# Comprehensive Car Dataset Analysis

Your Name

October 10, 2024

# Contents

# 1 Introduction

In this report, we perform an end-to-end machine learning project using the car dataset. The analysis includes data exploration, preprocessing, feature engineering, modeling, evaluation, and visualization. We aim to predict the miles per gallon (MPG) for cars using regression models and classify whether a car is from Ford using classification models.

# 2 Dataset Description

The **Auto MPG Dataset** consists of 398 cars, including various technical specifications such as:

- **mpg**: Miles per gallon (continuous)
- **cylinders**: Number of cylinders (multi-valued discrete)
- **displacement**: Engine displacement in cubic inches (continuous)
- **horsepower**: Engine horsepower (continuous)
- **weight**: Vehicle weight in pounds (continuous)
- **acceleration**: Time to accelerate from 0 to 60 mph (continuous)
- **model year**: Model year (multi-valued discrete)
- **origin**: Origin of the car (1: USA, 2: Europe, 3: Japan)
- **car name**: Car model name (string)

**Source**: The dataset is available from the UCI Machine Learning Repository.
https://archive.ics.uci.edu/ml/datasets/auto+mpg

# 3 Data Loading and Exploration

## 3.1 Importing Libraries

We start by importing the necessary libraries and ensuring compatibility.

```
# Import necessary libraries
import sys
assert sys.version_info >= (3, 7)

from packaging import version
import sklearn
assert version.parse(sklearn.__version__) >= version.parse("1.0.1")

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

sns.set(style="whitegrid")
%matplotlib inline

warnings.filterwarnings('ignore')
np.random.seed(42)
```

### 3.2 Loading the Dataset

We load the dataset using Pandas.

```
# Loading the car dataset
cars = pd.read_csv("cars.csv")

# Display the first few rows
cars.head()
```

### 3.3 Data Overview

We check data types and identify missing values.

```
# Checking data types and missing values
cars.info()

# Summary statistics
cars.describe()
```

## 4 Data Cleaning and Preprocessing

### 4.1 Handling Missing Values and Data Types

We handle missing values in the `horsepower` column and convert data types.

```
# Replace ’?’ with NaN and convert ’horsepower’ to numeric
cars[’horsepower’].replace(’?’, np.nan, inplace=True)
cars[’horsepower’] = pd.to_numeric(cars[’horsepower’], errors=’coerce’)

# Impute missing ’horsepower’ with median
median_horsepower = cars[’horsepower’].median()
cars[’horsepower’].fillna(median_horsepower, inplace=True)

# Convert ’origin’ to categorical
cars[’origin’] = cars[’origin’].map({1: ’USA’, 2: ’Europe’, 3: ’Japan’})

# Extract ’manufacturer’ from ’car name’
cars[’manufacturer’] = cars[’car␣name’].str.split().str[0]

# Display cleaned dataset
cars.head()
```

### 4.2 Handling Outliers

We detect and remove outliers using the z-score method.

```
# Detecting outliers using z-score
from scipy import stats

z_scores = stats.zscore(cars.select_dtypes(include=[np.number]))
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3).all(axis=1)
cars = cars[filtered_entries]

print("Dataset␣shape␣after␣removing␣outliers:", cars.shape)
```

# 5 Exploratory Data Analysis (EDA)

## 5.1 Distribution of MPG

We visualize the distribution of the target variable `mpg`.

```
# Distribution of MPG
plt.figure(figsize=(10, 6))
sns.histplot(cars["mpg"], bins=30, kde=True, color='blue', alpha=0.7)
plt.xlabel("Miles Per Gallon (MPG)")
plt.ylabel("Frequency")
plt.title("Distribution of MPG")
plt.savefig("mpg_distribution.png")
plt.show()
```

mpg_distribution.png

Figure 1: Distribution of MPG

## 5.2 Correlation Matrix

We compute and visualize the correlation matrix.

```
# Correlation Matrix
plt.figure(figsize=(12, 10))
corr_matrix = cars.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.savefig('correlation_matrix.png')
plt.show()
```

correlation_matrix.png

Figure 2: Correlation Matrix

### 5.3 Pair Plot

We create pair plots to visualize relationships between features.

```
# Pair Plot
sns.pairplot(cars[['mpg', 'horsepower', 'weight', 'acceleration', 'displacement']])
plt.savefig('pair_plot.png')
plt.show()
```

pair_plot.png

Figure 3: Pair Plot of Selected Features

### 5.4 Manufacturer Distribution

We analyze the distribution of car manufacturers.

```
# Distribution of Car Manufacturers
plt.figure(figsize=(12, 8))
top_manufacturers = cars['manufacturer'].value_counts().nlargest(20)
sns.barplot(x=top_manufacturers.index, y=top_manufacturers.values, palette="viridis")
plt.xticks(rotation=45)
plt.xlabel("Car Manufacturer")
plt.ylabel("Number of Cars")
plt.title("Top 20 Car Manufacturers in the Dataset")
plt.tight_layout()
plt.savefig("manufacturer_distribution.png")
plt.show()
```

manufacturer_distribution.png

Figure 4: Top 20 Car Manufacturers

## 5.5 Class Imbalance Analysis

We examine the class distribution for the classification task.

```python
# Class Imbalance Analysis
cars['is_ford'] = (cars['manufacturer'] == 'ford').astype(int)
class_counts = cars['is_ford'].value_counts()
print("Class Distribution:")
print(class_counts)

plt.figure(figsize=(6, 4))
sns.barplot(x=class_counts.index, y=class_counts.values)
plt.xticks([0, 1], ['Not Ford', 'Ford'])
plt.xlabel('Car Type')
plt.ylabel('Count')
plt.title('Ford vs. Not Ford Distribution')
plt.savefig('class_distribution.png')
plt.show()
```

class_distribution.png

Figure 5: Ford vs. Not Ford Distribution

# 6 Feature Engineering

## 6.1 Creating New Features

We engineer new features to enhance model performance.

- **Power-to-Weight Ratio**:

$$\text{Power-to-Weight Ratio} = \frac{\text{Horsepower}}{\text{Weight}}$$

- **Displacement per Cylinder**:

$$\text{Displacement per Cylinder} = \frac{\text{Displacement}}{\text{Cylinders}}$$

```python
# Creating power-to-weight ratio
cars['power_to_weight'] = cars['horsepower'] / cars['weight']

# Creating displacement per cylinder
cars['displacement_per_cylinder'] = cars['displacement'] / cars['cylinders']

# Encoding categorical variables
cars = pd.get_dummies(cars, columns=['origin'], drop_first=True)

cars.head()
```

# 7 Defining Features and Splitting Data

## 7.1 Preparing Features and Targets

We define the feature set and target variables for regression and classification.

```python
# Defining features and target for regression
features = ['horsepower', 'weight', 'acceleration', 'displacement',
            'power_to_weight', 'displacement_per_cylinder',
            'origin_Europe', 'origin_Japan']
X_regression = cars[features]
y_regression = cars['mpg']

# Defining features and target for classification
X_classification = cars[features]
y_classification = cars['is_ford']
```

## 7.2 Splitting the Data

We split the data into training and testing sets.

```python
# Splitting the data into training and testing sets
from sklearn.model_selection import train_test_split

# Regression data split
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_regression, y_regression, test_size=0.2, random_state=42
)

# Classification data split
X_train_clf, X_test_clf, y_train_clf, y_test_clf = train_test_split(
    X_classification, y_classification, test_size=0.2, random_state=42, stratify=y_classification
)
```

# 8 Modeling and Evaluation

## 8.1 Regression Models

### 8.1.1 Linear Regression

We train a Linear Regression model and evaluate its performance.

```python
# Linear Regression Model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

lin_reg = LinearRegression()
lin_reg.fit(X_train_reg, y_train_reg)

# Predictions and Evaluation
y_pred_lin_reg = lin_reg.predict(X_test_reg)
mse_lin_reg = mean_squared_error(y_test_reg, y_pred_lin_reg)
mae_lin_reg = mean_absolute_error(y_test_reg, y_pred_lin_reg)
r2_lin_reg = r2_score(y_test_reg, y_pred_lin_reg)

print("Linear Regression Evaluation:")
print(f"MSE: {mse_lin_reg:.2f}")
print(f"MAE: {mae_lin_reg:.2f}")
print(f"R^2 Score: {r2_lin_reg:.2f}")
```

**Linear Regression Equations**:
The prediction is made using:

$$\hat{y} = X\beta + \epsilon$$

where:

- $X$ is the feature matrix

- $\beta$ are the coefficients
- $\epsilon$ is the error term

### 8.1.2 Random Forest Regression

We train a Random Forest Regression model.

```python
# Random Forest Regression
from sklearn.ensemble import RandomForestRegressor

rf_reg = RandomForestRegressor(random_state=42)
rf_reg.fit(X_train_reg, y_train_reg)

# Predictions and Evaluation
y_pred_rf_reg = rf_reg.predict(X_test_reg)
mse_rf_reg = mean_squared_error(y_test_reg, y_pred_rf_reg)
mae_rf_reg = mean_absolute_error(y_test_reg, y_pred_rf_reg)
r2_rf_reg = r2_score(y_test_reg, y_pred_rf_reg)

print("Random Forest Regression Evaluation:")
print(f"MSE: {mse_rf_reg:.2f}")
print(f"MAE: {mae_rf_reg:.2f}")
print(f"R^2 Score: {r2_rf_reg:.2f}")
```

**Random Forest Regression** combines multiple decision trees to improve predictive accuracy and control over-fitting.

### 8.1.3 Support Vector Regression

We train a Support Vector Regression (SVR) model.

```python
# Support Vector Regression
from sklearn.svm import SVR

svr_reg = SVR()
svr_reg.fit(X_train_reg, y_train_reg)

# Predictions and Evaluation
y_pred_svr_reg = svr_reg.predict(X_test_reg)
mse_svr_reg = mean_squared_error(y_test_reg, y_pred_svr_reg)
mae_svr_reg = mean_absolute_error(y_test_reg, y_pred_svr_reg)
r2_svr_reg = r2_score(y_test_reg, y_pred_svr_reg)

print("Support Vector Regression Evaluation:")
print(f"MSE: {mse_svr_reg:.2f}")
print(f"MAE: {mae_svr_reg:.2f}")
print(f"R^2 Score: {r2_svr_reg:.2f}")
```

## 8.2 Classification Models

### 8.2.1 Logistic Regression

We train a Logistic Regression model.

```python
# Logistic Regression Model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
    classification_report

log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_clf, y_train_clf)

# Predictions and Evaluation
y_pred_log_reg = log_reg.predict(X_test_clf)
```

```python
accuracy_log_reg = accuracy_score(y_test_clf, y_pred_log_reg)
precision_log_reg = precision_score(y_test_clf, y_pred_log_reg)
recall_log_reg = recall_score(y_test_clf, y_pred_log_reg)
f1_log_reg = f1_score(y_test_clf, y_pred_log_reg)

print("Logistic Regression Evaluation:")
print(f"Accuracy: {accuracy_log_reg:.2f}")
print(f"Precision: {precision_log_reg:.2f}")
print(f"Recall: {recall_log_reg:.2f}")
print(f"F1 Score: {f1_log_reg:.2f}")
print("\nClassification Report:")
print(classification_report(y_test_clf, y_pred_log_reg))
```

**Logistic Regression Equation**:
The probability of the positive class is modeled as:

$$p = \frac{1}{1 + e^{-(X\beta)}}$$

### 8.2.2 Random Forest Classification

We train a Random Forest Classifier.

```python
# Random Forest Classification
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train_clf, y_train_clf)

# Predictions and Evaluation
y_pred_rf_clf = rf_clf.predict(X_test_clf)
accuracy_rf_clf = accuracy_score(y_test_clf, y_pred_rf_clf)
precision_rf_clf = precision_score(y_test_clf, y_pred_rf_clf)
recall_rf_clf = recall_score(y_test_clf, y_pred_rf_clf)
f1_rf_clf = f1_score(y_test_clf, y_pred_rf_clf)

print("Random Forest Classification Evaluation:")
print(f"Accuracy: {accuracy_rf_clf:.2f}")
print(f"Precision: {precision_rf_clf:.2f}")
print(f"Recall: {recall_rf_clf:.2f}")
print(f"F1 Score: {f1_rf_clf:.2f}")
print("\nClassification Report:")
print(classification_report(y_test_clf, y_pred_rf_clf))
```

### 8.2.3 Support Vector Classification

We train a Support Vector Classifier.

```python
# Support Vector Classification
from sklearn.svm import SVC

svc_clf = SVC(probability=True)
svc_clf.fit(X_train_clf, y_train_clf)

# Predictions and Evaluation
y_pred_svc_clf = svc_clf.predict(X_test_clf)
accuracy_svc_clf = accuracy_score(y_test_clf, y_pred_svc_clf)
precision_svc_clf = precision_score(y_test_clf, y_pred_svc_clf)
recall_svc_clf = recall_score(y_test_clf, y_pred_svc_clf)
f1_svc_clf = f1_score(y_test_clf, y_pred_svc_clf)

print("Support Vector Classification Evaluation:")
print(f"Accuracy: {accuracy_svc_clf:.2f}")
print(f"Precision: {precision_svc_clf:.2f}")
print(f"Recall: {recall_svc_clf:.2f}")
print(f"F1 Score: {f1_svc_clf:.2f}")
```

```
print("\nClassification Report:")
print(classification_report(y_test_clf, y_pred_svc_clf))
```

# 9 Hyperparameter Tuning

## 9.1 Random Forest Regression Tuning

We use GridSearchCV to fine-tune the Random Forest Regression model.

```
# Hyperparameter Tuning for Random Forest Regression
from sklearn.model_selection import GridSearchCV

param_grid_rf_reg = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10]
}

grid_search_rf_reg = GridSearchCV(RandomForestRegressor(random_state=42), param_grid_rf_reg, cv
    =5)
grid_search_rf_reg.fit(X_train_reg, y_train_reg)

print("Best Parameters for Random Forest Regression:")
print(grid_search_rf_reg.best_params_)

# Evaluating the tuned model
y_pred_rf_reg_tuned = grid_search_rf_reg.predict(X_test_reg)
mse_rf_reg_tuned = mean_squared_error(y_test_reg, y_pred_rf_reg_tuned)
mae_rf_reg_tuned = mean_absolute_error(y_test_reg, y_pred_rf_reg_tuned)
r2_rf_reg_tuned = r2_score(y_test_reg, y_pred_rf_reg_tuned)

print("Tuned Random Forest Regression Evaluation:")
print(f"MSE: {mse_rf_reg_tuned:.2f}")
print(f"MAE: {mae_rf_reg_tuned:.2f}")
print(f"R^2 Score: {r2_rf_reg_tuned:.2f}")
```

# 10 Results Visualization

## 10.1 Regression Results Plot

We plot the actual vs. predicted MPG values for the tuned Random Forest Regression model.