Comprehensive Car Dataset Analysis

Your Name

October 10, 2024

Contents

1	Introduction	2
2	Dataset Description	2
3	Data Loading and Exploration3.1 Importing Libraries3.2 Loading the Dataset3.3 Data Overview	2 3 3
4	Data Cleaning and Preprocessing4.1 Handling Missing Values and Data Types4.2 Handling Outliers	3 3
5	Exploratory Data Analysis (EDA) 5.1 Distribution of MPG 5.2 Correlation Matrix 5.3 Pair Plot 5.4 Manufacturer Distribution 5.5 Class Imbalance Analysis	4 4 4 5 6 7
6	Feature Engineering 6.1 Creating New Features	8
7	Defining Features and Splitting Data7.1 Preparing Features and Targets7.2 Splitting the Data	8 8 9
9	8.1.3 Support Vector Regression 8.2 Classification Models	9 9 10 10 10 11 11
Э	· · ·	12
10		1 2 12

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)
3
   from packaging import version
   import sklearn
6
   assert version.parse(sklearn.__version__) >= version.parse("1.0.1")
9
   import pandas as pd
10
   import numpy as np
   import seaborn as sns
11
   import matplotlib.pyplot as plt
13
   import warnings
14
   sns.set(style="whitegrid")
15
   %matplotlib inline
16
17
   warnings.filterwarnings('ignore')
18
   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
9
10
   cars['origin'] = cars['origin'].map({1: 'USA', 2: 'Europe', 3: 'Japan'})
11
   # Extract 'manufacturer' from 'car name'
12
   cars['manufacturer'] = cars['car_name'].str.split().str[0]
13
14
   # Display cleaned dataset
15
   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)</pre>
```

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()
```

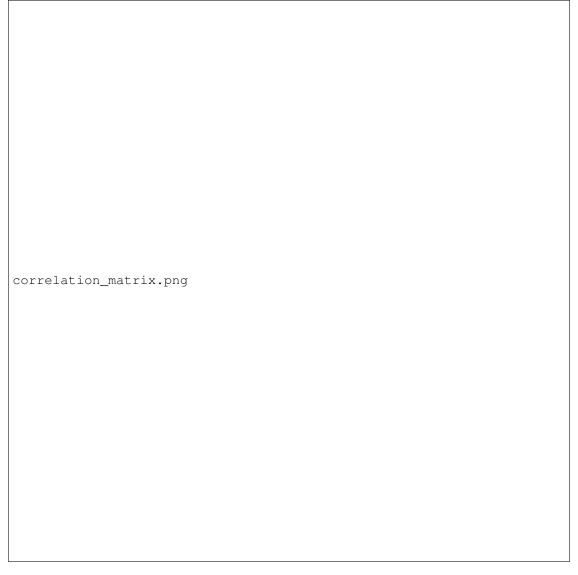


Figure 2: Correlation Matrix

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()
```

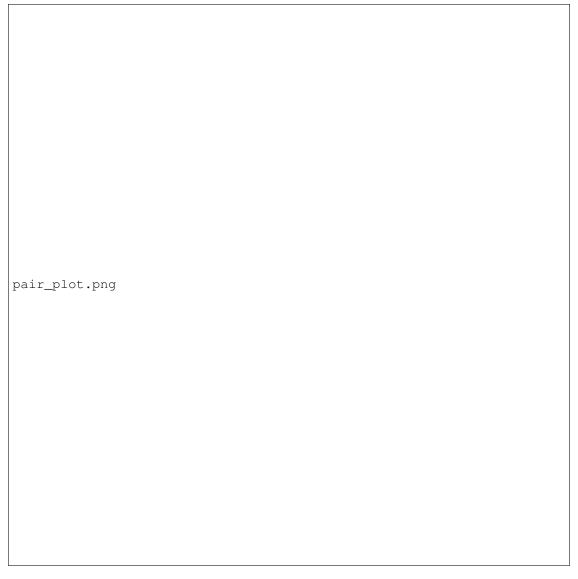


Figure 3: Pair Plot of Selected Features

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()
```

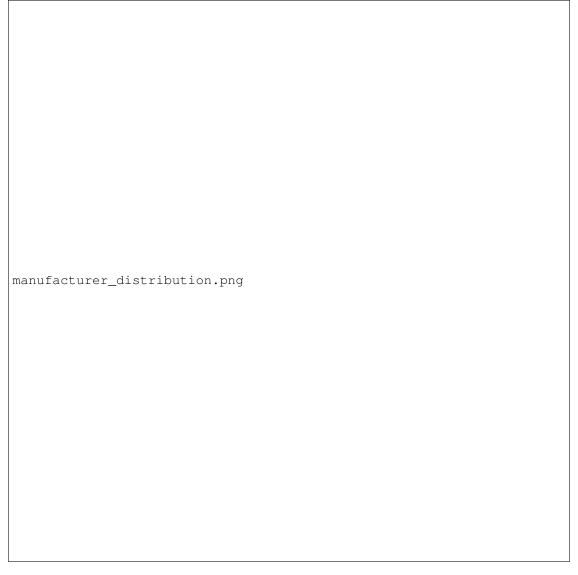


Figure 4: Top 20 Car Manufacturers

5.5 Class Imbalance Analysis

We examine the class distribution for the classification task.

```
# 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()
```

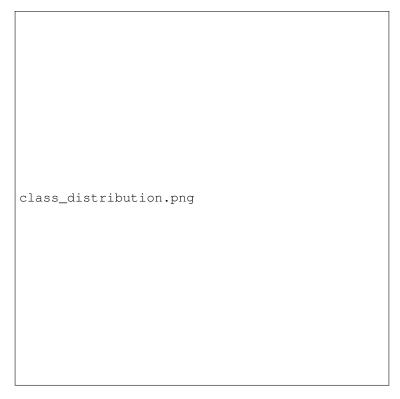


Figure 5: Ford vs. Not Ford Distribution

6 Feature Engineering

6.1 Creating New Features

We engineer new features to enhance model performance.

• Power-to-Weight Ratio:

$$\label{eq:Power-to-Weight Ratio} Power-to-Weight \ Ratio = \frac{Horsepower}{Weight}$$

• Displacement per Cylinder:

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

```
# 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.

7.2 Splitting the Data

We split the data into training and testing sets.

8 Modeling and Evaluation

8.1 Regression Models

8.1.1 Linear Regression

We train a Linear Regression model and evaluate its performance.

```
# Linear Regression Model
   from sklearn.linear_model import LinearRegression
   from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
3
   lin_reg = LinearRegression()
   lin_reg.fit(X_train_reg, y_train_reg)
   # Predictions and Evaluation
8
9
   y_pred_lin_reg = lin_reg.predict(X_test_reg)
   mse_lin_reg = mean_squared_error(y_test_reg, y_pred_lin_reg)
10
   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)
12
   print ("Linear_Regression_Evaluation:")
14
   print(f"MSE:_{mse_lin_reg:.2f}")
15
   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

- β are the coefficients
- ϵ is the error term

8.1.2 Random Forest Regression

We train a Random Forest Regression model.

```
# Random Forest Regression
   from sklearn.ensemble import RandomForestRegressor
2
3
   rf_reg = RandomForestRegressor(random_state=42)
   rf_reg.fit(X_train_reg, y_train_reg)
5
   # Predictions and Evaluation
7
8
   y_pred_rf_reg = rf_reg.predict(X_test_reg)
   mse_rf_reg = mean_squared_error(y_test_reg, y_pred_rf_reg)
9
   mae_rf_reg = mean_absolute_error(y_test_reg, y_pred_rf_reg)
10
   r2_rf_reg = r2_score(y_test_reg, y_pred_rf_reg)
12
   print ("Random_Forest_Regression_Evaluation:")
13
14
   print(f"MSE:_{mse_rf_reg:.2f}")
   print(f"MAE:_{mae_rf_reg:.2f}")
15
   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.

```
# Support Vector Regression
   from sklearn.svm import SVR
2
3
   svr_reg = SVR()
   svr_reg.fit(X_train_reg, y_train_reg)
6
   # 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)
10
   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)
11
12
   print("Support_Vector_Regression_Evaluation:")
13
   print(f"MSE:_{mse_svr_reg:.2f}")
   print(f"MAE:_{mae_svr_reg:.2f}")
15
   print(f"R^2_Score:_{r2_svr_reg:.2f}")
```

8.2 Classification Models

8.2.1 Logistic Regression

We train a Logistic Regression model.

```
accuracy_log_reg = accuracy_score(y_test_clf, y_pred_log_reg)
10
   precision_log_reg = precision_score(y_test_clf, y_pred_log_reg)
11
   recall_log_reg = recall_score(y_test_clf, y_pred_log_reg)
12
   f1_log_reg = f1_score(y_test_clf, y_pred_log_reg)
14
   print("Logistic_Regression_Evaluation:")
15
   print(f"Accuracy:_{accuracy_log_reg:.2f}")
16
   print(f"Precision:_{precision_log_reg:.2f}")
17
   print (f"Recall:_{recall_log_reg:.2f}")
   print(f"F1_Score:_{f1_log_reg:.2f}")
19
   print("\nClassification, Report:")
20
   print(classification_report(y_test_clf, y_pred_log_reg))
21
```

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.

```
# Random Forest Classification
   from sklearn.ensemble import RandomForestClassifier
2
3
   rf_clf = RandomForestClassifier(random_state=42)
4
   rf_clf.fit(X_train_clf, y_train_clf)
6
   # Predictions and Evaluation
7
   y_pred_rf_clf = rf_clf.predict(X_test_clf)
8
   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)
11
12
   f1_rf_clf = f1_score(y_test_clf, y_pred_rf_clf)
13
   print("Random_Forest_Classification_Evaluation:")
14
   print(f"Accuracy:_{accuracy_rf_clf:.2f}")
   print(f"Precision:_{precision_rf_clf:.2f}")
16
   print(f"Recall:_{recall_rf_clf:.2f}")
17
   print(f"F1_Score:_{f1_rf_clf:.2f}")
18
   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.

```
# Support Vector Classification
   from sklearn.svm import SVC
2
3
   svc_clf = SVC(probability=True)
   svc_clf.fit(X_train_clf, y_train_clf)
5
7
   # Predictions and Evaluation
   y_pred_svc_clf = svc_clf.predict(X_test_clf)
8
   accuracy_svc_clf = accuracy_score(y_test_clf, y_pred_svc_clf)
   precision_svc_clf = precision_score(y_test_clf, y_pred_svc_clf)
10
   recall_svc_clf = recall_score(y_test_clf, y_pred_svc_clf)
   f1_svc_clf = f1_score(y_test_clf, y_pred_svc_clf)
12
13
   print("Support_Vector_Classification_Evaluation:")
14
   print(f"Accuracy:_{accuracy_svc_clf:.2f}")
15
  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
2
3
   param_grid_rf_reg = {
       'n_estimators': [50, 100, 200],
5
       'max_depth': [None, 5, 10],
6
       'min_samples_split': [2, 5, 10]
   grid_search_rf_reg = GridSearchCV(RandomForestRegressor(random_state=42), param_grid_rf_reg, cv
10
11
   grid_search_rf_reg.fit(X_train_reg, y_train_reg)
12
   print("Best_Parameters_for_Random_Forest_Regression:")
13
   print (grid_search_rf_reg.best_params_)
14
15
   # Evaluating the tuned model
16
   y_pred_rf_reg_tuned = grid_search_rf_reg.predict(X_test_reg)
17
   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)
19
20
   r2_rf_reg_tuned = r2_score(y_test_reg, y_pred_rf_reg_tuned)
21
   print("Tuned_Random_Forest_Regression_Evaluation:")
   print(f"MSE:_{mse_rf_reg_tuned:.2f}")
23
   print(f"MAE:_{mae_rf_reg_tuned:.2f}")
24
   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.