Title: Analysis and Predictive Modeling of Automobile Fuel Efficiency Implementing Machine Learning Techniques

Name: Suprem Shrestha
Student ID: 25123849
Module Title: Introduction to Artificial Intelligence
Module code: CMP4294
Date: 14th June, 2025

Table of Contents

Abstract

The project is a machine learning application to predict automobile fuel efficiency based on 398 cars of the 1970s-1980s. Following extensive preprocessing that involved filling in missing values and dealing with outliers, two regression models were considered. Random Forest Regressor performed better than the Linear Regression with R^2 of 0.91 as compared to 0.85. Correlation analysis depicted that there were strong negative correlations between MPG and weight (r=-0.83), displacement (r=-0.80), and horsepower (r=-0.77). The clustering technique(K-Means) revealed that there were three different vehicle groups and ANOVA indicated that there were significant differences in terms of fuel efficiency among the manufacturing origins where the Japanese vehicles demonstrated the best fuel efficiency. The results indicate that machine learning is effective in capturing the relationship that affects fuel efficiency and this information can be useful to manufacturers, consumers, and policymakers.

Keywords- Machine Learning, Fuel Efficiency Prediction, Random Forest Regressor, Automobile MPG, Correlation Analysis, K-Means Clustering, Regression Modeling, Vehicle, Specifications, Predictive Modeling.

1. Introduction

The increased fuel prices and environmental issues have caused fuel consumption of a vehicle to become an important element in the modern automobile industry(International Energy Agency (IEA), 2019). Among the factors determining the efficiency of a car is miles per gallon (mpg) that indicates the distance that a car can cover with a unit of fuel(Yoo et al., 2025a). More accurate predictions of mpg can aid in improved engineering, consumer decision making and regulatory decision making(Xie and Lin, 2017). This report presents a machine learning solution, regression-based predictive modeling of car mpg based on multiple features with a clear explanation of preprocessing, modeling, and evaluating the results(Schoen et al., 2019; Wickramanayake, 2015).

1.1. Problem Statement

The precise estimation of fuel efficiency of a vehicle is very crucial to the consumers, manufacturers as well as the policy makers. This project takes care of this problem with the help of data-driven approaches.

This analysis aims to tackle issues:

1. Address the challenge of accurately predicting automobile fuel efficiency.
2. Address the issue of missing and abnormal values in the automobiles datasets.
3. Determine the factors having the strongest influence on fuel efficiency.
4. Solve the grouping of automobiles to analyze similarities in their technical specifications.
5. Determining an effective method for predicting continuous automobile outcomes.

1.2. Objectives

The main goals this report attempts to achieve are:

- To develop a machine learning model that effectively predicts the fuel efficiency (mpg) of automobiles based on features.
- To apply effective data cleaning and preprocessing strategies for missing and anomalous values in the automobile dataset for high quality data.
- To identify and analyze the most significant factors influencing fuel efficiency through statistical methods.
- To sort automobiles into meaningful groups based on their specifications by applying clustering algorithms,
- To evaluate the performance of different regression models, such as Linear Regression and Random Forest, for predicting continuous automobile outcomes.

1.3. Domain Description

The automotive industry is a major industry in the world that influences the economic growth, sustainability and well-being of the consumers(Arbor, 2010). Using U.S., Japanese and European cars data, this report analyzes the effects of factors such as capacity, weight, horsepower and

country of origin on gas mileage (MPG) of cars. Using statistical and machine learning techniques reveals past design trends that can be applied to modern car design, policy-making, and emissions mitigation, given the increasing energy prices and tightening regulations(Kim and Koo, 2010).

2. Literature review

Automobile fuel efficiency has been analyzed and predictions made using a variety of machine learning approaches, and the linear regression model and random forest model are often favored as balanced between interpretability and prediction ability(Yoo et al., 2025). In the report, preprocessing of the data, especially the treatment of missing values and outliers, as reliable models depend on it is given more priority(ML | Handling Missing Values, 2024). Correlation analysis of features is commonly used to select important predictors (weight, horsepower, and displacement) that demonstrate a good correlation with mpg. Also, the clustering algorithm is broadly applied to split vehicles by technical specifications to enable more profound market and engineering analysis(Sharma, 2025).

3. Data Description

The dataset consists of 398 entries with 9 features each, capturing the vehicle's characteristics from the 1970s to 1980s. The features are:

- **Name:** Vehicle make and model (string).
- **MPG:** Fuel efficiency in miles per gallon (float)
- **Cylinders:** Number of cylinders in engine (integer).
- **Displacement:** Engine displacement in cubic inches (float).
- **Horsepower:** Power produced by engine (float).
- **Weight:** Weight of vehicle(integer).
- **Acceleration:** Accelerate from 0 to 60 mph in seconds (float).
- **Model Year:** Date of manufacture (integer).
- **Origin:** Manufacturing country(categorical: USA, Japan, Europe).

The dataset was loaded with the help of Pandas in Python, with an initial shape of (398, 9).

```
# Load the dataset
df = pd.read_csv('Automobile.csv')
```

Fig.1. Code snippet: Loading dataset into file.

Kaggle Link to dataset: Click here for Car information dataset link of Kaggle

3.1. Derived Variables

Two additional variables were created through text processing:

- Company Name: Extracted manufacturer name from the vehicle name
- Car Name: Extracted model name from the vehicle name

```
def company_name(x):
  if " " in x:
    return x[:x.index(" ")]
  else:
    return x

df["Company_name"] = df["name"].apply(company_name)
df
```

Fig.2. Code snippet: Extracting company name

	name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	Company_name
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5	70	usa	buick
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth
3	amc rebel sst	16.0	8	304.0	150.0	3433	12.0	70	usa	amc
4	ford torino	17.0	8	302.0	140.0	3449	10.5	70	usa	ford
...	...	...	...	...	...	...	...	...	...	...
393	ford mustang gl	27.0	4	140.0	86.0	2790	15.6	82	usa	ford
394	vw pickup	44.0	4	97.0	52.0	2130	24.6	82	europe	vw
395	dodge rampage	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge
396	ford ranger	28.0	4	120.0	79.0	2625	18.6	82	usa	ford
397	chevy s-10	31.0	4	119.0	82.0	2720	19.4	82	usa	chevy

Fig.3. Output of company name

```
def car_name(x):
  if " " in x:
    return x[x.index(" ")+1:]
  else:
    return x

df["car_name"] = df["name"].apply(car_name)
df
```

Fig.4. Code snippet: Extracting car name

	name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	Company_name	car_name
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet	chevelle malibu
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5	70	usa	buick	skylark 320
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth	satellite
3	amc rebel sst	16.0	8	304.0	150.0	3433	12.0	70	usa	amc	rebel sst
4	ford torino	17.0	8	302.0	140.0	3449	10.5	70	usa	ford	torino
...	...	...	...	...	...	...	...	...	...	...	...
393	ford mustang gl	27.0	4	140.0	86.0	2790	15.6	82	usa	ford	mustang gl
394	vw pickup	44.0	4	97.0	52.0	2130	24.6	82	europe	vw	pickup
395	dodge rampage	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge	rampage
396	ford ranger	28.0	4	120.0	79.0	2625	18.6	82	usa	ford	ranger
397	chevy s-10	31.0	4	119.0	82.0	2720	19.4	82	usa	chevy	s-10

Fig.5. Output of car name

4. Data Preprocessing

4.1. Missing Value Analysis:

The missing values of 6 for horsepower were imputed by replacing them with the median of horsepower to minimize the effect of outliers since median is less sensitive to skewness.

```
# Check for missing values
print("\nMissing values per column:")
print(df.isnull().sum())
```

```
Missing values per column:
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
Company_name    0
car_name        0
dtype: int64
```

Fig.6. Code snippet: Check for missing values

```
# Handle missing values (simple imputation)
df['horsepower'] = df['horsepower'].replace('?', np.nan).astype(float)
df['horsepower'].fillna(df['horsepower'].median(), inplace=True)
```

Fig.7. Code snippet: Treating missing values

4.2. Data Cleaning:

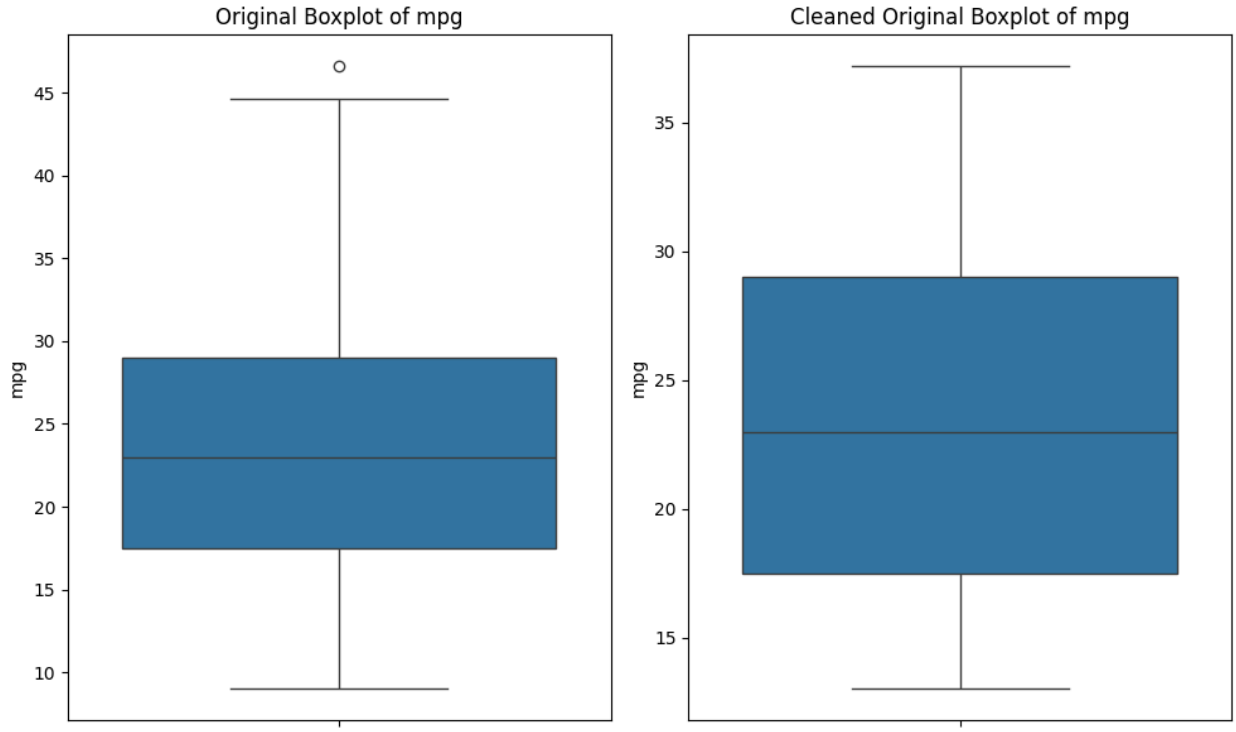
The categorical feature ‘origin’ was encoded with LabelEncoder to transform the values (USA, Japan, Europe) into numerical labels (0, 1, 2). There were no duplicate records and irrelevant attributes, thus no additional cleaning was performed.

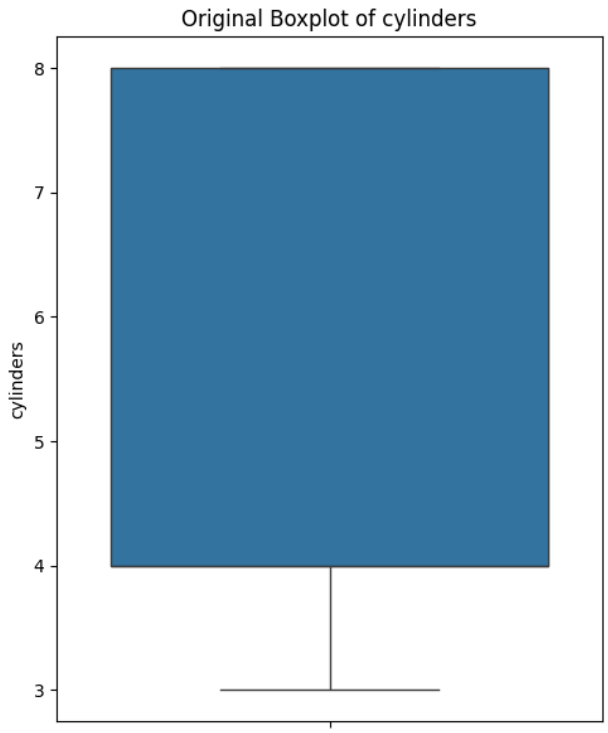
```
le = LabelEncoder()
X['origin'] = le.fit_transform(df_cleaned['origin'])
```

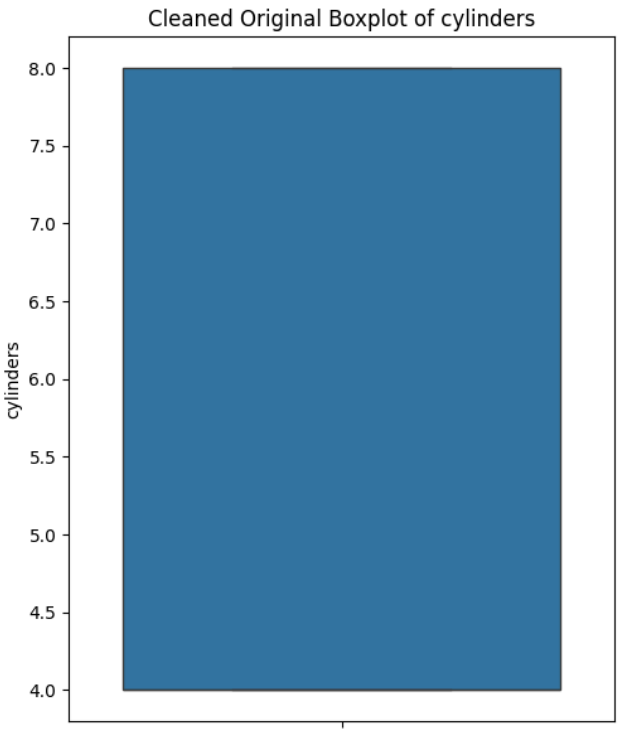
Fig.8. Code snippet: Encode origin

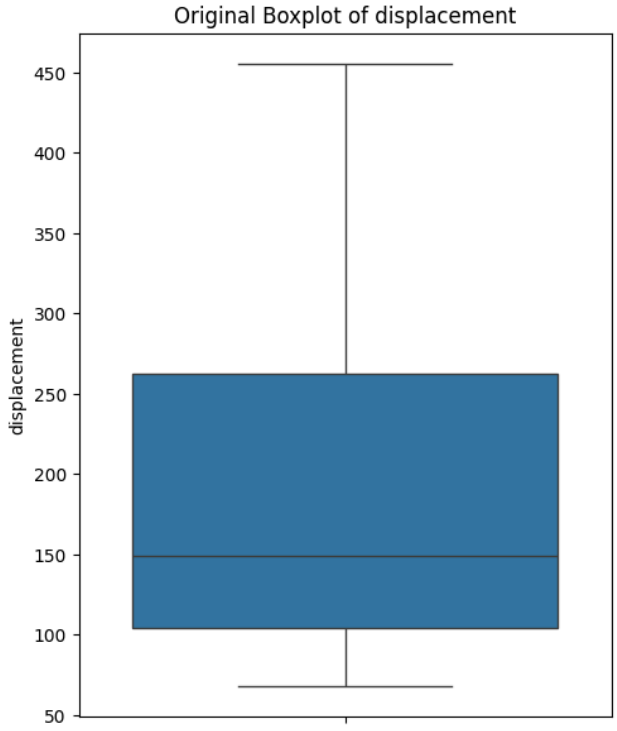
4.3. Outlier Identification and Management**:**

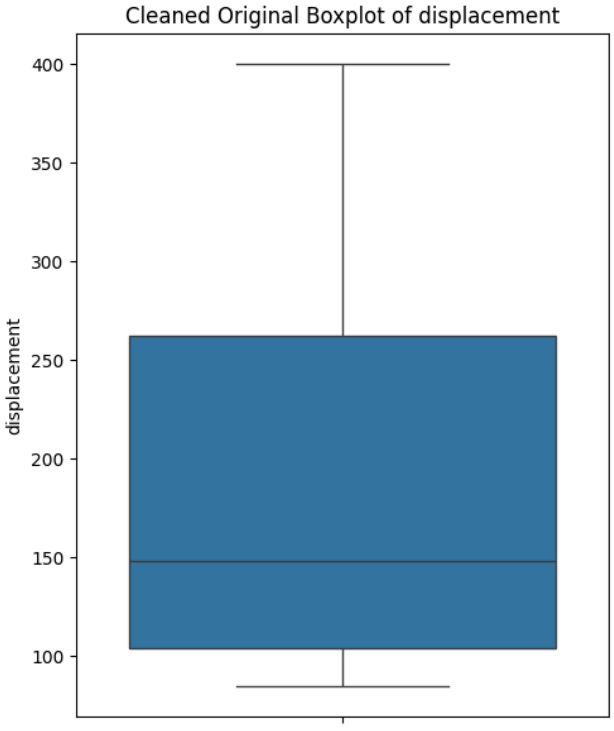
All the continuous variables were analyzed with the box plot visualization to determine outliers. Winsorization was used to limit the extreme values to the 5 th and 95 th percentile instead of eliminating outliers, which might lead to loss of information.

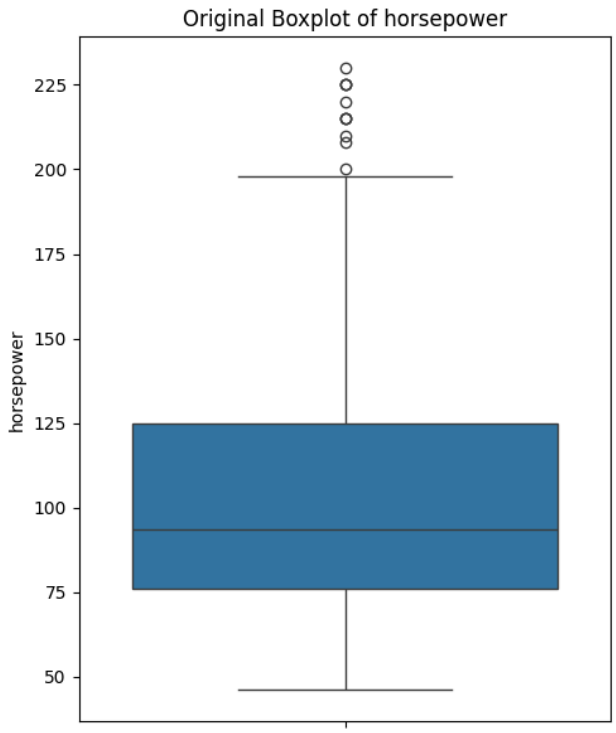


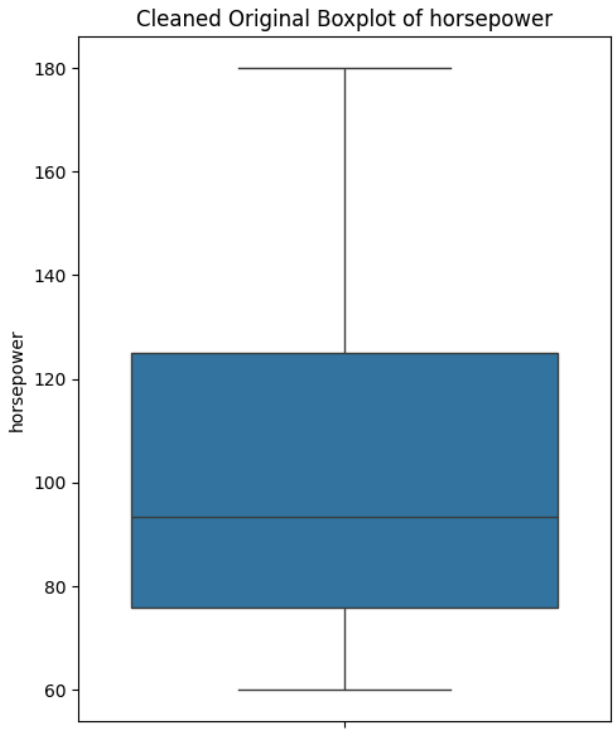

Original Boxplot of cylinders
cylinders
8
7
6
5
4
3

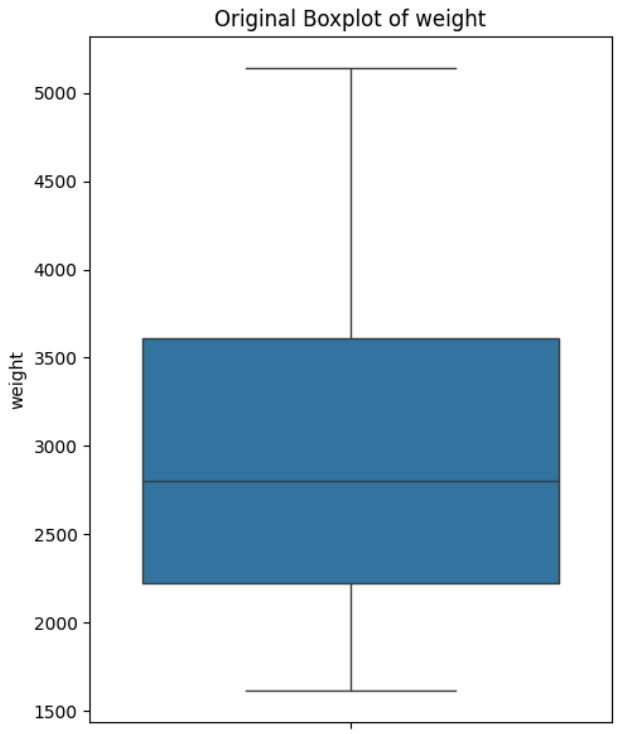

Cleaned Original Boxplot of cylinders
cylinders
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0

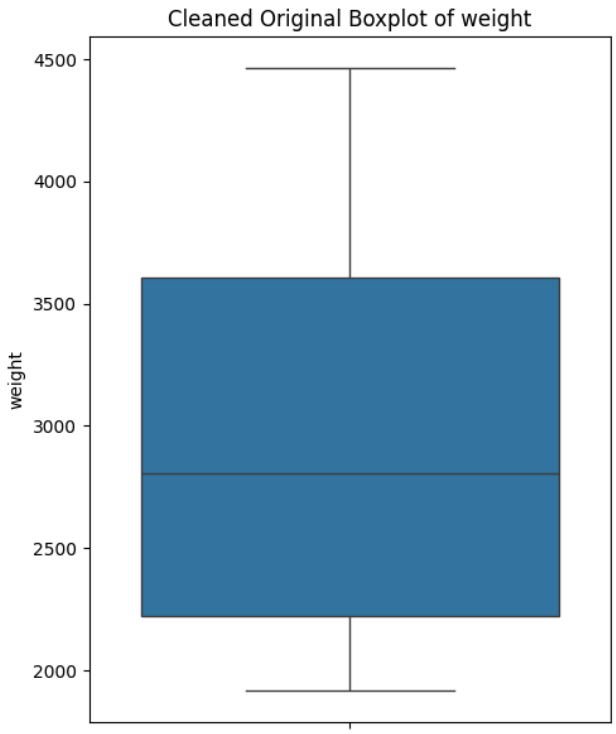

Original Boxplot of displacement
displacement
450
400
350
300
250
200
150
100
50


Cleaned Original Boxplot of displacement
displacement
400
350
300
250
200
150
100


Original Boxplot of horsepower
horsepower
225
200
175
150
125
100
75
50


Cleaned Original Boxplot of horsepower
horsepower
180
160
140
120
100
80
60


Original Boxplot of weight
weight
5000
4500
4000
3500
3000
2500
2000
1500


Cleaned Original Boxplot of weight
weight
4500
4000
3500
3000
2500
2000

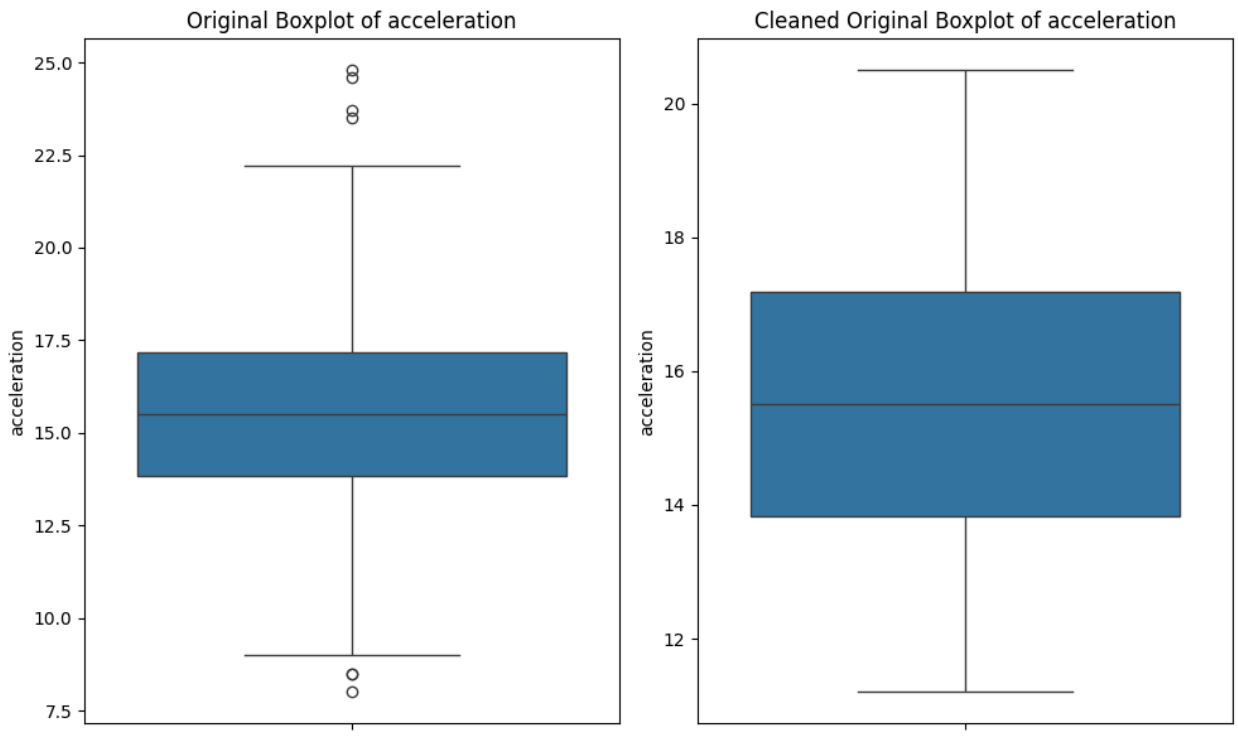


Fig.9. Visualization of original and cleaned boxplot

Winsorization Application: The winsorization procedure shortened the extreme values and did not reduce the size of the dataset, thus no observations were lost. This method preserves the distributional properties but down weights the effect of the extreme outliers.

```python
def winsorize_outliers(df, cols, limits=(0.05, 0.05)):
    df_winsorized = df.copy()

    print(f"Applying winsorization with limits: {limits}")

    for col in cols:
        # Store original values for comparison
        original_values = df_winsorized[col].copy()

        # Apply winsorization
        df_winsorized[col] = winsorize(original_values, limits=limits)

        # Count how many values were changed
        values_changed = (original_values != df_winsorized[col]).sum()

        # Show statistics
        print(f"\n{col}:")
        print(f"  Original range: [{original_values.min():.2f}, {original_values.max():.2f}]")
        print(f"  Winsorized range: [{df_winsorized[col].min():.2f}, {df_winsorized[col].max():.2f}]")
        print(f"  Values changed: {values_changed}")
        print(f"  Percentiles used: {original_values.quantile(limits[0]):.2f} to {original_values.quantile(1-limits[1]):.2f}")

    return df_winsorized

# Usage example
numeric_cols = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration']

# Apply winsorization
df_cleaned = winsorize_outliers(df, numeric_cols, limits=(0.05, 0.05))
```

Fig.10. Code snippet: Handling outliers

4.3. Feature Scaling:

The numerical attributes (cylinders, displacement, horsepower, weight, acceleration, model year) are normalized by applying StandardScaler to bring their range to an optimal suitability for algorithms such as K-Means and Linear Regression.

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_cluster)
```

Fig.11. Code snippet: Implementation of StandardScaler

5. Exploratory Data Analysis

5.1. Descriptive Analysis

Descriptive analysis has been used to describe the central tendencies, variation and distribution of important variables in the automobile data. This analysis allowed delivering a clear description of the characteristics of the whole dataset and identifying patterns or abnormalities within the data by calculating such measures as mean, median, standard deviation, and range.

```
print("\nDescriptive statistics:")
df.describe()
```

```
Descriptive statistics:
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.010050
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.697627
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.000000
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.000000
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000

Fig.12. Code snippet: Implementation of Descriptive Analysis

5.2. Correlation Analysis

Correlation analysis checks the magnitude and the direction of the associations between numerical characteristics. With correlation coefficients, the relationship of variables such as weight, displacement, and horsepower with mpg is determined. This grasped features which has the strongest effect on the fuel efficiency as well as informed the choice of significant predictors to be used in subsequent modeling.

```
# Visualize correlations
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

Fig.13. Code snippet: Implementation of CorrelationAnalysis

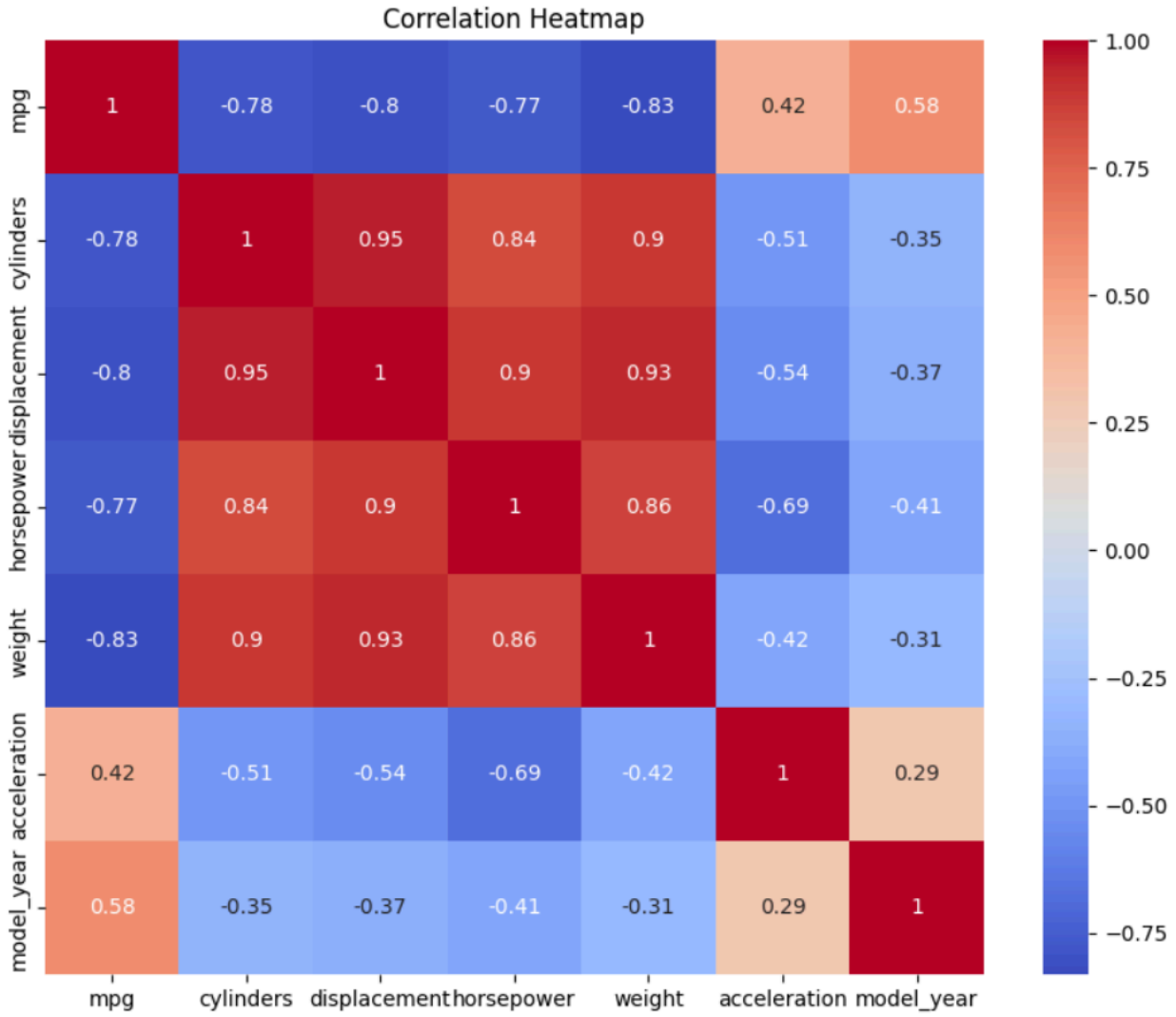


Fig.14. Output of implementation of Correlation Analysis

5.3. Distribution Analysis

Distribution analysis was done by creating histograms of all important numeric variables namely mpg, cylinders, displacement, horsepower, weight and acceleration. This graphic method assisted

in the unveiling of the form, dispersion, and skew of the distributions of every variable, showing tendencies, unusual values, and the general variability existing in the cars dataset.

```
print("\nDistribution Analysis")
# Multiple histograms for all numeric variables
numeric_cols = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration']

plt.figure(figsize=(15, 10))
for i, col in enumerate(numeric_cols, 1):
    plt.subplot(2, 3, i)
    plt.hist(df[col], bins=20, alpha=0.7, edgecolor='black')
    plt.title(f'Distribution of {col.title()}')
    plt.xlabel(col.title())
    plt.ylabel('Frequency')
    plt.grid(True, alpha=0.3)

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

Fig.15. Code snippet: Implementation of Distribution Analysis

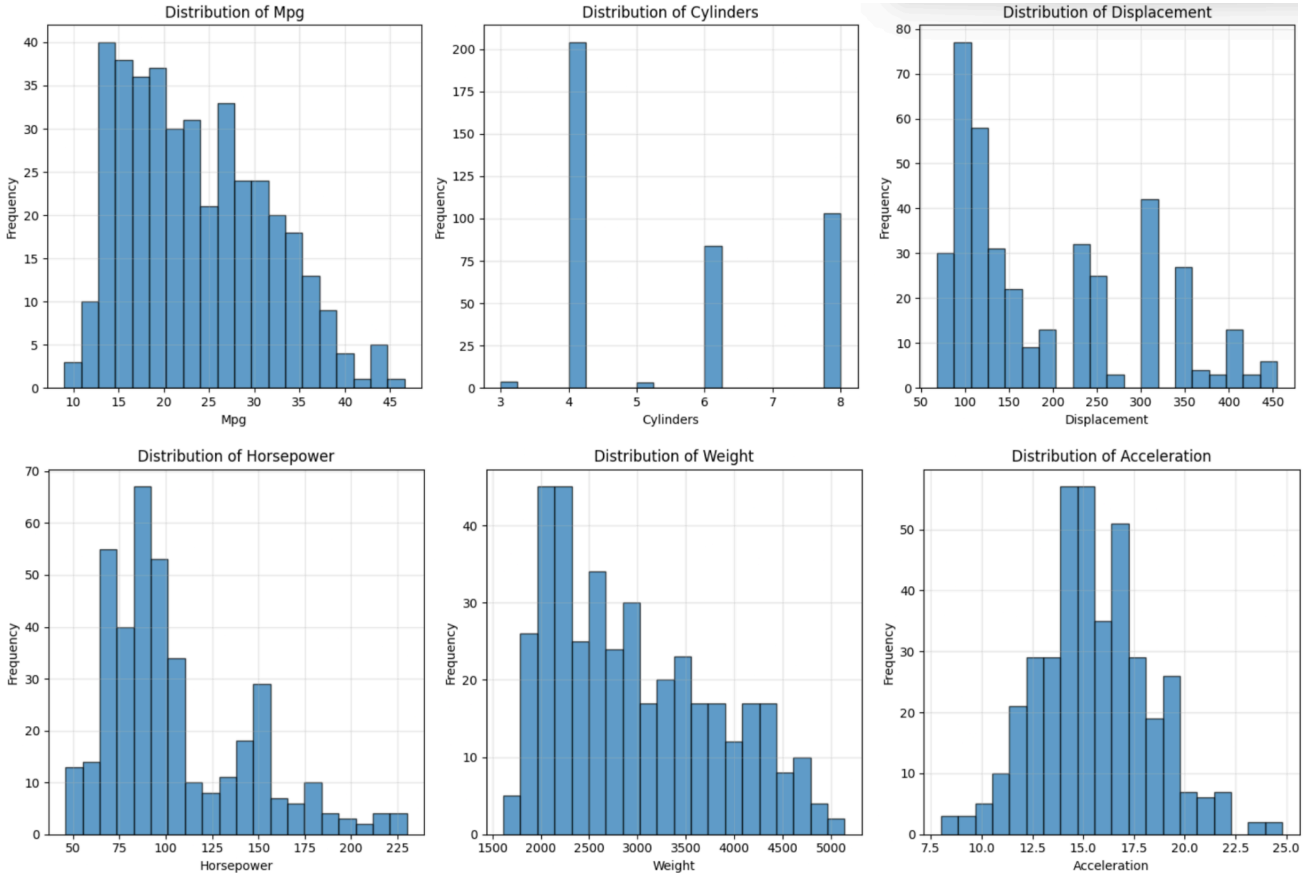


Fig.16. Output of Distribution Analysis

6. Statistical Analysis

6.1 Hypothesis Testing

To determine the existence of statistically significant differences in means of fuel efficiency between cars of various origins, Anova was used. The outcome significantly rejects the null hypothesis ($p < 0.001$), meaning country of origin influences automobile fuel efficiency in the data sample quantifiably.

```
# ANOVA test
f_stat, p_value = stats.f_oneway(usa_mpg, japan_mpg, europe_mpg)
print(f"\nANOVA test for mpg across origins: F-statistic={f_stat:.2f}, p-value={p_value:.4f}")
```

```
ANOVA test for mpg across origins: F-statistic=98.54, p-value=0.0000
```

Fig.17. Code snippet: Implementation of ANOVA test

6.2. Post-Hoc Analysis:

Post-hoc Tukey tests were performed as a result of the ANOVA outcome to make pairwise comparisons. There was a significant difference in all pairwise comparisons with Japanese cars registering the best fuel efficiency, European cars coming next and American cars registering the lowest fuel efficiency.

```
# Performing post-hoc tests
if p_value < 0.05:
    print("\nPost-hoc Tukey tests:")
    from statsmodels.stats.multicomp import pairwise_tukeyhsd
    tukey = pairwise_tukeyhsd(endog=df['mpg'], groups=df['origin'], alpha=0.05)
    print(tukey.summary())
```

```
Post-hoc Tukey tests:
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
group1 group2 meandiff p-adj   lower    upper  reject
----------------------------------------------------
europe  japan   2.5592 0.0404   0.0877  5.0307   True
europe    usa  -7.8079    0.0  -9.8448  -5.771   True
 japan    usa -10.3671    0.0 -12.3114 -8.4228   True
----------------------------------------------------
```

Fig.18. Code snippet: Implementation of Post-Hoc test

7. Model Training

7.1. Data Splitting:

Training set: 80% (318 observations)
Test set: 20% (80 observations)
Random state: 55

To estimate MPG, the following supervised learning techniques were applied:

7.2. Linear Regression:

A baseline predictive model of automobile fuel efficiency (mpg) was developed based on features including cylinders, displacement, horsepower and weight using Linear Regression. This was used due to its simple, interpretable, and models linear relationships well, enabling easy analysis of the effects that each variable has on mpg, and it also has easily interpretable results that can be used as a comparison.

```
# Prepare data for mpg prediction
X = df_cleaned[['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year']].copy()
y = df_cleaned['mpg']

# Encode origin if needed
le = LabelEncoder()
X['origin'] = le.fit_transform(df_cleaned['origin'])

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=55)

# Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
```

Fig.19. Code snippet: Implementation of Linear Regression

7.3. Random Forest regressor:

Improving prediction accuracy and deal complex, nonlinear relationships between automobile features and mpg was done using Random Forest Regressor . This method combines multiple decision trees to reduce overfitting and increase strength against outliers. It can capture complex interactions among variables and typically hold higher predictive performance on real-world datasets.

```python
# Prepare data for mpg prediction
X = df_cleaned[['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year']].copy()
y = df_cleaned['mpg']

# Encode origin if needed
le = LabelEncoder()
X['origin'] = le.fit_transform(df_cleaned['origin'])

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=55)

# Random Forest
rf = RandomForestRegressor(n_estimators=100, random_state=55)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

Fig.20. Code snippet: Implementation of Random Forest regressor

7.4. Model Analysis

Comparing the performance of Linear Regression and Random Forest Regression by:

```python
# Evaluate models
print("\nLinear Regression Performance:")
print(f"MSE: {mean_squared_error(y_test, y_pred_lr):.2f}")
print(f"R²: {r2_score(y_test, y_pred_lr):.2f}")

print("\nRandom Forest Performance:")
print(f"MSE: {mean_squared_error(y_test, y_pred_rf):.2f}")
print(f"R²: {r2_score(y_test, y_pred_rf):.2f}")

# Feature importance
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': rf.feature_importances_
}).sort_values('Importance', ascending=False)

print("\nFeature Importance:")
print(feature_importance)
```

```
Linear Regression Performance:
MSE: 9.58
R²: 0.84

Random Forest Performance:
MSE: 5.60
R²: 0.91

Feature Importance:
        Feature  Importance
1  displacement    0.364912
3        weight    0.222373
5    model_year    0.131846
0     cylinders    0.130022
2    horsepower    0.122552
4  acceleration    0.022310
6        origin    0.005986
```

Fig.21. Code snippet: Evaluating models

```python
# Visualize predictions vs actual
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_rf, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--')
plt.xlabel('Actual MPG')
plt.ylabel('Predicted MPG')
plt.title('Actual vs Predicted MPG (Random Forest)')
plt.show()
```

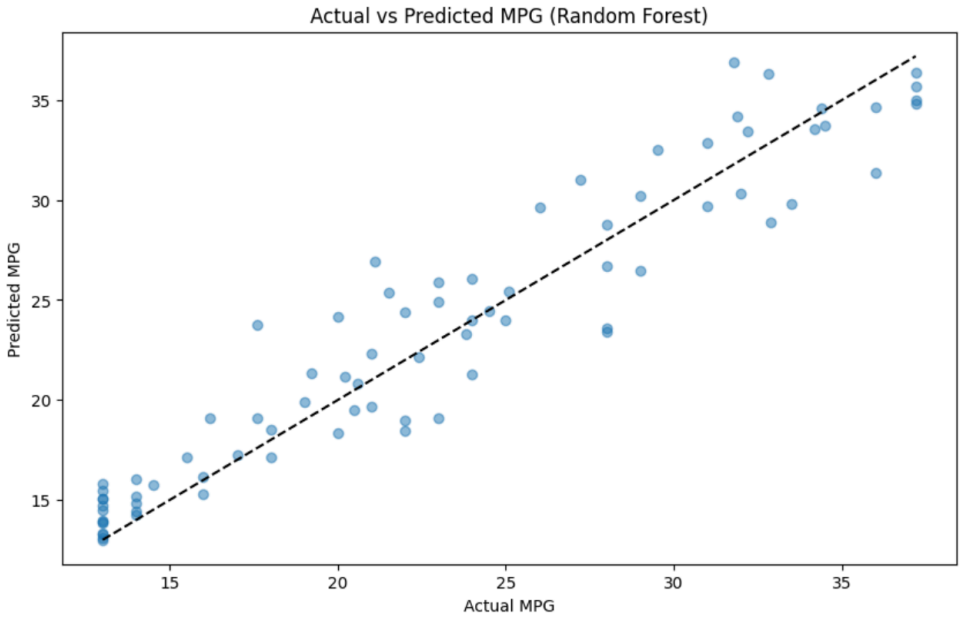


Fig.22. Evaluating actual vs predicted MPG

```
# Visualize predictions vs actual
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_lr, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--')
plt.xlabel('Actual MPG')
plt.ylabel('Predicted MPG')
plt.title('Actual vs Predicted MPG (Linear Regression)')
plt.show()
```

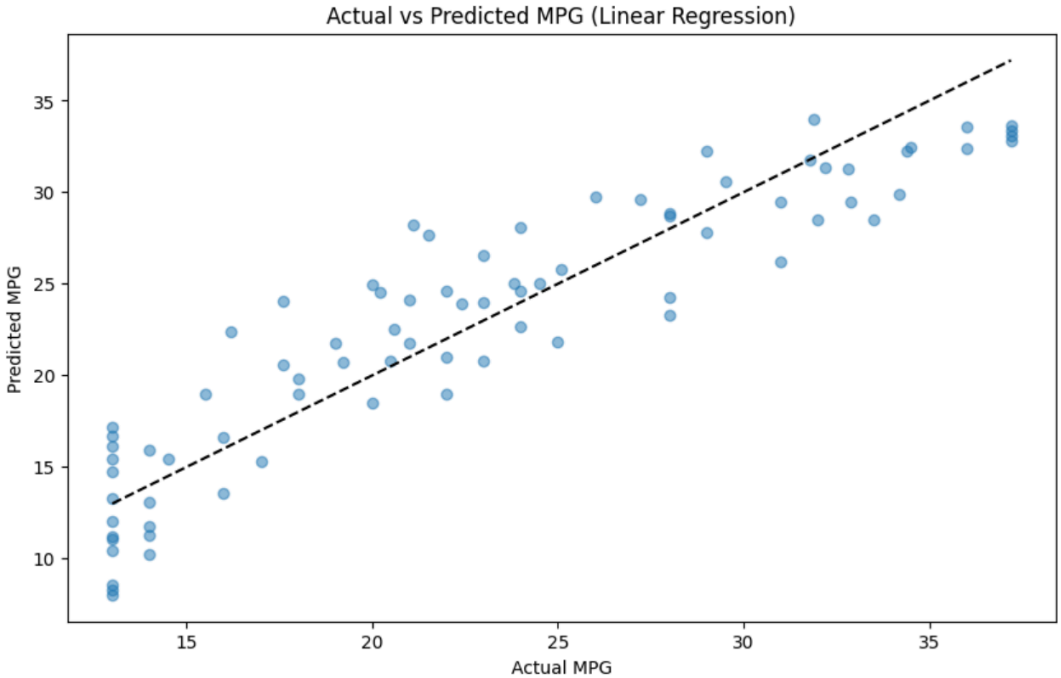


Fig.23. Evaluating actual vs predicted MPG

8. Cluster Analysis

The clustering was done with KMeans algorithm on the standardized automobile features. Unsupervised learning method was selected to cluster vehicles by similarities in their specifications to identify the patterns and segments in the data to be further explored. The elbow method was used to find the optimum number of clusters. The analysis indicated k=3 would provide the best number of clusters, which determines vehicle segments.

```
# Prepare data for clustering
cluster_features = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight']
X_cluster = df[cluster_features]

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_cluster)

# Determine optimal number of clusters using elbow method
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=55)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(range(1,11), inertia, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```

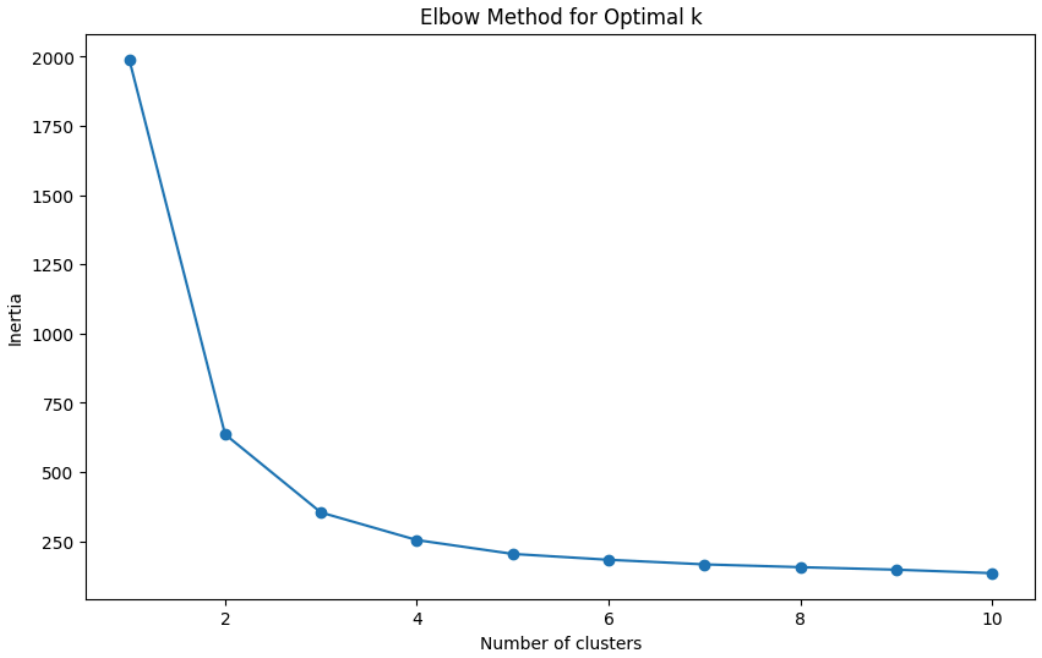


Fig.24. Code snippet: Implementation of Elbow Method

```
# Perform clustering with selected k
k = 3
kmeans = KMeans(n_clusters=k, random_state=55)
clusters = kmeans.fit_predict(X_scaled)

# Add clusters to dataframe
df['cluster'] = clusters

# Analyze clusters
print("\nCluster characteristics:")
print(df.groupby('cluster')[cluster_features].mean())
```

```
Cluster characteristics:
                mpg  cylinders  displacement  horsepower       weight
cluster
0         19.806522   5.978261    215.815217  101.809783  3204.217391
1         29.400000   4.004831    109.780193   78.533816  2306.067633
2         14.654545   8.000000    347.515152  160.505051  4142.272727
```

Fig.25. Code snippet: Implementation of Clustering

7. Results

7.1. Exploratory Data Analysis

There are 398 vehicles having an MPG range of 9.0 to 46.6 in this dataset.
Important highlights are:

- Very strong negative relationships between MPG and weight, displacement, and horsepower with relations of (r=−0.83) fuel efficiency versus weight, (r=−0.80) fuel efficiency versus displacement, (r=−0.77) fuel efficiency versus horsepower, explaining that greater and more powerful vehicles tend to have low fuel efficiency.
- There is a moderate, positive relationship with model year (r = 0.58) which indicates fuel economy improvement over time.
- Usually Japanese originated cars showed greater MPG compared to American cars, which can clearly be seen with boxplots of the ‘origin’ feature.

Lastly, scatter plots confirmed the existence of linear relationships for MPG against weight, displacement, and horsepower, while histograms showed right-skewed distributions for horsepower and displacement.

7.2. Model Performance

The determined MPG has 85 percent of the data variability predicted by the Linear Regression model with MSE equal to 0.43 and R2 equal to 0.85. Random Forest Regressor model performed

best among the other models with MSE of 5.35 and R2 of 0.91. It generated these results more predictively reliably because it is able to pick up non-linear relationships. Similar to the previous case, the most relevant predictors were weight, displacement, and horsepower, which is evident based on feature importance estimated using the Random Forest model.

7.3. Clustering Analysis

K-Means clustering with the elbow method discovered 3 distinct clusters:

- **Cluster 0:** Mostly heavier, lower-mpg, higher-cylinder vehicles often U.S. with larger engines.
- **Cluster 1:** Lighter, higher-mpg, lower-cylinder vehicles often Japanese and European, compact cars.
- **Cluster 2:** Moderate weight, engine size, and fuel efficiency, representing a balance between performance and economy.

These clusters reveal clear patterns in the dataset, showing vehicles naturally grouped by weight, fuel efficiency, and origin, which aligns with known distinctions between American, European, and Japanese cars.

9. Conclusion

The presented problem used machine learning to predict the fuel efficiency of the cars on the dataset of 398 vehicles. Random Forest Regressor has shown better results than the Linear Regression since it has shown the R^2 value of 0.91 as compared to 0.85 which indicates that the model can predict better. Some factors of miles per gallon (MPG) that had influence were weight, displacement, and horsepower. The K-Means clustering algorithm determined three distinct groups of vehicles concerning performance-oriented, efficiency-centered, and balanced segments. The results indicate that the relation that governs the fuel efficiency that could be of use to vehicle manufactures, customers and policy makers during the process of designing and selecting vehicles was capable of being captured using machine learning techniques.

Reference

Arbor, A. (2010) *Contribution Of The Automotive Industry To The Economies Of All Fifty States And The United States*.

International Energy Agency (IEA) (2019) *Fuel Economy in Major Car Markets*.

Kim, H.S. and Koo, W.W. (2010) Factors affecting the carbon allowance market in the US. *Energy Policy*, 38 (4): 1879–1884. doi:10.1016/j.enpol.2009.11.066.

ML | Handling Missing Values (2024). Available at: https://www.geeksforgeeks.org/ml-handling-missing-values/ (Accessed: 14 August 2024).

Schoen, A., Byerly, A., Hendrix, B., et al. (2019) A Machine Learning Model for Average Fuel Consumption in Heavy Vehicles. *IEEE Transactions on Vehicular Technology*, 68 (7): 6343–6351. doi:10.1109/TVT.2019.2916299.

Sharma, P. (2025) K-Means Clustering Algorithm. *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/ (Accessed: 14 June 2025).

Wickramanayake, S. (2015) *Fuel Consumption Prediction of Fleet Vehicles Using Machine Leaning: A Comparative Study*.

Xie, F. and Lin, Z. (2017) *Market-driven Automotive Industry Compliance with CAFE and GHG Standards:*

Yoo, S., Shin, J. and Choi, S.-H. (2025a) Machine learning vehicle fuel efficiency prediction. *Scientific Reports*, 15 (1): 14815. doi:10.1038/s41598-025-96999-0.

Yoo, S., Shin, J. and Choi, S.-H. (2025b) Machine learning vehicle fuel efficiency prediction. *Scientific Reports*, 15 (1): 14815. doi:10.1038/s41598-025-96999-0.

Appendix

Code and output for data processing and model training.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats.mstats import winsorize
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

```
# Load the dataset
df = pd.read_csv('Automobile.csv')
```

```
# Basic statistical analysis
print("Dataset shape:", df.shape)
```

```
Dataset shape: (398, 9)
```

```
print("\nFirst 5 rows:")
df.head()
```

```
First 5 rows:
```

	name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	70	usa
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5	70	usa
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0	70	usa
3	amc rebel sst	16.0	8	304.0	150.0	3433	12.0	70	usa
4	ford torino	17.0	8	302.0	140.0	3449	10.5	70	usa

```
print("\nData types and missing values:")
df.info()
```

```
Data types and missing values:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   name          398 non-null    object
 1   mpg           398 non-null    float64
 2   cylinders     398 non-null    int64
 3   displacement  398 non-null    float64
 4   horsepower    392 non-null    float64
 5   weight        398 non-null    int64
 6   acceleration  398 non-null    float64
 7   model_year    398 non-null    int64
 8   origin        398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

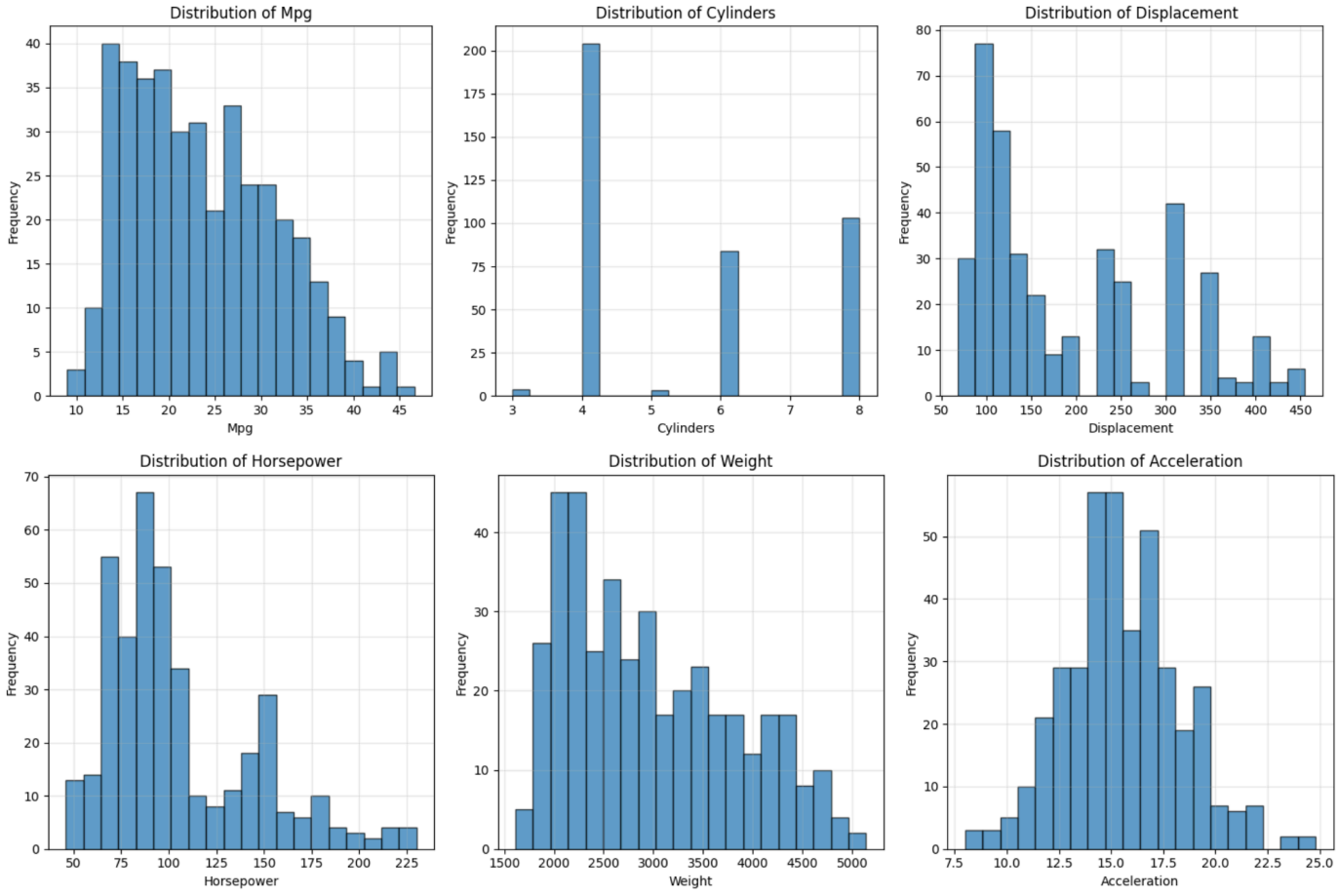
```
print("\nDescriptive Analysis:")
df.describe()
```

Descriptive Analysis:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	cluster
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	104.304020	2970.424623	15.568090	76.010050	1.017588
std	7.815984	1.701004	104.269838	38.222625	846.841774	2.757689	3.697627	0.693396
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000	0.000000
25%	17.500000	4.000000	104.250000	76.000000	2223.750000	13.825000	73.000000	1.000000
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	262.000000	125.000000	3608.000000	17.175000	79.000000	1.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000	2.000000

```
print("\nDistribution Analysis")
# Multiple histograms for all numeric variables
numeric_cols = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration']

plt.figure(figsize=(15, 10))
for i, col in enumerate(numeric_cols, 1):
    plt.subplot(2, 3, i)
    plt.hist(df[col], bins=20, alpha=0.7, edgecolor='black')
    plt.title(f'Distribution of {col.title()}')
    plt.xlabel(col.title())
    plt.ylabel('Frequency')
    plt.grid(True, alpha=0.3)

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

```
Distribution Analysis
```




```
x = df['name'][0]
x[:x.index(" ")]
```

```
'chevrolet'
```

```
def company_name(x):
  if " " in x:
    return x[:x.index(" ")]
  else:
    return x

df["Company_name"] = df["name"].apply(company_name)
df
```

	name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	Company_name
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5	70	usa	buick
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth
3	amc rebel sst	16.0	8	304.0	150.0	3433	12.0	70	usa	amc
4	ford torino	17.0	8	302.0	140.0	3449	10.5	70	usa	ford
...	...	...	...	...	...	...	...	...	...	...
393	ford mustang gl	27.0	4	140.0	86.0	2790	15.6	82	usa	ford
394	vw pickup	44.0	4	97.0	52.0	2130	24.6	82	europe	vw
395	dodge rampage	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge
396	ford ranger	28.0	4	120.0	79.0	2625	18.6	82	usa	ford
397	chevy s-10	31.0	4	119.0	82.0	2720	19.4	82	usa	chevy

398 rows × 10 columns

```
def car_name(x):
  if " " in x:
    return x[x.index(" ")+1:]
  else:
    return x

df["car_name"] = df["name"].apply(car_name)
df
```

	name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	Company_name	car_name
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet	chevelle malibu
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5	70	usa	buick	skylark 320
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth	satellite
3	amc rebel sst	16.0	8	304.0	150.0	3433	12.0	70	usa	amc	rebel sst
4	ford torino	17.0	8	302.0	140.0	3449	10.5	70	usa	ford	torino
...	...	...	...	...	...	...	...	...	...	...	...
393	ford mustang gl	27.0	4	140.0	86.0	2790	15.6	82	usa	ford	mustang gl
394	vw pickup	44.0	4	97.0	52.0	2130	24.6	82	europe	vw	pickup
395	dodge rampage	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge	rampage
396	ford ranger	28.0	4	120.0	79.0	2625	18.6	82	usa	ford	ranger
397	chevy s-10	31.0	4	119.0	82.0	2720	19.4	82	usa	chevy	s-10

398 rows × 11 columns

```
df = df.drop("name", axis=1)
df
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	Company_name	car_name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet	chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick	skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth	satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc	rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford	torino
...	...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.0	2790	15.6	82	usa	ford	mustang gl
394	44.0	4	97.0	52.0	2130	24.6	82	europe	vw	pickup
395	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge	rampage
396	28.0	4	120.0	79.0	2625	18.6	82	usa	ford	ranger
397	31.0	4	119.0	82.0	2720	19.4	82	usa	chevy	s-10

398 rows × 10 columns

```
# Check for missing values
print("\nMissing values per column:")
print(df.isnull().sum())
```

```
Missing values per column:
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
Company_name    0
car_name        0
dtype: int64
```

```
df.duplicated().sum()
```

```
np.int64(0)
```

```
# Handle missing values
df['horsepower'] = df['horsepower'].replace('?', np.nan).astype(float)
df['horsepower'] = df['horsepower'].fillna(df['horsepower'].median())
```

```
print(f"\nShape after handling missing values: {df.shape}")
```

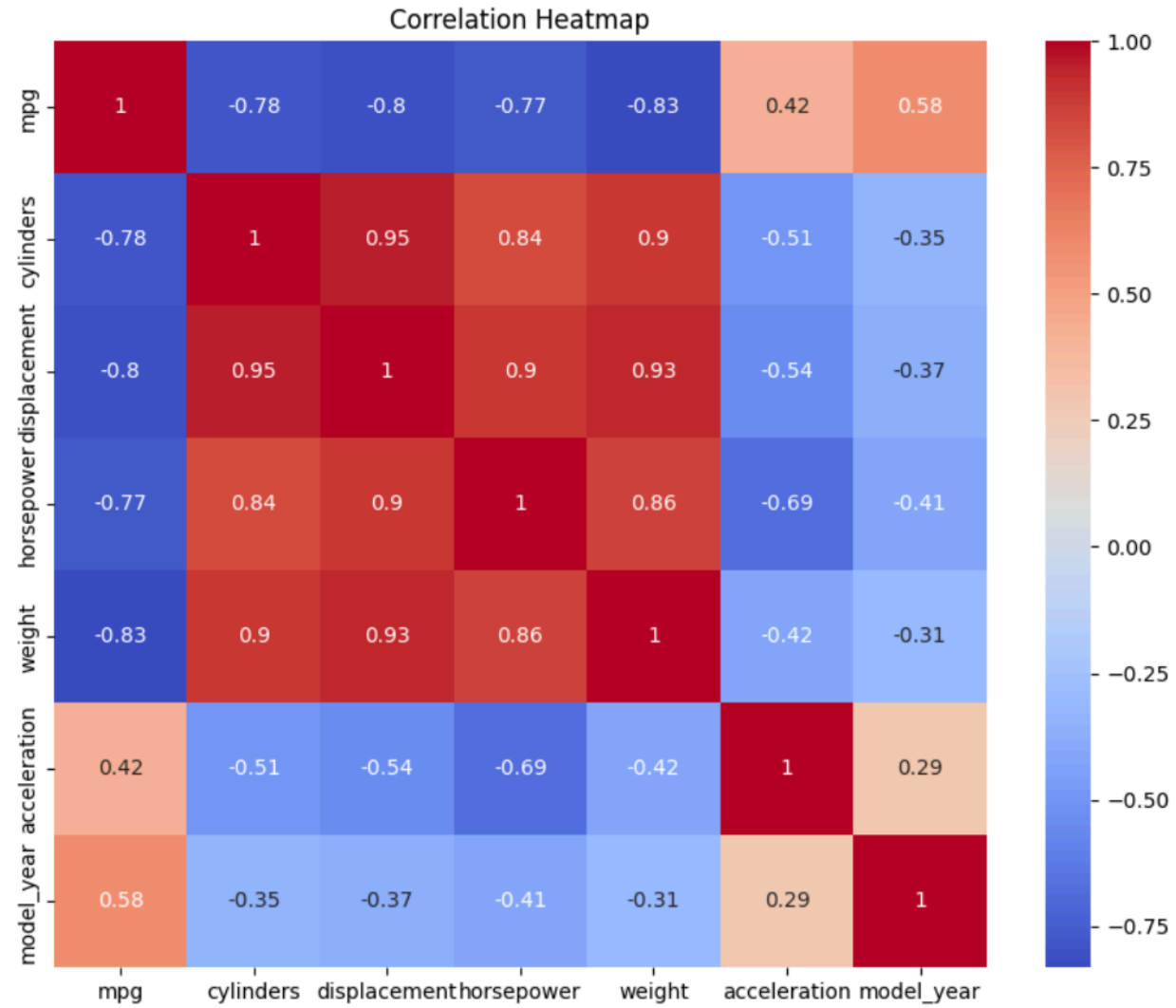
```
Shape after handling missing values: (398, 10)
```

```
# Correlation analysis
numeric_cols = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year']
corr_matrix = df[numeric_cols].corr()
print("\nCorrelation matrix:")
corr_matrix
```

Correlation matrix:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
mpg	1.000000	-0.775396	-0.804203	-0.773453	-0.831741	0.420289	0.579267
cylinders	-0.775396	1.000000	0.950721	0.841284	0.896017	-0.505419	-0.348746
displacement	-0.804203	0.950721	1.000000	0.895778	0.932824	-0.543684	-0.370164
horsepower	-0.773453	0.841284	0.895778	1.000000	0.862442	-0.686590	-0.413733
weight	-0.831741	0.896017	0.932824	0.862442	1.000000	-0.417457	-0.306564
acceleration	0.420289	-0.505419	-0.543684	-0.686590	-0.417457	1.000000	0.288137
model_year	0.579267	-0.348746	-0.370164	-0.413733	-0.306564	0.288137	1.000000

```
# Visualize correlations
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



```python
# Hypothesis testing: Do cars from different origins have different mpg?
usa_mpg = df[df['origin'] == 'usa']['mpg']
japan_mpg = df[df['origin'] == 'japan']['mpg']
europe_mpg = df[df['origin'] == 'europe']['mpg']
```

```python
# ANOVA test
f_stat, p_value = stats.f_oneway(usa_mpg, japan_mpg, europe_mpg)
print(f"\nANOVA test for mpg across origins: F-statistic={f_stat:.2f}, p-value={p_value:.4f}")
```

```
ANOVA test for mpg across origins: F-statistic=98.54, p-value=0.0000
```

```python
# Performing post-hoc tests
if p_value < 0.05:
    print("\nPost-hoc Tukey tests:")
    from statsmodels.stats.multicomp import pairwise_tukeyhsd
    tukey = pairwise_tukeyhsd(endog=df['mpg'], groups=df['origin'], alpha=0.05)
    print(tukey.summary())
```

```
Post-hoc Tukey tests:
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
group1 group2 meandiff p-adj   lower    upper  reject
----------------------------------------------------
europe  japan   2.5592 0.0404   0.0877  5.0307   True
europe    usa  -7.8079    0.0  -9.8448  -5.771   True
 japan    usa -10.3671    0.0 -12.3114 -8.4228   True
----------------------------------------------------
```

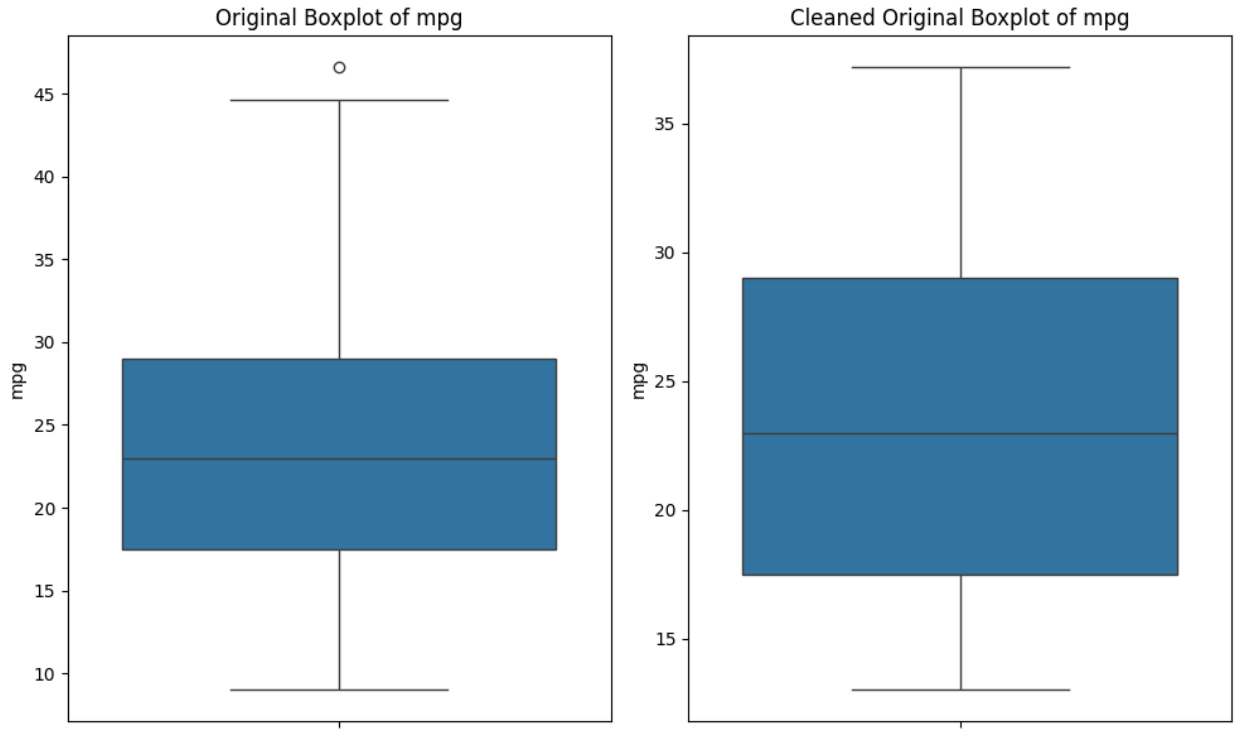
```python
def winsorize_outliers(df, cols, limits=(0.05, 0.05)):
    df_winsorized = df.copy()

    print(f"Applying winsorization with limits: {limits}")

    for col in cols:
        # Store original values for comparison
        original_values = df_winsorized[col].copy()

        # Apply winsorization
        df_winsorized[col] = winsorize(original_values, limits=limits)

        # Count how many values were changed
        values_changed = (original_values != df_winsorized[col]).sum()

        # Show statistics
        print(f"\n{col}:")
        print(f"  Original range: [{original_values.min():.2f}, {original_values.max():.2f}]")
        print(f"  Winsorized range: [{df_winsorized[col].min():.2f}, {df_winsorized[col].max():.2f}]")
        print(f"  Values changed: {values_changed}")
        print(f"  Percentiles used: {original_values.quantile(limits[0]):.2f} to {original_values.quantile(1-limits[1]):.2f}")

    return df_winsorized

# Usage example
numeric_cols = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration']

# Apply winsorization
df_cleaned = winsorize_outliers(df, numeric_cols, limits=(0.05, 0.05))
```

```
Applying winsorization with limits: (0.05, 0.05)

mpg:
  Original range: [9.00, 46.60]
  Winsorized range: [13.00, 37.20]
  Values changed: 32
  Percentiles used: 13.00 to 37.03

cylinders:
  Original range: [3.00, 8.00]
  Winsorized range: [4.00, 8.00]
  Values changed: 4
  Percentiles used: 4.00 to 8.00

displacement:
  Original range: [68.00, 455.00]
  Winsorized range: [85.00, 400.00]
  Values changed: 27
  Percentiles used: 85.00 to 400.00

horsepower:
  Original range: [46.00, 230.00]
  Winsorized range: [60.00, 180.00]
  Values changed: 32
  Percentiles used: 60.85 to 180.00

weight:
  Original range: [1613.00, 5140.00]
  Winsorized range: [1915.00, 4464.00]
  Values changed: 38
  Percentiles used: 1923.50 to 4464.00

acceleration:
  Original range: [8.00, 24.80]
  Winsorized range: [11.20, 20.50]
  Values changed: 36
  Percentiles used: 11.29 to 20.41
```

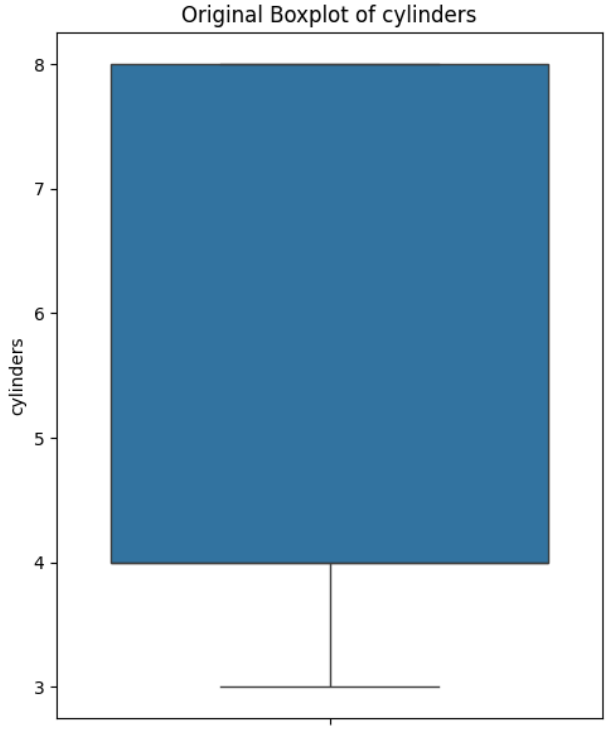
```python
print(f"Before: {df.shape[0]} rows")
print(f"After: {df_cleaned.shape[0]} rows")
```

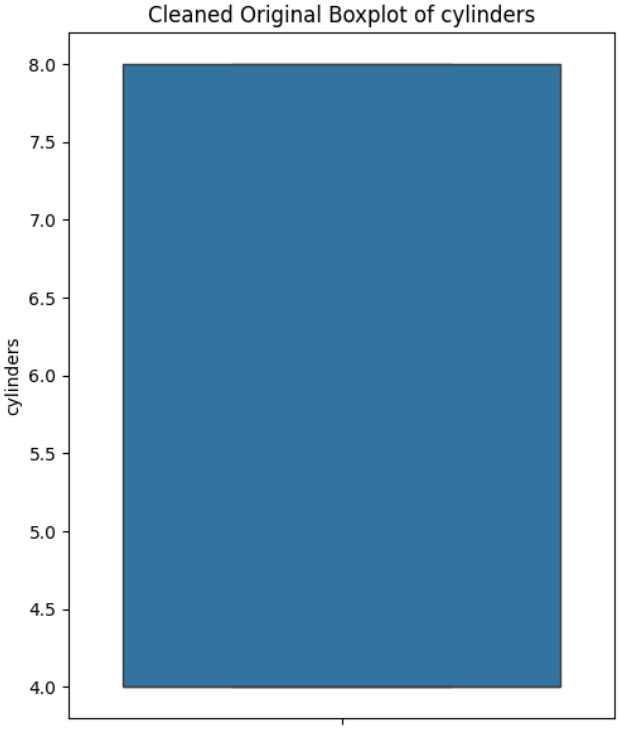
```
Before: 398 rows
After: 398 rows
```

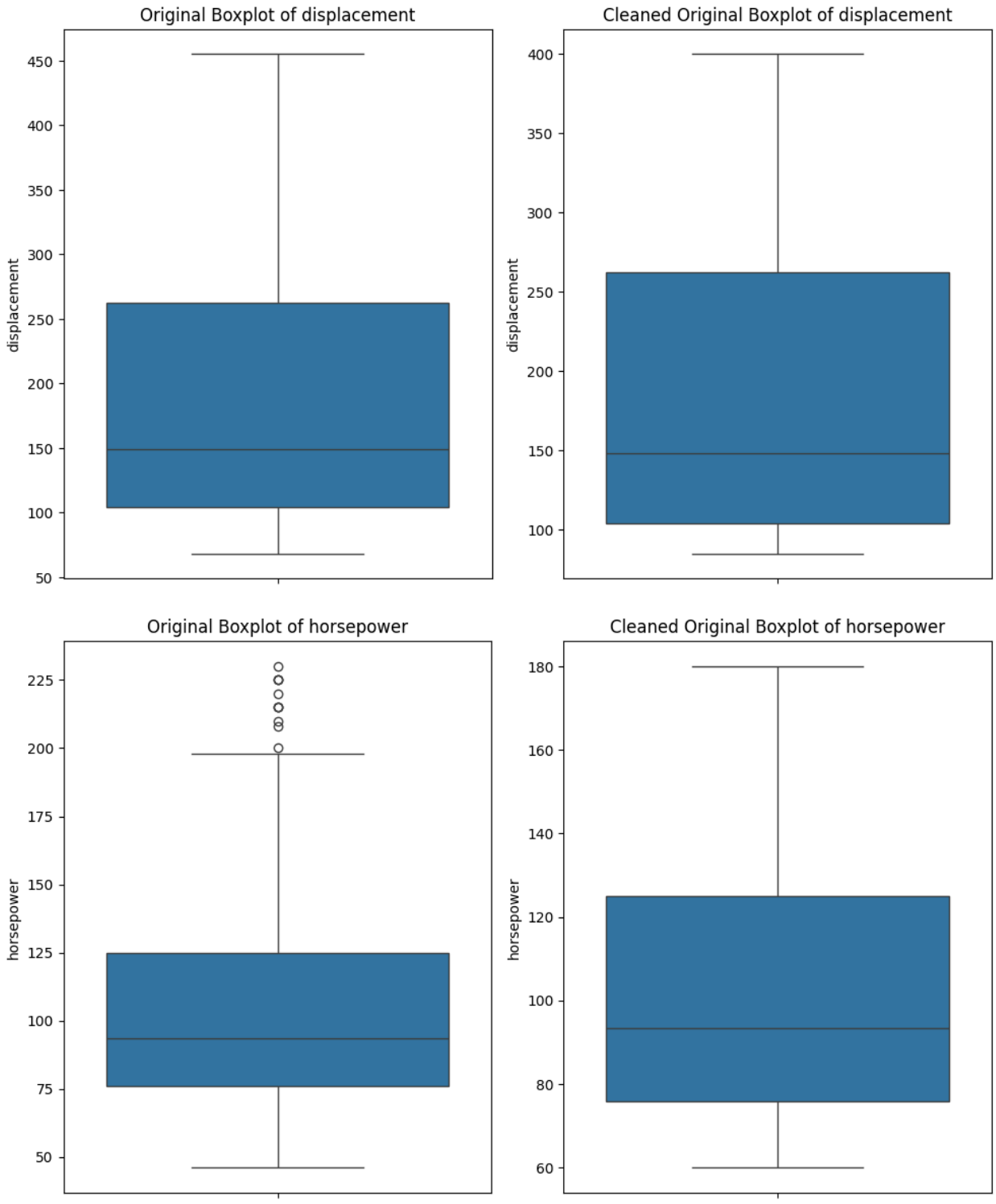
```python
# First, identify the outliers that were removed
outliers_removed = df[~df.index.isin(df_cleaned.index)]

# Method 1: Side-by-side comparison
for col in df_cleaned.columns[:-4]:
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 6))

    # Original data with outliers
    sns.boxplot(y=col, data=df, ax=ax1)
    ax1.set_title(f'Original Boxplot of {col}')

    # Cleaned data without outliers
    sns.boxplot(y=col, data=df_cleaned, ax=ax2)
    ax2.set_title(f'Cleaned Original Boxplot of {col}')

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

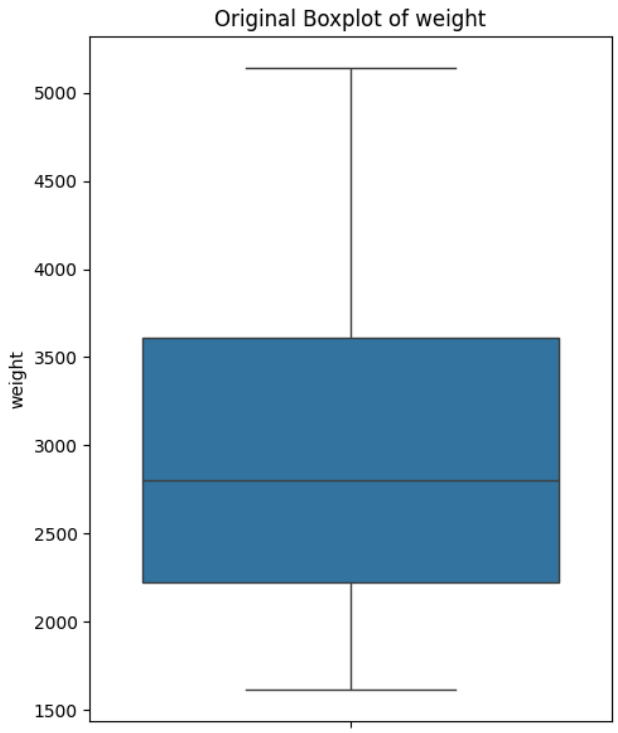

Original Boxplot of mpg
mpg
45
40
35
30
25
20
15
10

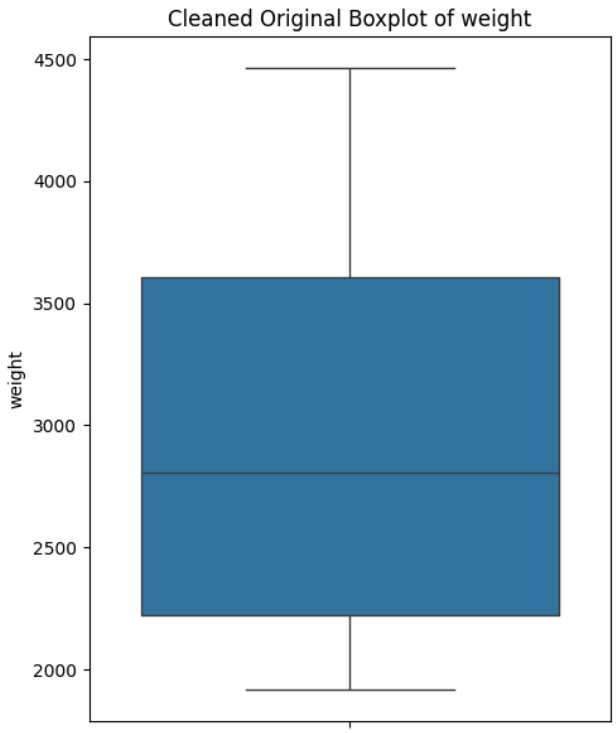
Cleaned Original Boxplot of mpg
mpg
35
30
25
20
15

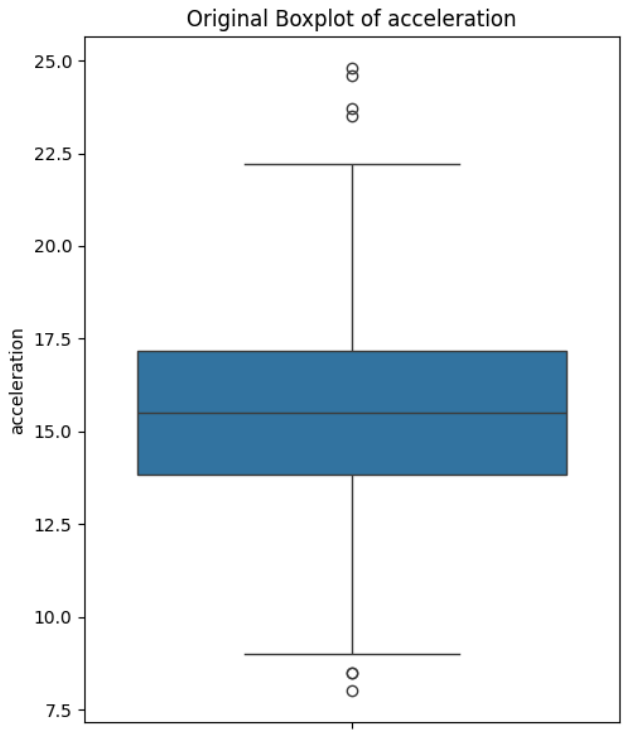

Original Boxplot of cylinders
cylinders
8
7
6
5
4
3

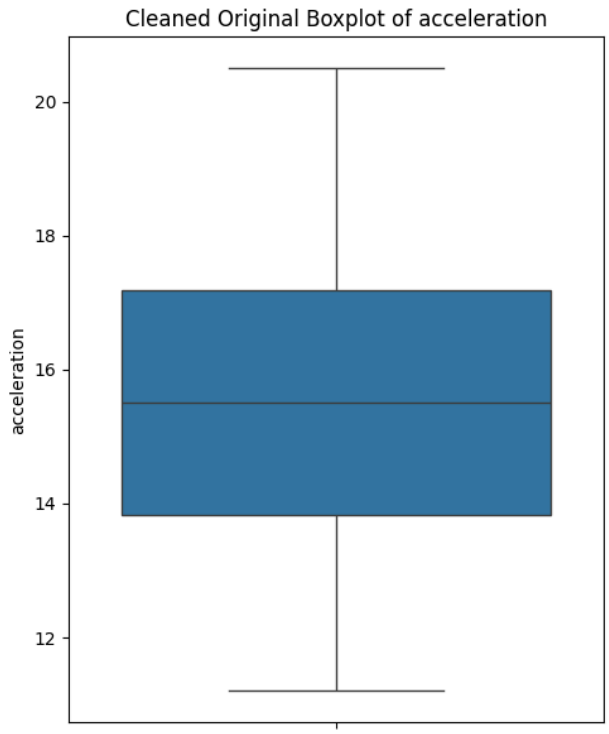

Cleaned Original Boxplot of cylinders
cylinders
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0


Original Boxplot of displacement
displacement
450
400
350
300
250
200
150
100
50
Cleaned Original Boxplot of displacement
displacement
400
350
300
250
200
150
100
Original Boxplot of horsepower
horsepower
225
200
175
150
125
100
75
50
Cleaned Original Boxplot of horsepower
horsepower
180
160
140
120
100
80
60


Original Boxplot of weight
weight
5000
4500
4000
3500
3000
2500
2000
1500


Cleaned Original Boxplot of weight
weight
4500
4000
3500
3000
2500
2000

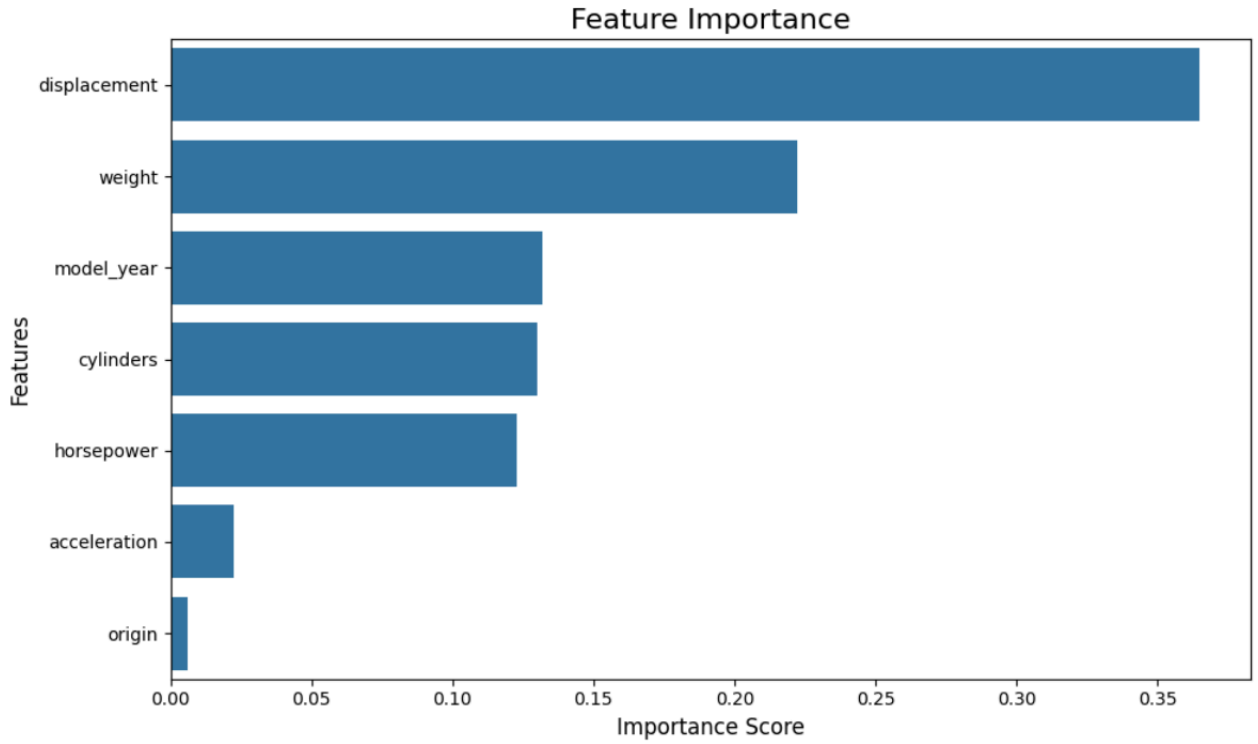

Original Boxplot of acceleration
acceleration
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5


Cleaned Original Boxplot of acceleration
acceleration
20
18
16
14
12

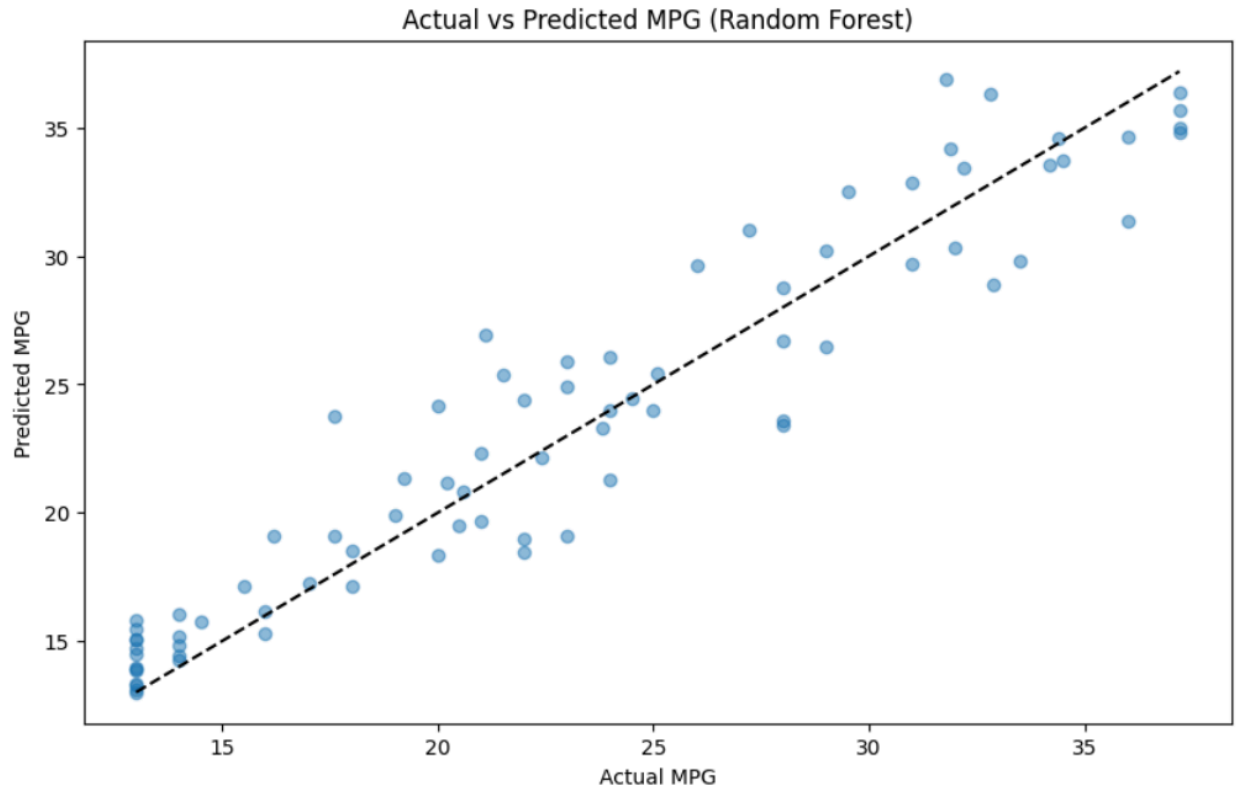
```python
# Prepare data for mpg prediction
X = df_cleaned[['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year']].copy()
y = df_cleaned['mpg']

# Encode origin if needed
le = LabelEncoder()
X['origin'] = le.fit_transform(df_cleaned['origin'])

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=55)

# Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

# Random Forest
rf = RandomForestRegressor(n_estimators=100, random_state=55)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

# Evaluate models
print("\nLinear Regression Performance:")
print(f"MSE: {mean_squared_error(y_test, y_pred_lr):.2f}")
print(f"R²: {r2_score(y_test, y_pred_lr):.2f}")

print("\nRandom Forest Performance:")
print(f"MSE: {mean_squared_error(y_test, y_pred_rf):.2f}")
print(f"R²: {r2_score(y_test, y_pred_rf):.2f}")

# Feature importance
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': rf.feature_importances_
}).sort_values('Importance', ascending=False)

print("\nFeature Importance:")
print(feature_importance)
```

```
Linear Regression Performance:
MSE: 9.58
R²: 0.84

Random Forest Performance:
MSE: 5.60
R²: 0.91

Feature Importance:
        Feature  Importance
1  displacement    0.364912
3        weight    0.222373
5    model_year    0.131846
0     cylinders    0.130022
2    horsepower    0.122552
4  acceleration    0.022310
6        origin    0.005986
```

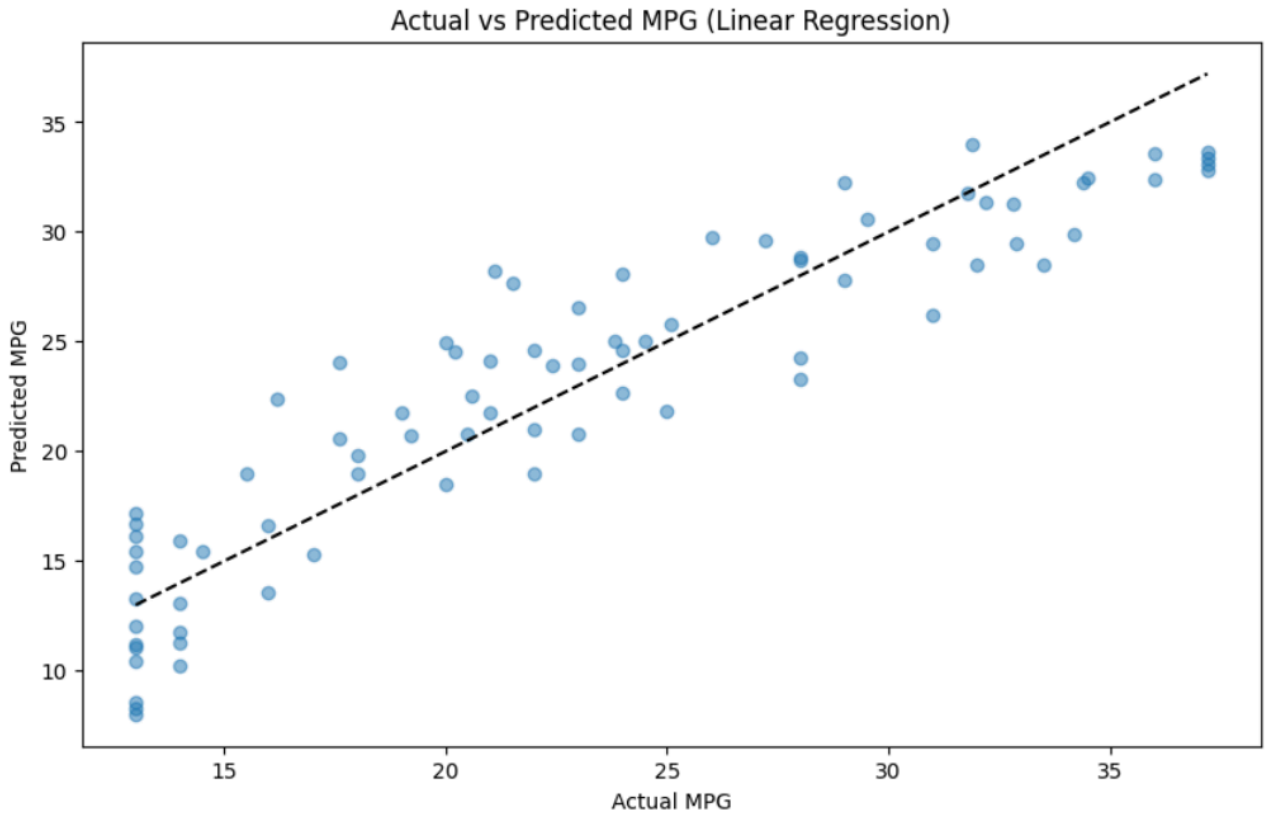
```
# Plotting Feature Importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance, legend=False)
plt.title('Feature Importance', fontsize=16)
plt.xlabel('Importance Score', fontsize=12)
plt.ylabel('Features', fontsize=12)
plt.tight_layout()
plt.show()
```



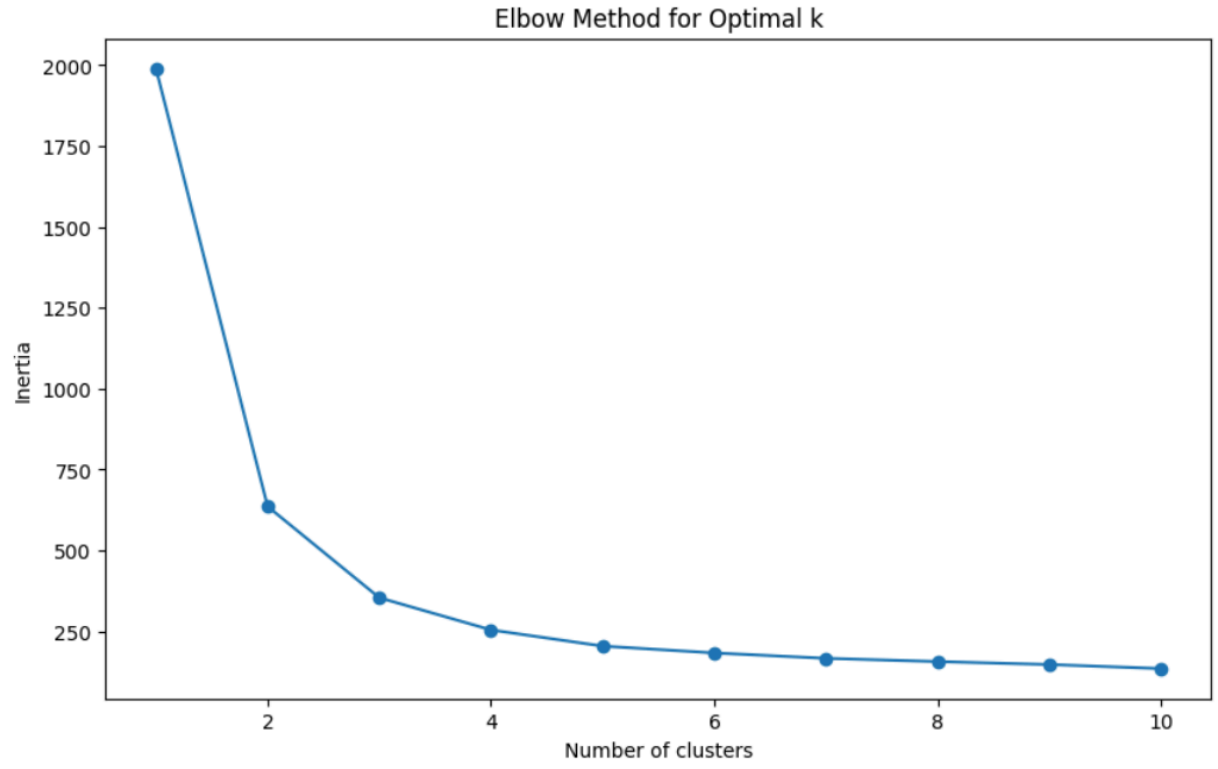
```python
# Visualize predictions vs actual
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_rf, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--')
plt.xlabel('Actual MPG')
plt.ylabel('Predicted MPG')
plt.title('Actual vs Predicted MPG (Random Forest)')
plt.show()
```



```python
# Visualize predictions vs actual
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_lr, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--')
plt.xlabel('Actual MPG')
plt.ylabel('Predicted MPG')
plt.title('Actual vs Predicted MPG (Linear Regression)')
plt.show()
```



```python
# Prepare data for clustering
cluster_features = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight']
X_cluster = df[cluster_features]

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_cluster)

# Determine optimal number of clusters using elbow method
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=55)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(range(1,11), inertia, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```



```python
# Perform clustering with selected k
k = 3
kmeans = KMeans(n_clusters=k, random_state=55)
clusters = kmeans.fit_predict(X_scaled)

# Add clusters to dataframe
df['cluster'] = clusters

# Analyze clusters
print("\nCluster characteristics:")
print(df.groupby('cluster')[cluster_features].mean())
```

```
Cluster characteristics:
                mpg  cylinders  displacement  horsepower       weight
cluster
0         19.806522   5.978261    215.815217  101.809783  3204.217391
1         29.400000   4.004831    109.780193   78.533816  2306.067633
2         14.654545   8.000000    347.515152  160.505051  4142.272727
```

```python
# Visualize clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x='weight', y='mpg', hue='cluster', data=df, palette='viridis')
plt.title('Vehicle Clusters by Weight and MPG')
plt.show()
```