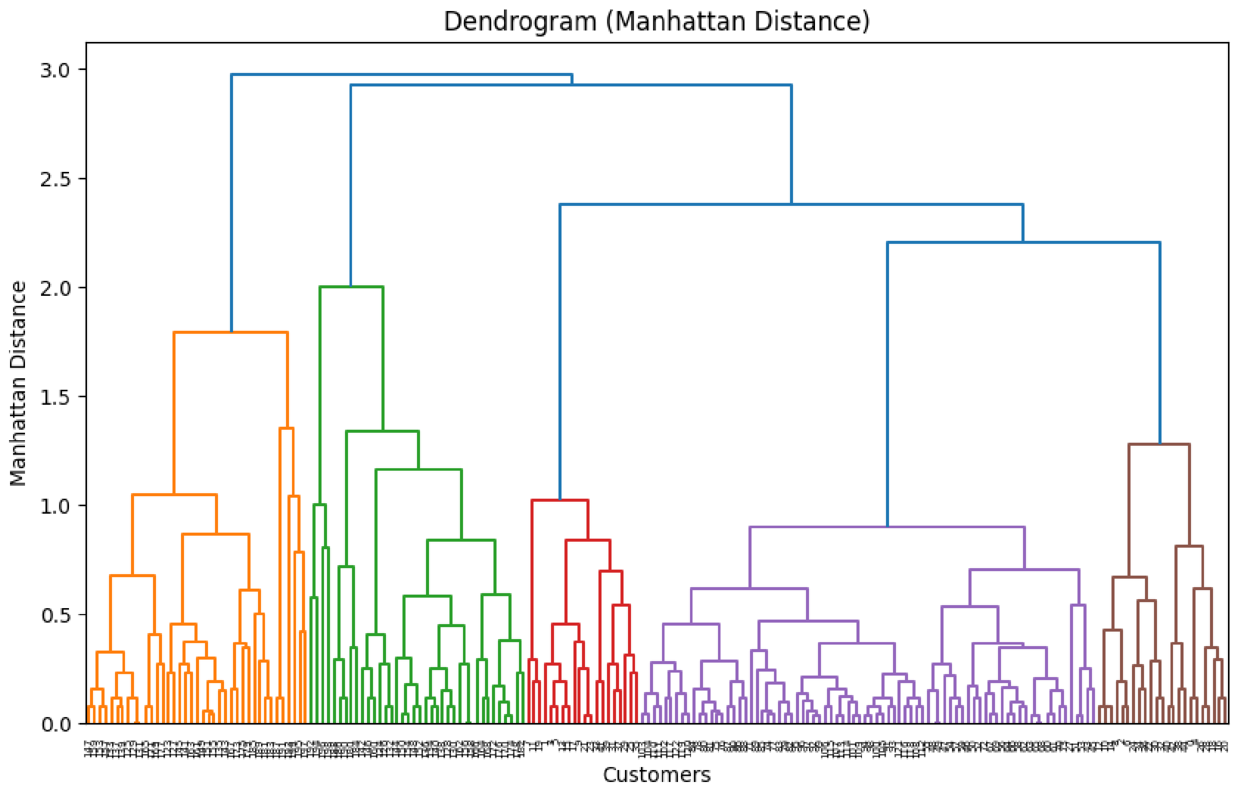
```python
distance_matrix_canberra_0098 =
squareform(distance_matrix_canberra_0098)
print("Distance Matrix (Canberra):\n", distance_matrix_canberra_0098)

# Perform Hierarchical Clustering using Canberra distance and average
method
Z_canberra_0098 = linkage(HarshArya_0098_dataset_scaled,
method='average', metric='canberra')

# Plotting the Dendrogram for Canberra distance
plt.figure(figsize=(10, 6))
dendrogram(Z_canberra_0098)
plt.title('Dendrogram (Canberra Distance)')
plt.xlabel('Customers')
plt.ylabel('Canberra Distance')
plt.show()

# Compute the distance matrix using Binary method
distance_matrix_binary_0098 = pdist(HarshArya_0098_dataset_scaled,
metric='jaccard')
distance_matrix_binary_0098 = squareform(distance_matrix_binary_0098)
print("Distance Matrix (Binary):\n", distance_matrix_binary_0098)

# Perform Hierarchical Clustering using Binary distance and average
method
Z_binary_0098 = linkage(HarshArya_0098_dataset_scaled,
method='average', metric='jaccard')

# Plotting the Dendrogram for Binary distance
plt.figure(figsize=(10, 6))
dendrogram(Z_binary_0098)
plt.title('Dendrogram (Binary Distance)')
plt.xlabel('Customers')
plt.ylabel('Binary Distance')
plt.show()

# Compute the distance matrix using Minkowski method (p=3, typical
choice for Minkowski distance)
distance_matrix_minkowski_0098 = pdist(HarshArya_0098_dataset_scaled,
metric='minkowski', p=3)
distance_matrix_minkowski_0098 =
squareform(distance_matrix_minkowski_0098)
print("Distance Matrix (Minkowski):\n",
distance_matrix_minkowski_0098)

# Perform Hierarchical Clustering using Minkowski distance and average
method
Z_minkowski_0098 = linkage(HarshArya_0098_dataset_scaled,
method='average', metric='minkowski')
```

```python
# Plotting the Dendrogram for Minkowski distance
plt.figure(figsize=(10, 6))
dendrogram(Z_minkowski_0098)
plt.title('Dendrogram (Minkowski Distance)')
plt.xlabel('Customers')
plt.ylabel('Minkowski Distance')
plt.show()
```

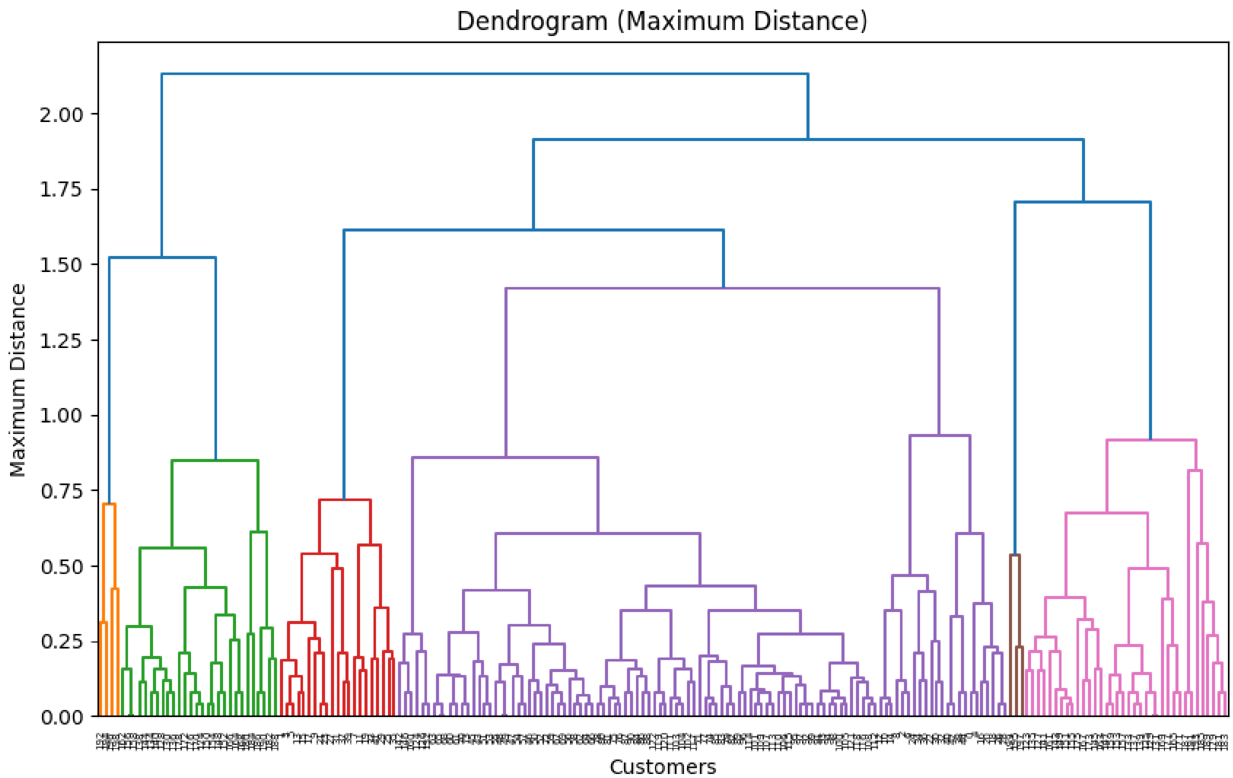
```
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
CustomerID                0
Genre                     0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
Statistical Summary:

       Annual Income (k$)  Spending Score (1-100)
count          200.000000              200.000000
mean            60.560000               50.200000
std             26.264721               25.823522
min             15.000000                1.000000
25%             41.500000               34.750000
50%             61.500000               50.000000
75%             78.000000               73.000000
max            137.000000               99.000000
Distance Matrix (Euclidean):
 [[0.         1.63050555 1.28167999 ... 4.44935328 4.72749573
4.96007568]
 [1.63050555 0.         2.91186723 ... 4.24551281 5.25987762
4.65731761]
 [1.28167999 2.91186723 0.         ... 4.95958139 4.64193658
5.50147501]
 ...
 [4.44935328 4.24551281 4.95958139 ... 0.         2.21418015
0.54622499]
 [4.72749573 5.25987762 4.64193658 ... 2.21418015 0.
2.52340145]
 [4.96007568 4.65731761 5.50147501 ... 0.54622499 2.52340145 0.
]]
```



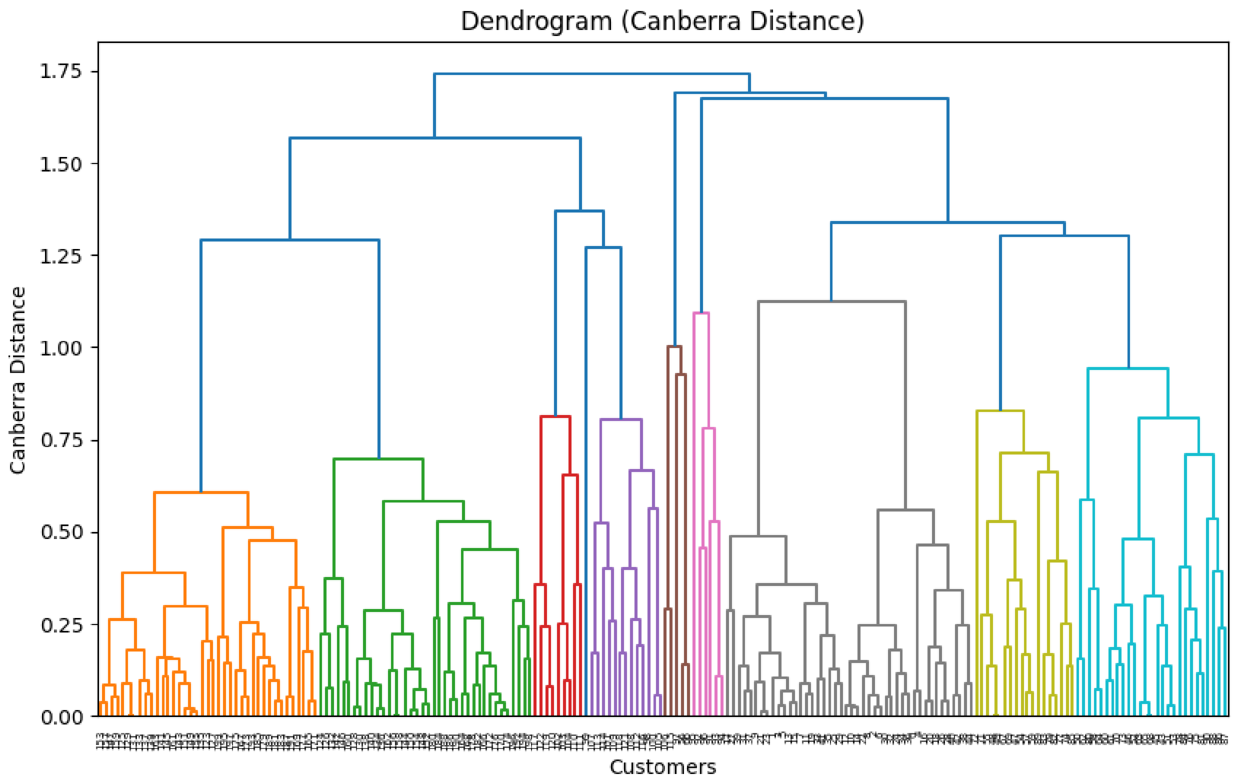

```
Distance Matrix (Manhattan):
 [[0.         1.63050555 1.31928093 ... 5.59556126 5.47192313
6.36481903]
 [1.63050555 0.         2.94978648 ... 4.50855756 7.10242868
4.73431348]
 [1.31928093 2.94978648 0.         ... 6.83850334 5.08435966 7.6077611
]
 ...
 [5.59556126 4.50855756 6.83850334 ... 0.         2.59387112
0.76925777]
 [5.47192313 7.10242868 5.08435966 ... 2.59387112 0.
2.52340145]
 [6.36481903 4.73431348 7.6077611  ... 0.76925777 2.52340145 0.
]]
```



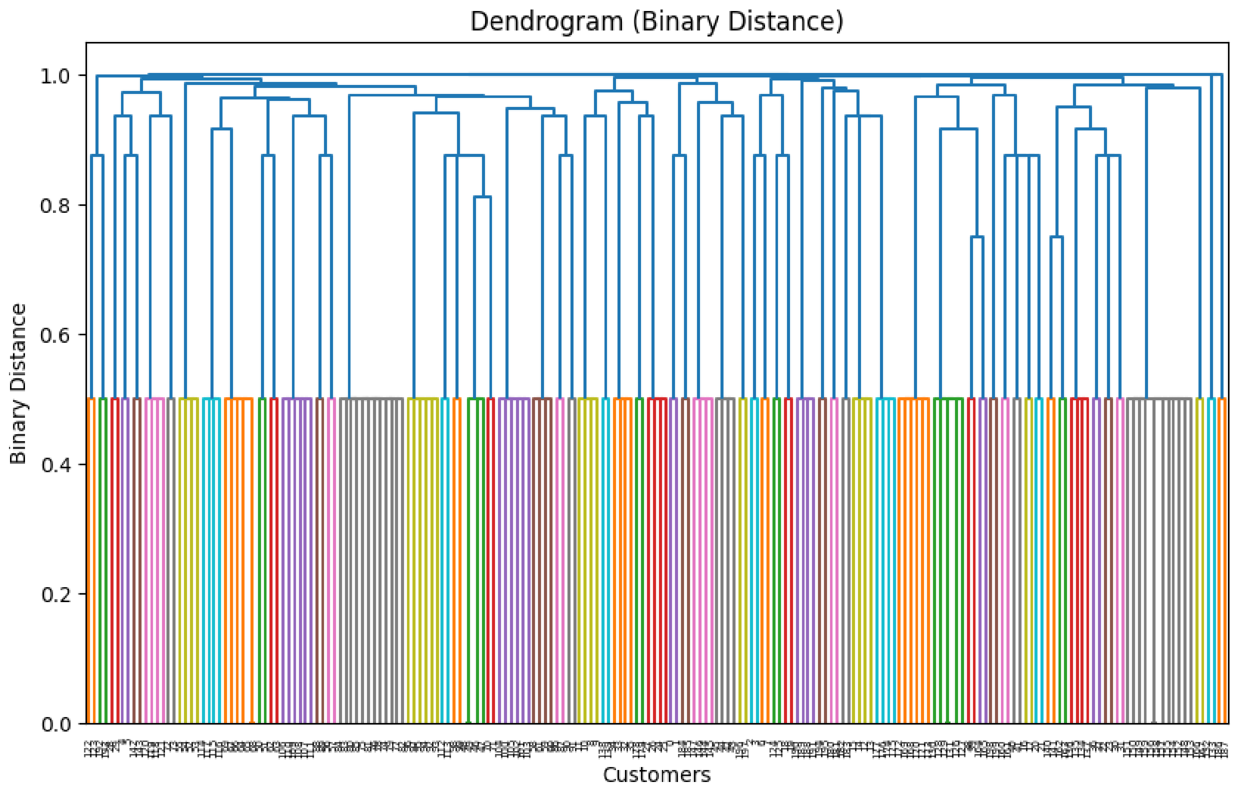

```
Distance Matrix (Maximum):
 [[0.         1.63050555 1.2811115  ... 4.23680664 4.65667036
4.65667036]
 [1.63050555 0.         2.91161705 ... 4.23680664 4.65667036
4.65667036]
 [1.2811115  2.91161705 0.         ... 4.19863721 4.61850093
4.61850093]
 ...
 [4.23680664 4.23680664 4.19863721 ... 0.         2.1740074
0.41986372]
 [4.65667036 4.65667036 4.61850093 ... 2.1740074  0.
2.52340145]
 [4.65667036 4.65667036 4.61850093 ... 0.41986372 2.52340145 0.
]]
```



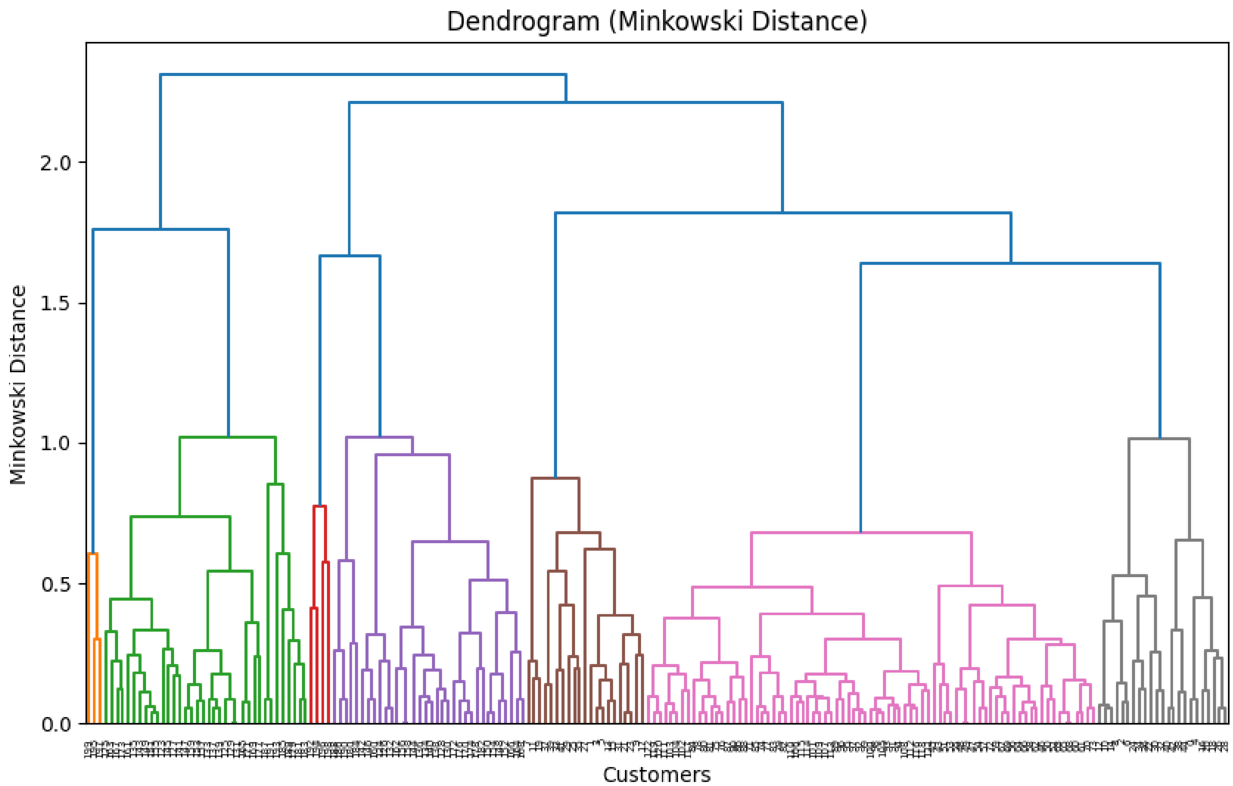

```
Distance Matrix (Canberra):
 [[0.         1.         0.60676419 ... 2.         1.48387097 2.
]
 [1.         0.         1.01109632 ... 1.12820513 2.
1.03144654]
 [0.60676419 1.01109632 0.         ... 2.         1.15706806 2.
]
 ...
 [2.         1.12820513 2.         ... 0.         1.07753031
0.23654091]
 [1.48387097 2.         1.15706806 ... 1.07753031 0.         1.
]
 [2.         1.03144654 2.         ... 0.23654091 1.         0.
]]
```




```
Distance Matrix (Binary):
 [[0.  0.5 1.  ... 1.  1.  1. ]
 [0.5 0.  1.  ... 1.  1.  1. ]
 [1.  1.  0.  ... 1.  1.  1. ]
 ...
 [1.  1.  1.  ... 0.  1.  1. ]
 [1.  1.  1.  ... 1.  0.  0.5]
 [1.  1.  1.  ... 1.  0.5 0. ]]
```




```
Distance Matrix (Minkowski):
 [[0.         1.63050555 1.2811228  ... 4.28288635 4.66498473
4.73205675]
 [1.63050555 0.         2.91161924 ... 4.23717927 4.87149647
4.65667755]
 [1.2811228  2.91161924 0.         ... 4.52110008 4.62008033
5.00301459]
 ...
 [4.28288635 4.23717927 4.52110008 ... 0.         2.17921505 0.4886351
]
 [4.66498473 4.87149647 4.62008033 ... 2.17921505 0.
2.52340145]
 [4.73205675 4.65667755 5.00301459 ... 0.4886351  2.52340145 0.
]]
```



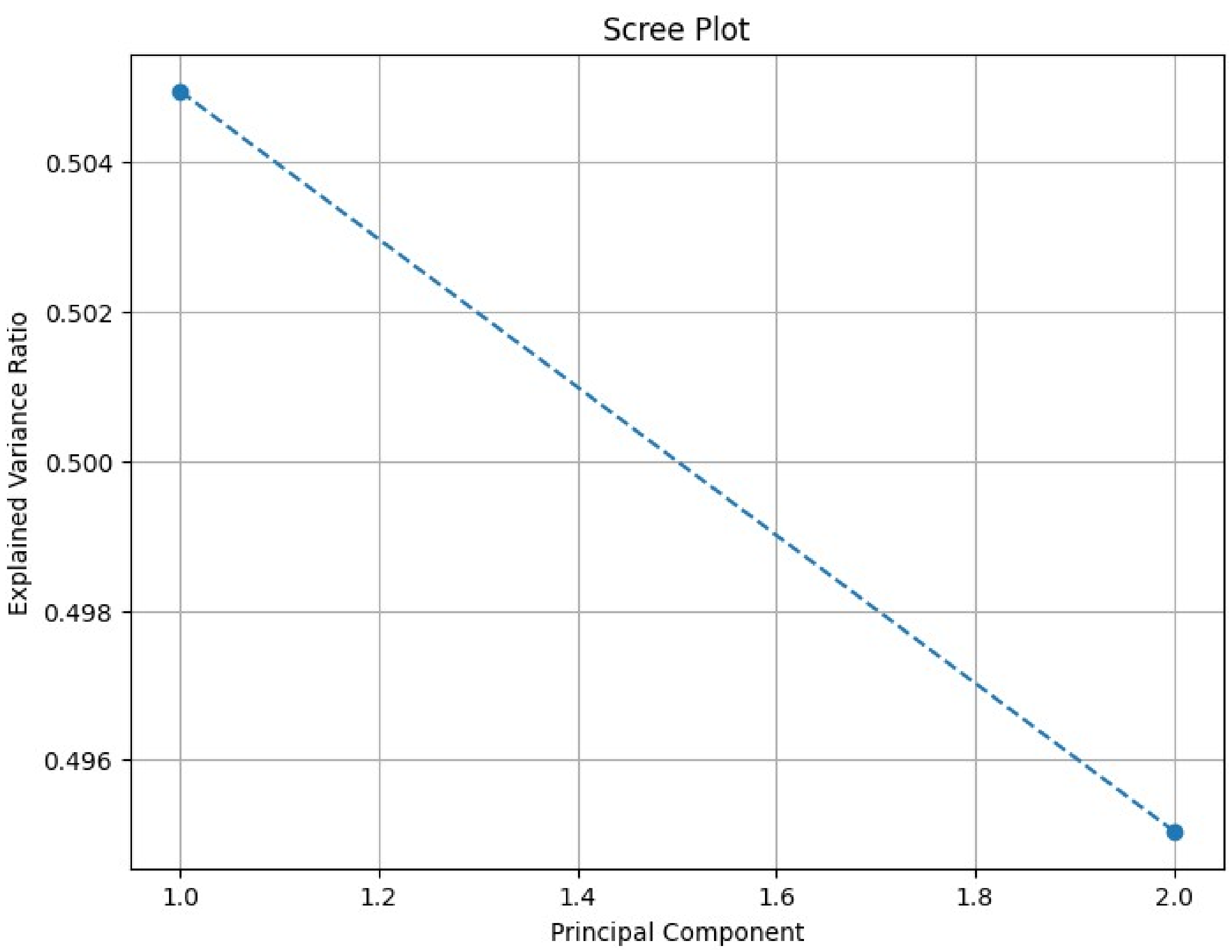
MODULE 6

```
# Step 1: Install and import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn import datasets

# Step 2: Load the dataset
# Replace 'mall_customer.csv' with the actual file path if needed
HarshArya_0098 = pd.read_csv("Mall_Customers.csv")

# Display the first few rows of the dataset to understand its
structure
print(HarshArya_0098.head())

# Assuming that the numeric columns are 'Annual Income (k$)' and
'Spending Score (1-100)'
# If column names are different, make sure to update the column names
accordingly.
```

```python
# Step 3: Data Preprocessing (Selecting only numeric columns for PCA)
# We assume 'Annual Income (k$)' and 'Spending Score (1-100)' are the
numeric columns.
data_0098 = HarshArya_0098[['Annual Income (k$)', 'Spending Score (1-
100)']]

# Step 4: Standardizing the data (important for PCA)
scaler_0098 = StandardScaler()
data_scaled_0098 = scaler_0098.fit_transform(data_0098)

# Step 5: Perform PCA
pca_0098 = PCA()
pca_result_0098 = pca_0098.fit_transform(data_scaled_0098)

# Step 6: Print PCA results - Eigenvalues (explained variance) and
loadings
print("\nEigenvalues (Explained Variance):")
print(pca_0098.explained_variance_)

print("\nExplained Variance Ratio (Percentage of variance explained by
each component):")
print(pca_0098.explained_variance_ratio_)

print("\nPCA Components (Loadings):")
print(pca_0098.components_)

# Step 7: Plot the explained variance ratio (Scree plot)
plt.figure(figsize=(8,6))
plt.plot(range(1, len(pca_0098.explained_variance_ratio_) + 1),
pca_0098.explained_variance_ratio_, marker='o', linestyle='--')
plt.title("Scree Plot")
plt.xlabel("Principal Component")
plt.ylabel("Explained Variance Ratio")
plt.grid(True)
plt.show()

# Step 8: Visualizing the first two principal components
plt.figure(figsize=(8,6))
sns.scatterplot(x=pca_result_0098[:, 0], y=pca_result_0098[:, 1],
palette='viridis')
plt.title("PCA - First Two Principal Components")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid(True)
plt.show()

# Step 9: Plot the first two components as a biplot (variables and
individuals)
plt.figure(figsize=(8, 6))
sns.scatterplot(x=pca_result_0098[:, 0], y=pca_result_0098[:, 1],
```

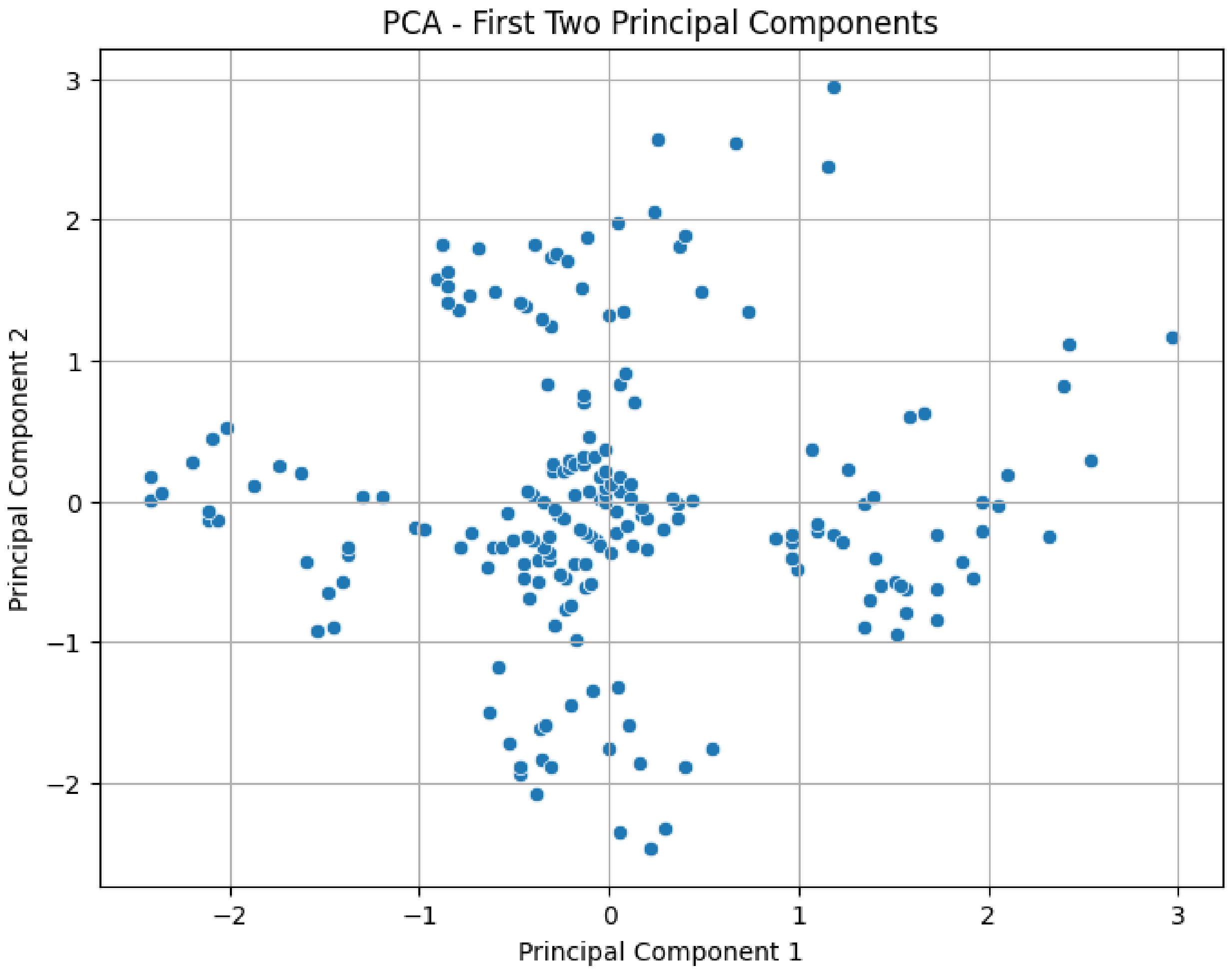
```
color='blue', label='Individuals')

# Plotting the components (loadings) on the same plot
for i, feature in enumerate(data_0098.columns):
    plt.arrow(0, 0, pca_0098.components_[0][i],
pca_0098.components_[1][i], color='red', alpha=0.5)
    plt.text(pca_0098.components_[0][i] * 1.2, pca_0098.components_[1]
[i] * 1.2, feature, color='red', ha='center', va='center')

plt.title("PCA - Biplot")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid(True)
plt.show()

# Step 10: Summary of PCA results
print("\nSummary of PCA:")
print("Explained Variance (Eigenvalues):",
pca_0098.explained_variance_)
print("Explained Variance Ratio:", pca_0098.explained_variance_ratio_)
print("Cumulative Explained Variance:",
np.cumsum(pca_0098.explained_variance_ratio_))

# Based on the Scree plot, decide how many components to keep (usually
the first few with eigenvalues > 1 or that explain most of the
variance).
# We can check how many components are required to explain, say 90% of
the variance.

cumulative_variance_0098 =
np.cumsum(pca_0098.explained_variance_ratio_)
print("Cumulative Variance Explained by Top Components:",
cumulative_variance_0098)

# Example: Selecting the first two components (if they explain most of
the variance)
n_comp_0098 = 2
pca_result_selected_0098 = pca_result_0098[:, :n_comp_0098]

# Visualize the selected PCA components
plt.figure(figsize=(8,6))
sns.scatterplot(x=pca_result_selected_0098[:, 0],
y=pca_result_selected_0098[:, 1], palette='viridis')
plt.title("Selected PCA - First Two Components")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid(True)
plt.show()
```

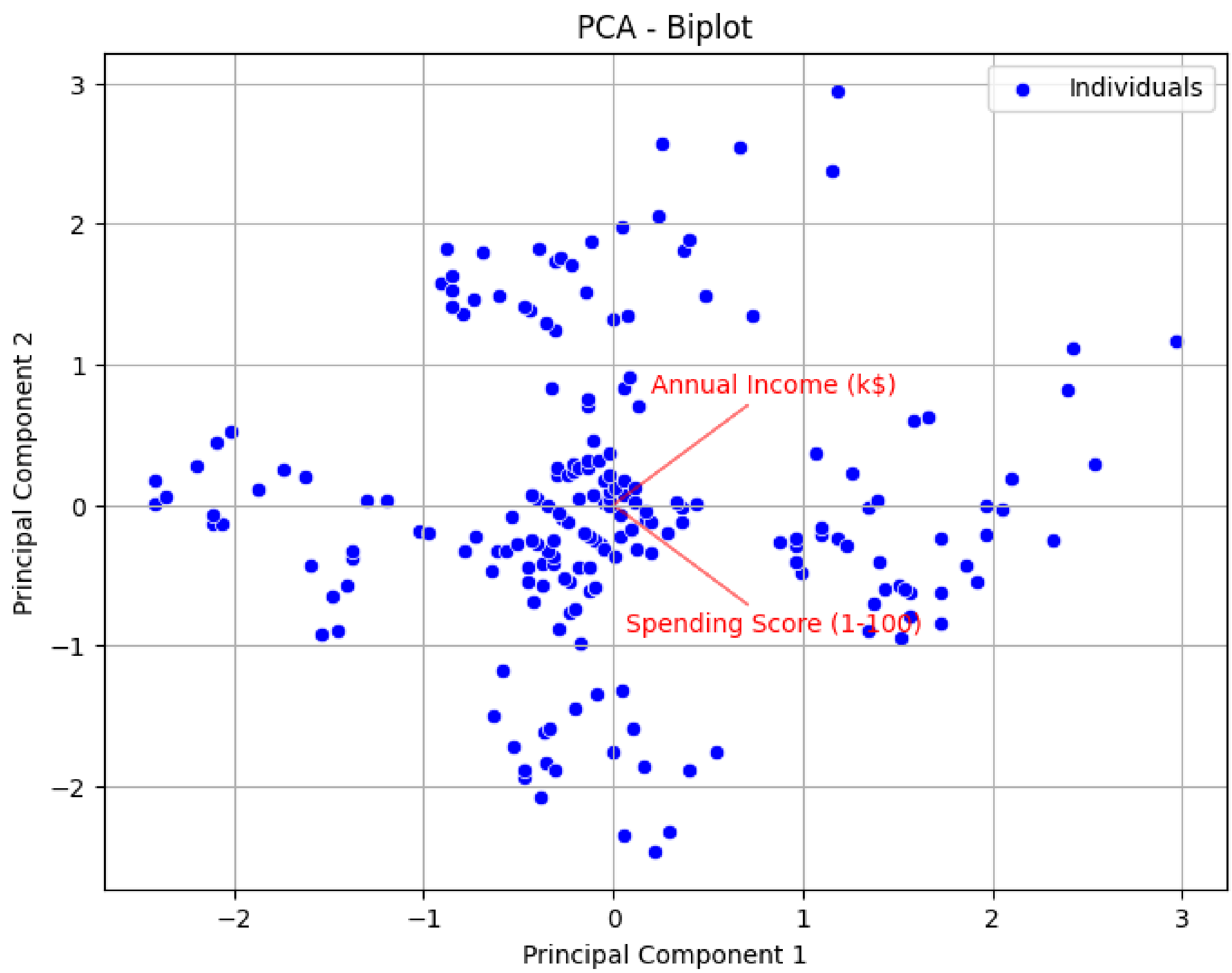
```
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40

Eigenvalues (Explained Variance):
[1.01497774 0.99507251]

Explained Variance Ratio (Percentage of variance explained by each
component):
[0.50495142 0.49504858]

PCA Components (Loadings):
[[ 0.70710678  0.70710678]
 [ 0.70710678 -0.70710678]]
```



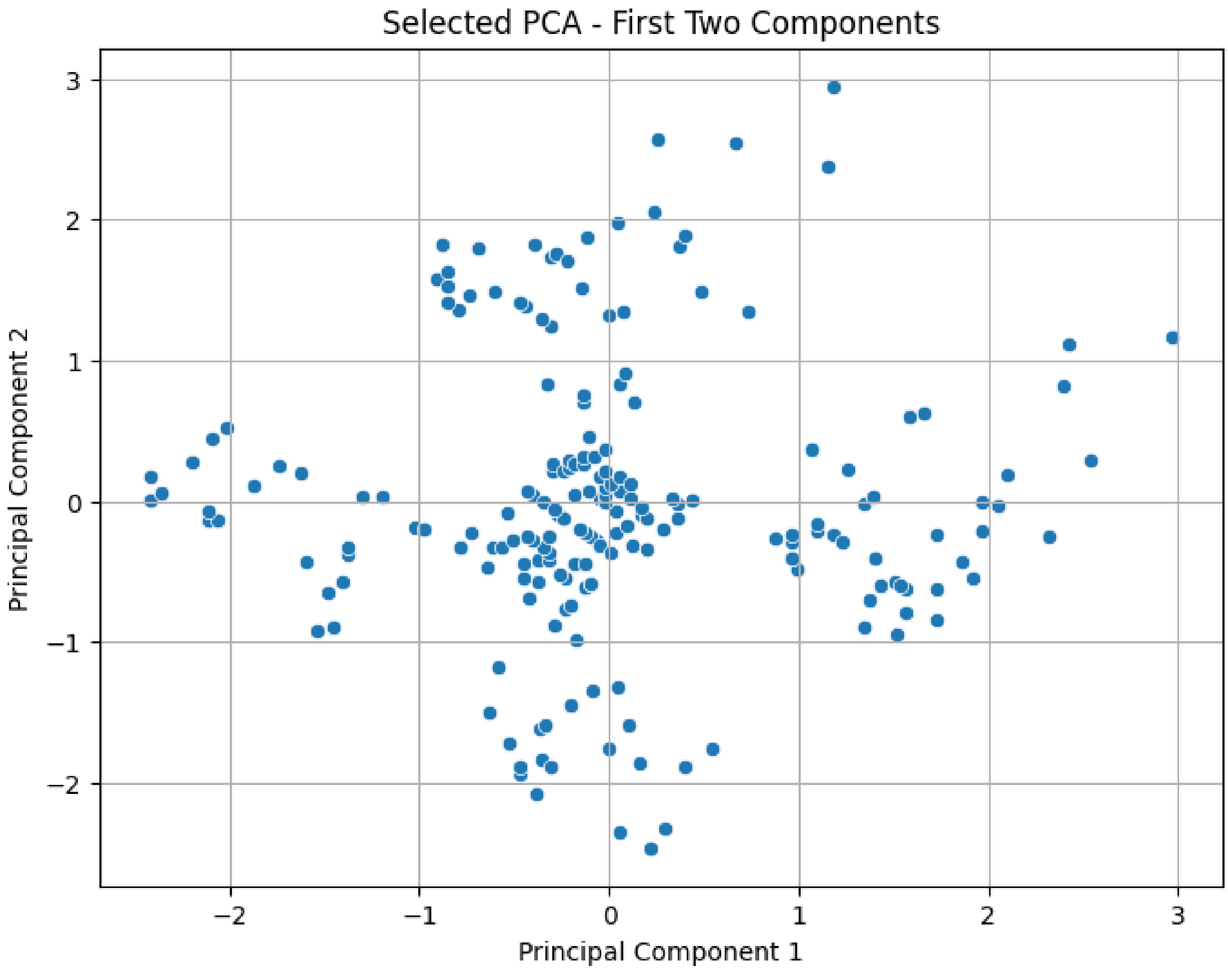

```
<ipython-input-104-7d2d184ace15>:53: UserWarning:
```

```
Ignoring `palette` because no `hue` variable has been assigned.
```






```
Summary of PCA:
Explained Variance (Eigenvalues): [1.01497774 0.99507251]
Explained Variance Ratio: [0.50495142 0.49504858]
Cumulative Explained Variance: [0.50495142 1.        ]
Cumulative Variance Explained by Top Components: [0.50495142 1.
]

<ipython-input-104-7d2d184ace15>:93: UserWarning:

Ignoring `palette` because no `hue` variable has been assigned.
```


Selected PCA - First Two Components
Principal Component 2
Principal Component 1
3
2
1
0
−1
−2
−2
−1
0
1
2
3