Bilenkin550Week4_Exercise_4.2

April 5, 2025

4.2 Exercise: Predicting Fuel Efficiency

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

[1]:
```python
import pandas as pd

# Loading dataset from local folder
df = pd.read_csv(r'C:\Users\maxim\OneDrive\Desktop\BU\DSC 550\auto-mpg.csv')

# Displaying the first few rows to confirm the dataset loaded correctly
df.head()
```

[1]:
```
    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
```

2. Begin by prepping the data for modeling:

a. Remove the car name column.

b. The horsepower column values likely imported as a string data type. Figure out why and replace any strings with the column mean.

c. Create dummy variables for the origin column.

[2]:
```python
# a) Dropping the 'car name' column
df = df.drop('car name', axis=1)

# b) Converting 'horsepower' to numeric and handle non-numeric entries
df['horsepower'] = pd.to_numeric(df['horsepower'], errors='coerce')
```

```
# Replacing missing values (NaN) in horsepower with the column mean
df['horsepower'] = df['horsepower'].fillna(df['horsepower'].mean())

# c) Creating dummy variables for the 'origin' column
df = pd.get_dummies(df, columns=['origin'], prefix='origin', drop_first=True)

# Converting boolean dummy columns to integers because normally machine␣
↪learning models expect numerical input such as 0s and 1s instead of True/
↪False
origin_cols = [col for col in df.columns if col.startswith('origin_')]
df[origin_cols] = df[origin_cols].astype(int)

# Displaying the first 5 rows of the cleaned dataset
df.head()
```

[2]:

```
    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_2  origin_3
0          70         0         0
1          70         0         0
2          70         0         0
3          70         0         0
4          70         0         0
```

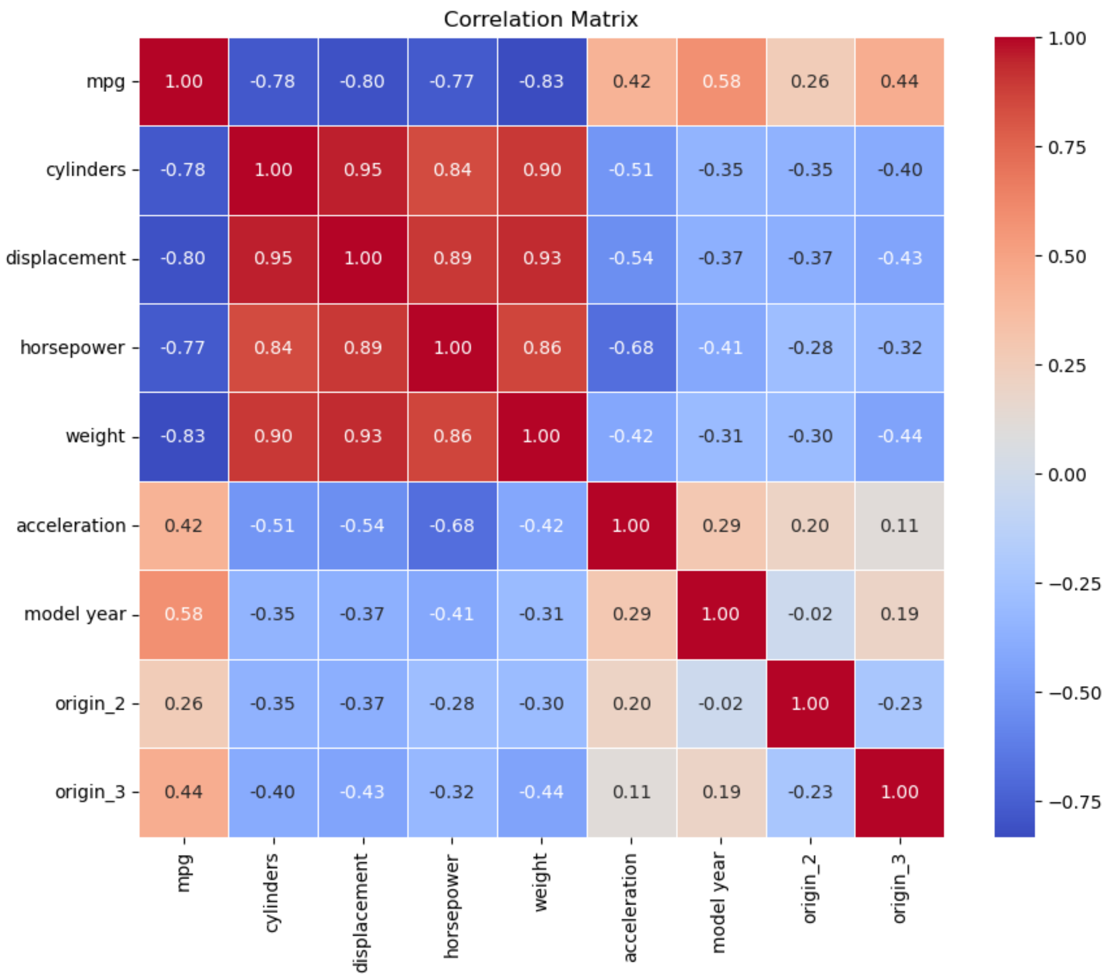
3. Create a correlation coefficient matrix and/or visualization. Are there features highly correlated with mpg?

[3]:

```
# a) Creating the correlation matrix
correlation_matrix = df.corr()

# b) Visualizing the correlation matrix using a heatmap
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',␣
↪linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin_2	origin_3
mpg	1.00	-0.78	-0.80	-0.77	-0.83	0.42	0.58	0.26	0.44
cylinders	-0.78	1.00	0.95	0.84	0.90	-0.51	-0.35	-0.35	-0.40
displacement	-0.80	0.95	1.00	0.89	0.93	-0.54	-0.37	-0.37	-0.43
horsepower	-0.77	0.84	0.89	1.00	0.86	-0.68	-0.41	-0.28	-0.32
weight	-0.83	0.90	0.93	0.86	1.00	-0.42	-0.31	-0.30	-0.44
acceleration	0.42	-0.51	-0.54	-0.68	-0.42	1.00	0.29	0.20	0.11
model year	0.58	-0.35	-0.37	-0.41	-0.31	0.29	1.00	-0.02	0.19
origin_2	0.26	-0.35	-0.37	-0.28	-0.30	0.20	-0.02	1.00	-0.23
origin_3	0.44	-0.40	-0.43	-0.32	-0.44	0.11	0.19	-0.23	1.00

Looking at the heatmap, we can see that there are several features that have very strong negative correlations, and to a lesser extent, positive correlations. The correlation between mpg and cylinders is -0.78. Similarly, mpg and displacement have a strong negative correlation of -0.80. The correlation between mpg and horsepower is -0.77, and mpg and weight stands at -0.83. This makes sense because as cylinders, displacement, horsepower, and weight increase, the engine typically needs to consume more fuel, which in turn decreases mpg.

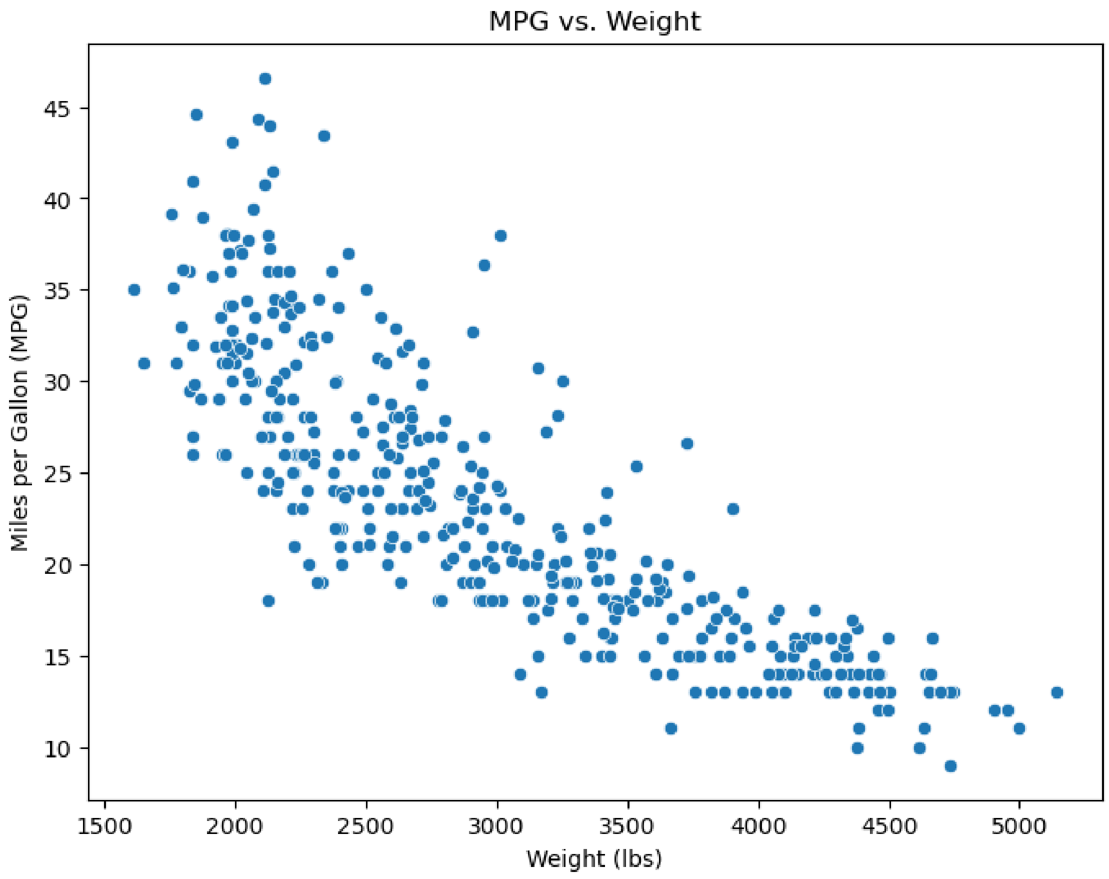
On the other hand, the highest positive correlation of 0.58 is between mpg and model year. This likely suggests that newer cars are implementing more advanced technology, making them more fuel-efficient. As the model year increases, cars tend to become more efficient, leading to higher mpg.

The other positive correlations are substantially lower than 0.58, so they are not as significant in this analysis.

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

[4]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting mpg vs. weight
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='weight', y='mpg')
plt.title('MPG vs. Weight')
plt.xlabel('Weight (lbs)')
plt.ylabel('Miles per Gallon (MPG)')
plt.show()
```



As observed from the scatter plot, most of the data points show a downward slope, indicating that as the weight of a car increases, the miles per gallon (mpg) tends to decrease. This aligns with the strong negative correlation of -0.83 between mpg and weight, confirming that heavier cars generally have lower fuel efficiency.

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

[5]:
```python
from sklearn.model_selection import train_test_split
```

```
# Defining target and features
X = df.drop('mpg', axis=1)  # Features
y = df['mpg']  # Target variable

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

# Displaying the shapes of the training and test sets
print(f"Training set size: {X_train.shape[0]} samples")
print(f"Test set size: {X_test.shape[0]} samples")
```

```
Training set size: 318 samples
Test set size: 80 samples
```

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

[6]:
```
from sklearn.linear_model import LinearRegression

# Initializing the model
model = LinearRegression()

# Training the model on the training data
model.fit(X_train, y_train)

# Displaying the intercept
print(f"Intercept: {model.intercept_:.3f}")

# Pairing the coefficients with the feature names
coefficients = model.coef_
features = X_train.columns  # Assuming X_train is a pandas DataFrame

# Displaying each coefficient with its corresponding feature (formatted to 3␣
 ↪decimal places)
for feature, coef in zip(features, coefficients):
    print(f"Coefficient for {feature}: {coef:.3f}")
```

```
Intercept: -22.067
Coefficient for cylinders: -0.164
Coefficient for displacement: 0.020
Coefficient for horsepower: -0.013
Coefficient for weight: -0.007
Coefficient for acceleration: 0.073
Coefficient for model year: 0.827
Coefficient for origin_2: 2.939
Coefficient for origin_3: 2.653
```

For this step, I trained an ordinary linear regression model using 80% of the dataset for training.

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

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

# Predicting values for the training and test set
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Calculating R² for both training and test sets
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

# Calculating RMSE for both training and test sets
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Calculating MAE for both training and test sets
mae_train = mean_absolute_error(y_train, y_train_pred)
mae_test = mean_absolute_error(y_test, y_test_pred)

# Displaying the results
print(f"R² (Training): {r2_train:.3f}")
print(f"R² (Test): {r2_test:.3f}")
print(f"RMSE (Training): {rmse_train:.3f}")
print(f"RMSE (Test): {rmse_test:.3f}")
print(f"MAE (Training): {mae_train:.3f}")
print(f"MAE (Test): {mae_test:.3f}")
```

```
R² (Training): 0.819
R² (Test): 0.845
RMSE (Training): 3.370
RMSE (Test): 2.888
MAE (Training): 2.605
MAE (Test): 2.288
```

The R^2 (Training): 0.819 means that approximately 81.9% of the variance in the target variable (mpg) is explained by the model using the training data. We can conclude that the model fits the training data very well.

The R^2 (Test): 0.845 is even higher than the training R^2 at 84.5%. This indicates that the model generalizes very well to unseen data. Normally, we would expect a lower R^2 when moving from training to testing if the model overfits, but in this case, it performs slightly better on the test data. This suggests a good balance between fitting the training data and generalizing to new data.

The RMSE (Training): 3.370 means that the deviation between the actual and predicted results is about 3.37 miles per gallon.

The RMSE (Test): 2.888 indicates that the test model predicts slightly better than the training model. Its error is lower (2.88 mpg vs. 3.37 mpg for the training model). This means that the test model's predicted values are closer to the actual values.

The MAE (Training): 2.605 means that on average, the training model's predictions deviate approximately 2.61 mpg from the actual values in the training set.

The MAE (Test): 2.288 is slightly lower than the training set, suggesting that the model's predictions are marginally better on average compared to the training set.

In conclusion, the model performs well, with high R² values for both training and testing. Additionally, both RMSE and MAE values are low for both the training and testing models, indicating that the deviation from true values is very small. This makes the model a powerful predictor.

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.

[8]:
```python
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Initializing the Ridge regression model
ridge_model = Ridge()

# Training the Ridge model on the training data
ridge_model.fit(X_train, y_train)

# Making predictions on the training and test data
y_train_pred = ridge_model.predict(X_train)
y_test_pred = ridge_model.predict(X_test)

# Calculating R^2, RMSE, and MAE for training and test sets
r2_train_ridge = r2_score(y_train, y_train_pred)
r2_test_ridge = r2_score(y_test, y_test_pred)

rmse_train_ridge = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test_ridge = np.sqrt(mean_squared_error(y_test, y_test_pred))

mae_train_ridge = mean_absolute_error(y_train, y_train_pred)
mae_test_ridge = mean_absolute_error(y_test, y_test_pred)

# Displaying the results
print(f"R² (Training): {r2_train_ridge:.3f}")
print(f"R² (Test): {r2_test_ridge:.3f}")
print(f"RMSE (Training): {rmse_train_ridge:.3f}")
print(f"RMSE (Test): {rmse_test_ridge:.3f}")
print(f"MAE (Training): {mae_train_ridge:.3f}")
print(f"MAE (Test): {mae_test_ridge:.3f}")
```

```
R² (Training): 0.819
R² (Test): 0.845
RMSE (Training): 3.371
RMSE (Test): 2.889
MAE (Training): 2.605
MAE (Test): 2.292
```

The results from the Ridge Regression model are as follows:

R^2 (Training): 0.819 means the model explains 81.9% of the variance in the training data. This result is similar to that of the ordinary linear regression model.

R^2 (Test): 0.845 indicates that the model performs slightly better on the test data, explaining 84.5% of the variance.

RMSE (Training): 3.371 means the deviation between actual and predicted values in the training set is about 3.37 mpg (miles per gallon). This value is very close to that from the linear regression model.

RMSE (Test): 2.889 indicates the predicted values in the test set have an average error of 2.89 mpg from the actual values, suggesting good predictive accuracy.

MAE (Training): 2.605 shows that, on average, the predictions for the training set deviate by about 2.61 mpg.

MAE (Test): 2.292 indicates that the test set predictions have an even smaller average error of 2.29 mpg, suggesting better performance on unseen data.

We can conclude that the performance of the Ridge Regression model is not significantly different from the linear regression model—likely because the data did not suffer from high multicollinearity or overfitting.

Overall, the model does a good job generalizing to unseen data, as reflected by the similar or slightly improved test set performance compared to the training set.