4.2 Exercise.DSC550 - Jennifer Barrera Conde

April 4, 2024

1 4.2 Exercise

1.1 DSC550-T301 Data Mining

1.2 Jennifer Barrera Conde

1.2.1 1. Load the data as a Pandas data frame and ensure that it imported correctly.

[1]:
```
import pandas as pd

df = pd.read_csv("auto-mpg.csv")

print(df.head())
```

```
    mpg  cylinders  displacement horsepower  weight  acceleration  model year  \
0  18.0          8         307.0        130    3504          12.0          70
1  15.0          8         350.0        165    3693          11.5          70
2  18.0          8         318.0        150    3436          11.0          70
3  16.0          8         304.0        150    3433          12.0          70
4  17.0          8         302.0        140    3449          10.5          70

   origin                   car name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino
```

[ ]:
```

```

1.2.2 2. Begin by prepping the data for modeling.

[2]:
```
# 1. Remove the car name column
df.drop('car name', axis=1, inplace=True)

# 2. The horsepower column values likely imported as a string data type. Figure␣
 ↪out why and replace any strings with the column mean.
# I had to convert horsepower column to numeric.
df['horsepower'] = pd.to_numeric(df['horsepower'], errors='coerce')
```

```
df['horsepower'].fillna(df['horsepower'].mean(), inplace=True)

# 3. Create dummy variables for the origin column
df = pd.get_dummies(df, columns=['origin'], drop_first=True)
```

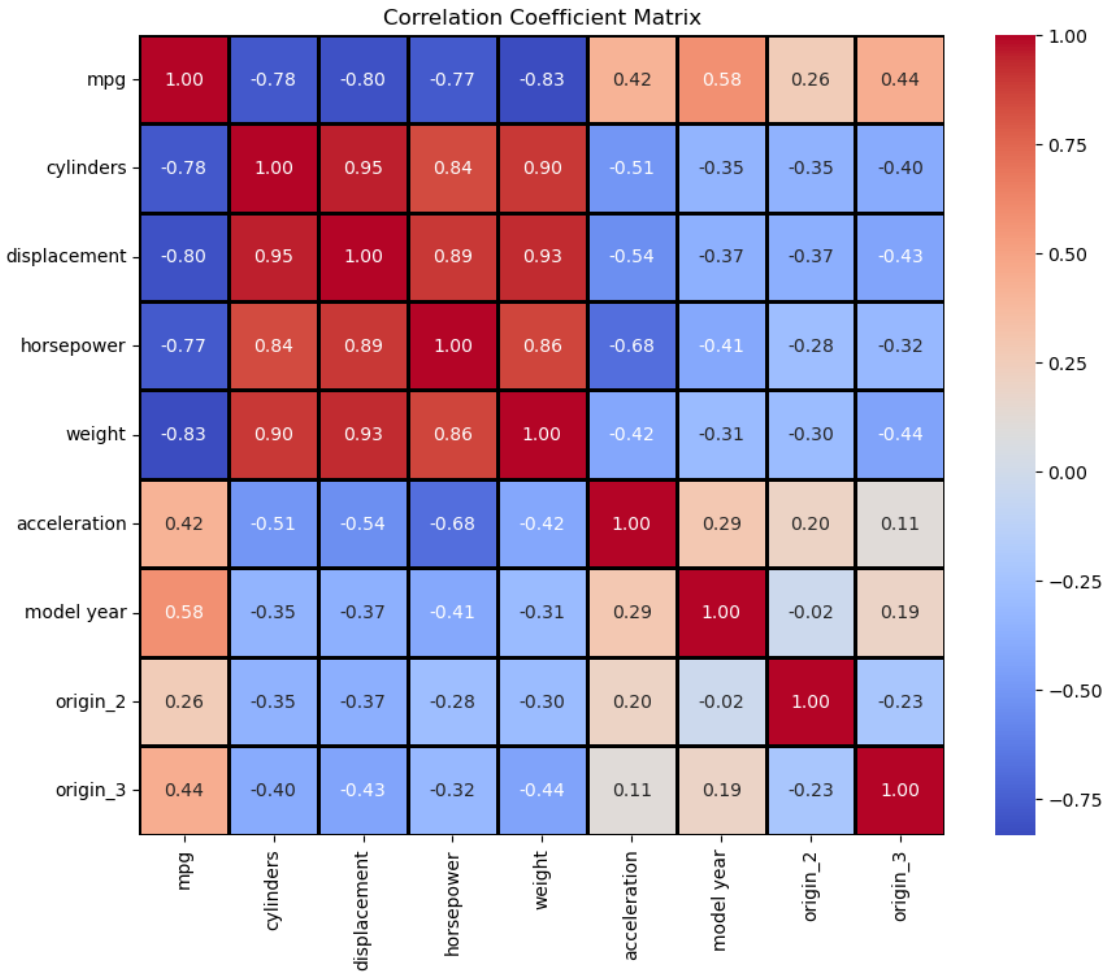
[ ]:

1.2.3 3. Create a correlation coefficient matrix and/or visualization. Are there features highly correlated with mpg?

[3]:
```
import seaborn as sns
import matplotlib.pyplot as plt

corr_matrix = df.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=1,␣
↪linecolor='black')
plt.title('Correlation Coefficient Matrix')
plt.show()
```



To find if there are features highly correlated with mpg I had to do the following steps. they will be organized from highest to lowest correlation:

[4]:
```
# Extract correlations of 'mpg' with other features
mpg_correlations = corr_matrix['mpg'].drop('mpg')  # Drop mpg itself

# Sort correlations by absolute values in descending order
mpg_correlations = mpg_correlations.abs().sort_values(ascending=False)

# Print features highly correlated with mpg
print("Features highly correlated with MPG:")
print(mpg_correlations)
```

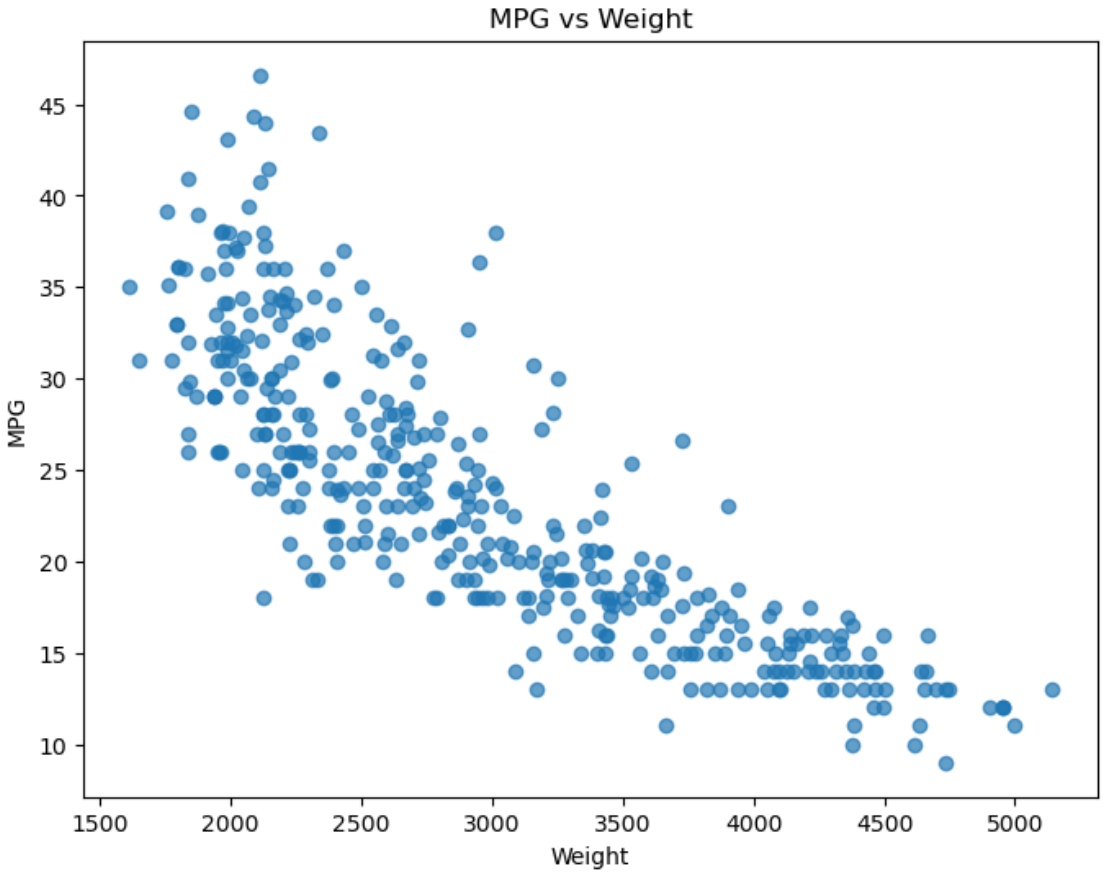
```
Features highly correlated with MPG:
weight          0.831741
displacement    0.804203
cylinders       0.775396
horsepower      0.771437
```

```
model year      0.579267
origin_3        0.442174
acceleration    0.420289
origin_2        0.259022
Name: mpg, dtype: float64
```

[ ]:

1.2.4 4. Plot mpg versus weight. Analyze this graph and explain how it relates to the corresponding correlation coefficient.

[5]:
```
plt.figure(figsize=(8, 6))
plt.scatter(df['weight'], df['mpg'], alpha=0.7)
plt.title('MPG vs Weight')
plt.xlabel('Weight')
plt.ylabel('MPG')
plt.show()
```



This correlation is negative, which means, that the heavier the car the lower the MPG.

[ ]:

1.2.5 5. Randomly split the data into 80% training data and 20% test data, where your target is mpg.

[6]:
```
from sklearn.model_selection import train_test_split

# Features all columns except mpg
X = df.drop('mpg', axis=1)
# Targets mpg
y = df['mpg']

# Split the data into training and test sets (80% training, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Display the shape of the resulting datasets
print("Training set - X:", X_train.shape, " y:", y_train.shape)
print("Test set - X:", X_test.shape, " y:", y_test.shape)
```

```
Training set - X: (318, 8)  y: (318,)
Test set - X: (80, 8)  y: (80,)
```

By doing the training and test with and without mpg, it provides me a data for a better comparison. The training set tells me that there are 318 samples and 8 attributes (features). The test set tells me there are 80 samples and 8 attributes (features).

[ ]:

1.2.6 6. Train an ordinary linear regression on the training data.

[7]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import numpy as np

# Instantiate and train linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

[7]:
```
LinearRegression()
```

[ ]:

1.2.7 7. Calculate R2, RMSE, and MAE on both the training and test sets and interpret your results.

[8]:
```
def evaluate_model(model, X_train, y_train, X_test, y_test):
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)

    r2_train = r2_score(y_train, y_pred_train)
    r2_test = r2_score(y_test, y_pred_test)

    rmse_train = np.sqrt(mean_squared_error(y_train, y_pred_train))
    rmse_test = np.sqrt(mean_squared_error(y_test, y_pred_test))

    mae_train = mean_absolute_error(y_train, y_pred_train)
    mae_test = mean_absolute_error(y_test, y_pred_test)

    print("Training Set:")
    print(f"R2 Score: {r2_train:.4f}, RMSE: {rmse_train:.2f}, MAE: {mae_train:.
 ↪2f}")
    print("Test Set:")
    print(f"R2 Score: {r2_test:.4f}, RMSE: {rmse_test:.2f}, MAE: {mae_test:.
 ↪2f}")

# Evaluate linear regression model
evaluate_model(model, X_train, y_train, X_test, y_test)
```

```
Training Set:
R2 Score: 0.8188, RMSE: 3.37, MAE: 2.61
Test Set:
R2 Score: 0.8449, RMSE: 2.89, MAE: 2.29
```

[ ]:
```

```

1.2.8 8. Pick another regression model and repeat the previous two steps. Note: Do NOT choose logistic regression as it is more like a classification model.

[9]:
```
from sklearn.ensemble import RandomForestRegressor

# Random forest regression model
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)

# Evaluate random forest regression model
evaluate_model(rf_model, X_train, y_train, X_test, y_test)
```

```
Training Set:
R2 Score: 0.9810, RMSE: 1.09, MAE: 0.75
Test Set:
```

```
R2 Score: 0.9106, RMSE: 2.19, MAE: 1.63
```

I decided to do RandomForestRegressor because: 1. It is flexible and robust. 2. Is good for non-linear relationships. 3. I provides insights into features that contribute most to the prediction. 4. Can handle missing values. 5. Good performance across datasets.

[ ]: