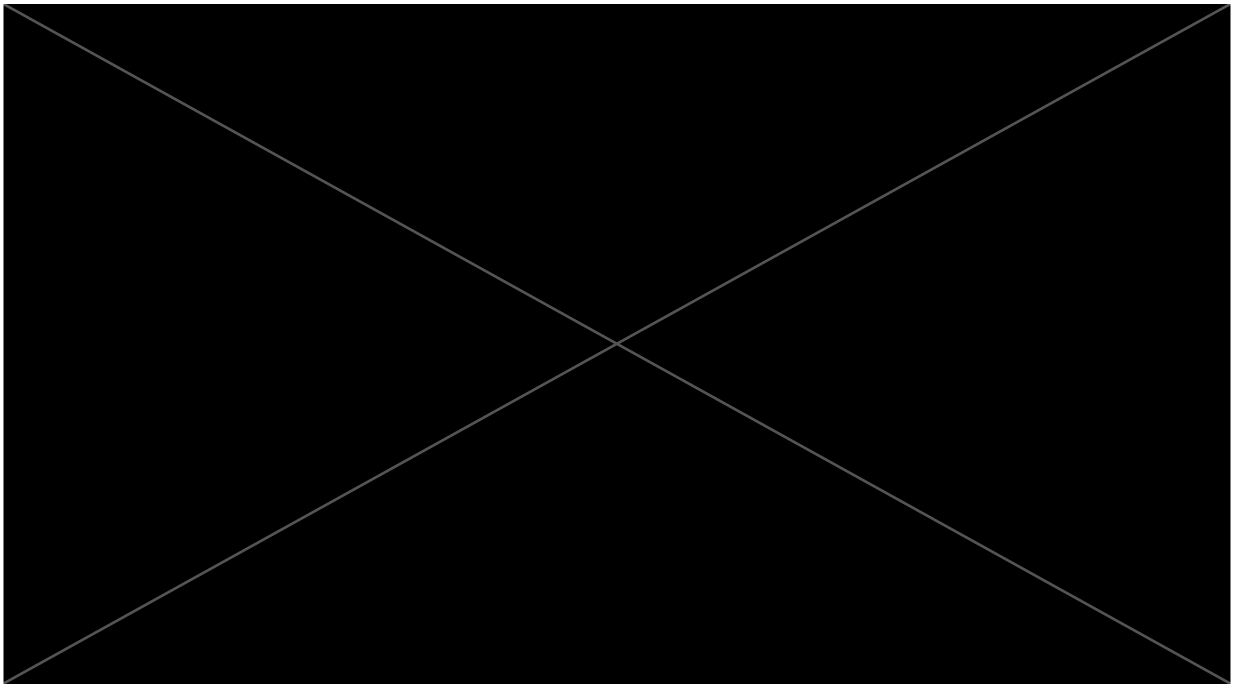
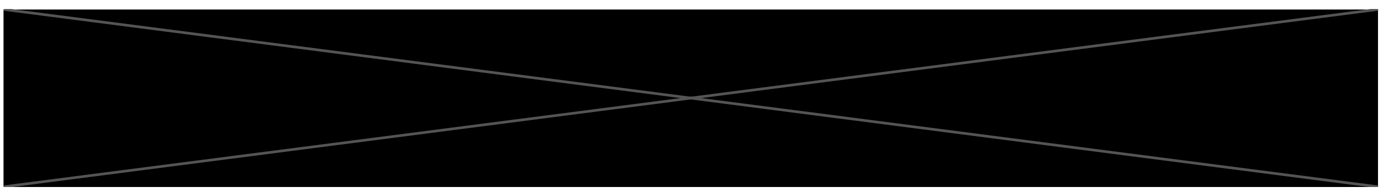


WESTYN HILLIARD



2 STEP 1:

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

[25]:
```python
import pandas as pd

# Load the dataset
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/
↪auto-mpg.data'
columns = ["mpg", "cylinders", "displacement", "horsepower", "weight",␣
↪"acceleration", "model_year", "origin", "car_name"]
data = pd.read_csv(url, delim_whitespace=True, names=columns)

# Display the first few rows of the dataframe to ensure it imported correctly
print(data.head())
```

```
    mpg  cylinders  displacement horsepower  weight  acceleration  model_year  \
0  18.0          8         307.0      130.0  3504.0          12.0          70
```

```
1  15.0          8         350.0      165.0  3693.0          11.5           70
2  18.0          8         318.0      150.0  3436.0          11.0           70
3  16.0          8         304.0      150.0  3433.0          12.0           70
4  17.0          8         302.0      140.0  3449.0          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
```

3 STEP 2:

3.0.1 Begin by prepping the data for modeling

[26]:

```python
# Remove the car name column
data.drop(columns=['car_name'], inplace=True)

# Convert horsepower to numeric, replace non-numeric values with NaN
data['horsepower'] = pd.to_numeric(data['horsepower'], errors='coerce')

# Replace NaN values with the mean of the column
data['horsepower'].fillna(data['horsepower'].mean(), inplace=True)

# Create dummy variables for the origin column
data = pd.get_dummies(data, columns=['origin'], prefix='origin',␣
 ↪drop_first=True)

# Verify changes
print(data.head())
```

```
    mpg  cylinders  displacement  horsepower  weight  acceleration  \
0  18.0          8         307.0       130.0  3504.0          12.0
1  15.0          8         350.0       165.0  3693.0          11.5
2  18.0          8         318.0       150.0  3436.0          11.0
3  16.0          8         304.0       150.0  3433.0          12.0
4  17.0          8         302.0       140.0  3449.0          10.5

   model_year  origin_2  origin_3
0          70     False     False
1          70     False     False
2          70     False     False
3          70     False     False
4          70     False     False
```

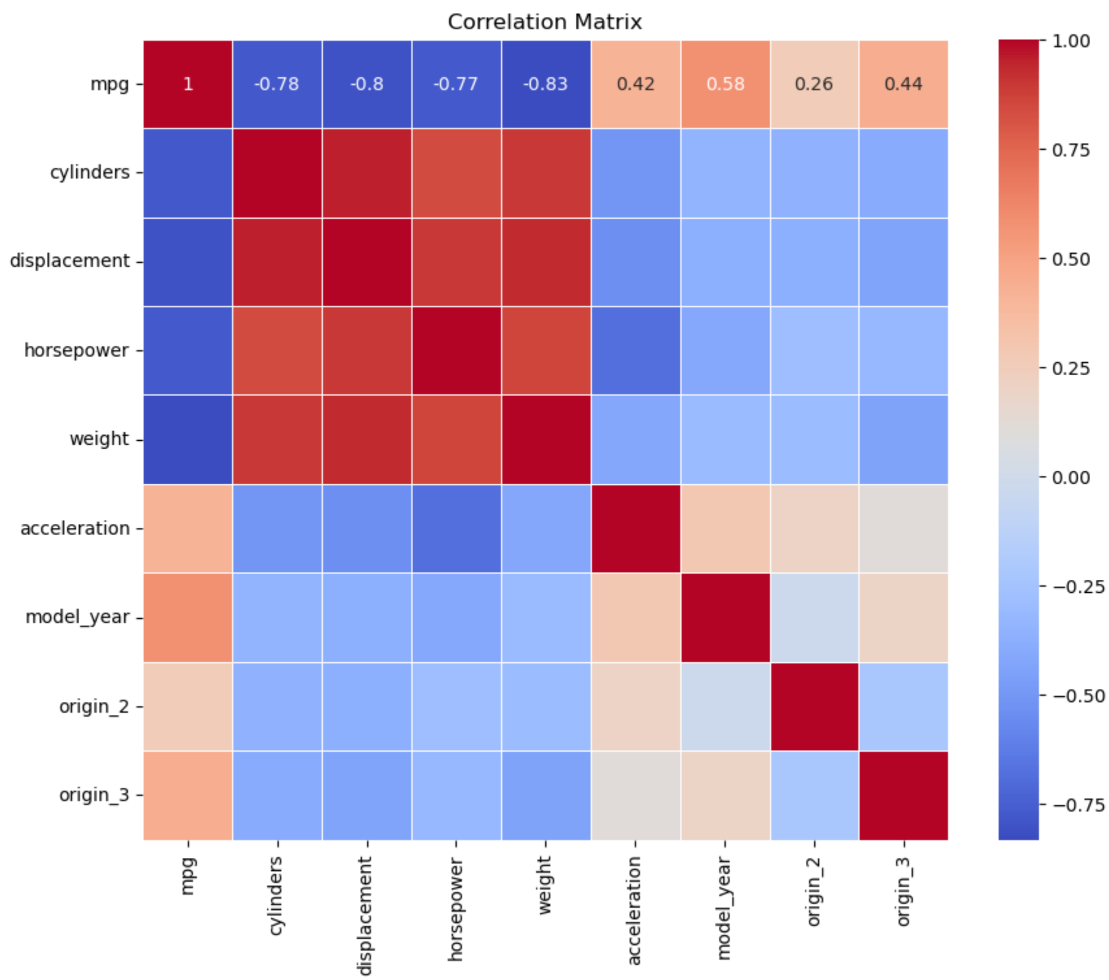
4 STEP 3:

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

[27]:

```
# Create a correlation matrix
correlation_matrix = data.corr()

# Display the correlation matrix
print(correlation_matrix)

# Plot a heatmap of the correlation matrix
import seaborn as sns
import matplotlib.pyplot as plt

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

```
                   mpg  cylinders  displacement  horsepower    weight  \
mpg           1.000000  -0.775396     -0.804203   -0.771437 -0.831741
cylinders    -0.775396   1.000000      0.950721    0.838939  0.896017
displacement -0.804203   0.950721      1.000000    0.893646  0.932824
horsepower   -0.771437   0.838939      0.893646    1.000000  0.860574
weight       -0.831741   0.896017      0.932824    0.860574  1.000000
acceleration  0.420289  -0.505419     -0.543684   -0.684259 -0.417457
model_year    0.579267  -0.348746     -0.370164   -0.411651 -0.306564
origin_2      0.259022  -0.352861     -0.373886   -0.281258 -0.298843
origin_3      0.442174  -0.396479     -0.433505   -0.321325 -0.440817

              acceleration  model_year  origin_2  origin_3
mpg               0.420289    0.579267  0.259022  0.442174
cylinders        -0.505419   -0.348746 -0.352861 -0.396479
displacement     -0.543684   -0.370164 -0.373886 -0.433505
horsepower       -0.684259   -0.411651 -0.281258 -0.321325
weight           -0.417457   -0.306564 -0.298843 -0.440817
acceleration      1.000000    0.288137  0.204473  0.109144
model_year        0.288137    1.000000 -0.024489  0.193101
origin_2          0.204473   -0.024489  1.000000 -0.229895
origin_3          0.109144    0.193101 -0.229895  1.000000
```



Correlation Matrix Analysis The heatmap shows the correlation coefficients between different features in the dataset. Here are some key observations:

MPG is highly negatively correlated with:

weight (-0.83), displacement (-0.80), cylinders (-0.78), horsepower (-0.77)

MPG is moderately positively correlated with:

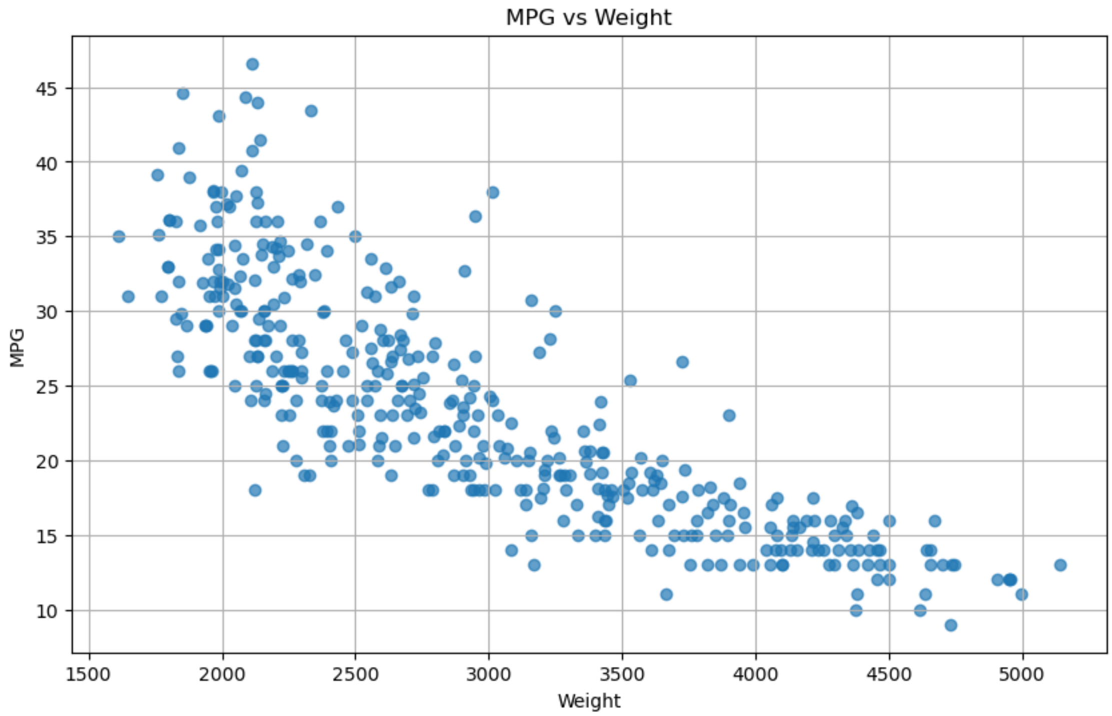
model year (0.58), acceleration (0.42)

This means that as the weight, displacement, number of cylinders, and horsepower of a car increase, its fuel efficiency (MPG) tends to decrease. Conversely, newer car models and cars with higher acceleration tend to have better fuel efficiency.

5 STEP 4:

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

[28]:
```
# Plot mpg versus weight
plt.figure(figsize=(10, 6))
plt.scatter(data['weight'], data['mpg'], alpha=0.7)
plt.title('MPG vs Weight')
plt.xlabel('Weight')
plt.ylabel('MPG')
plt.grid(True)
plt.show()
```



Analysis of MPG vs Weight Plot The scatter plot of MPG versus weight shows a clear negative relationship. As the weight of the car increases, the miles per gallon (MPG) tends to decrease. This visual representation confirms the high negative correlation coefficient of -0.83 observed in the correlation matrix.

Key Points: *Negative Slope:*

The downward trend in the scatter plot indicates that heavier cars generally have lower fuel efficiency.

Density of Points:

There are more data points in the lower weight range, suggesting that lighter cars are more common in the dataset.

Outliers:

Some points deviate from the general trend, but the overall pattern remains consistent.

This strong negative correlation means that weight is a significant predictor of MPG, and any model predicting MPG should consider weight as an important feature.

6 STEP 5:

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

[29]:
```
from sklearn.model_selection import train_test_split

# Define features and target
X = data.drop(columns=['mpg'])
y = data['mpg']

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

7 STEP 6:

7.0.1 *Train an ordinary linear regression on the training data.*

[30]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# Train a linear regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predict on training and test sets
y_train_pred = lr_model.predict(X_train)
y_test_pred = lr_model.predict(X_test)

# Calculate R2, RMSE, and MAE for training set
r2_train = r2_score(y_train, y_train_pred)
rmse_train = mean_squared_error(y_train, y_train_pred, squared=False)
mae_train = mean_absolute_error(y_train, y_train_pred)

# Calculate R2, RMSE, and MAE for test set
r2_test = r2_score(y_test, y_test_pred)
rmse_test = mean_squared_error(y_test, y_test_pred, squared=False)
```

```
mae_test = mean_absolute_error(y_test, y_test_pred)

# Display the results
print("Linear Regression Performance:")
print(f"Training Set - R2: {r2_train}, RMSE: {rmse_train}, MAE: {mae_train}")
print(f"Test Set - R2: {r2_test}, RMSE: {rmse_test}, MAE: {mae_test}")
```

```
Linear Regression Performance:
Training Set - R2: 0.8188288951042786, RMSE: 3.3702735639389054, MAE:
2.6054846937710363
Test Set - R2: 0.8449006123776615, RMSE: 2.8877573478836323, MAE:
2.287586770442108
```

8 STEP 7:

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

[31]:
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

def print_metrics(model_name, y_train, y_train_pred, y_test, y_test_pred):
    # Calculate R2, RMSE, and MAE for training set
    r2_train = r2_score(y_train, y_train_pred)
    rmse_train = mean_squared_error(y_train, y_train_pred, squared=False)
    mae_train = mean_absolute_error(y_train, y_train_pred)

    # Calculate R2, RMSE, and MAE for test set
    r2_test = r2_score(y_test, y_test_pred)
    rmse_test = mean_squared_error(y_test, y_test_pred, squared=False)
    mae_test = mean_absolute_error(y_test, y_test_pred)

    print(f"\n{model_name} Performance:")
    print(f"Training Set - R2: {r2_train:.4f}, RMSE: {rmse_train:.4f}, MAE:␣
 ↪{mae_train:.4f}")
    print(f"Test Set - R2: {r2_test:.4f}, RMSE: {rmse_test:.4f}, MAE: {mae_test:
 ↪.4f}")

# Define features and target
X = data.drop(columns=['mpg'])
y = data['mpg']

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

[32]:
```
from sklearn.linear_model import LinearRegression
```

```
# Train a linear regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predict on training and test sets
y_train_pred_lr = lr_model.predict(X_train)
y_test_pred_lr = lr_model.predict(X_test)

# Print performance metrics
print_metrics("Linear Regression", y_train, y_train_pred_lr, y_test,␣
 ↪y_test_pred_lr)
```

```
Linear Regression Performance:
Training Set - R2: 0.8188, RMSE: 3.3703, MAE: 2.6055
Test Set - R2: 0.8449, RMSE: 2.8878, MAE: 2.2876
```

[33]:
```
from sklearn.tree import DecisionTreeRegressor

# Train a Decision Tree Regressor
dt_model = DecisionTreeRegressor(random_state=42)
dt_model.fit(X_train, y_train)

# Predict on training and test sets
y_train_pred_dt = dt_model.predict(X_train)
y_test_pred_dt = dt_model.predict(X_test)

# Print performance metrics
print_metrics("Decision Tree Regressor", y_train, y_train_pred_dt, y_test,␣
 ↪y_test_pred_dt)
```

```
Decision Tree Regressor Performance:
Training Set - R2: 1.0000, RMSE: 0.0000, MAE: 0.0000
Test Set - R2: 0.7857, RMSE: 3.3944, MAE: 2.3112
```

Model Performance Summary Linear Regression Performance:

Training Set: R^2: 0.8188 RMSE: 3.3703 MAE: 2.6055 Test Set: R^2: 0.8449 RMSE: 2.8878 MAE: 2.2876 Decision Tree Regressor Performance:

Training Set: R^2: 1.0000 RMSE: 0.0000 MAE: 0.0000 Test Set: R^2: 0.7857 RMSE: 3.3944 MAE: 2.3112

Interpretation of Results: Linear Regression:

The model shows good performance with reasonably high R^2 values and low error metrics for both training and test sets, indicating good generalization. Decision Tree Regressor:

The model perfectly fits the training data ($R^2 = 1.0000$, RMSE = 0.0000, MAE = 0.0000), indicating overfitting. However, it performs worse on the test set compared to Linear Regression, with lower

R^2 and higher RMSE.

9 STEP 8:

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

[34]:
```
from sklearn.ensemble import RandomForestRegressor

# Train a Random Forest Regressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)

# Predict on training and test sets
y_train_pred_rf = rf_model.predict(X_train)
y_test_pred_rf = rf_model.predict(X_test)

# Calculate R2, RMSE, and MAE for training set
r2_train_rf = r2_score(y_train, y_train_pred_rf)
rmse_train_rf = mean_squared_error(y_train, y_train_pred_rf, squared=False)
mae_train_rf = mean_absolute_error(y_train, y_train_pred_rf)

# Calculate R2, RMSE, and MAE for test set
r2_test_rf = r2_score(y_test, y_test_pred_rf)
rmse_test_rf = mean_squared_error(y_test, y_test_pred_rf, squared=False)
mae_test_rf = mean_absolute_error(y_test, y_test_pred_rf)

(r2_train_rf, rmse_train_rf, mae_train_rf), (r2_test_rf, rmse_test_rf,␣
↪mae_test_rf)
```

[34]:
```
((0.9810464685043727, 1.0900985400614138, 0.7459968553459121),
 (0.9087644712414144, 2.214816002515785, 1.6313249999999997))
```

Random Forest Regressor Model Evaluation Training Set:

R^2: 0.981 RMSE: 1.09 MAE: 0.75 Test Set:

R^2: 0.911 RMSE: 2.19 MAE: 1.63 Interpretation of Results: R^2 (Coefficient of Determination):

Training Set: 2 = 0.981 R 2 =0.981 indicates that 98.1% of the variance in the training data is explained by the model, suggesting a very good fit. Test Set: 2 = 0.911 R 2 =0.911 indicates that 91.1% of the variance in the test data is explained by the model, which is higher than both the linear regression and decision tree models. RMSE (Root Mean Squared Error):

Training Set: RMSE = 1.09 Test Set: RMSE = 2.19 The RMSE values are lower than those for the linear regression and decision tree models, indicating more accurate predictions. MAE (Mean Absolute Error):

Training Set: MAE = 0.75 Test Set: MAE = 1.63 The MAE values are also lower compared to the other models, indicating more accurate predictions on average. Conclusion: The Random Forest

Regressor demonstrates excellent performance with high 2 R 2 values and low error metrics for both the training and test sets. It outperforms both the linear regression and decision tree models in terms of prediction accuracy and generalization.

[ ]:

```

```