Linear Regression Model to Predict Car's MPG

Goals:

1. Display an understanding of the steps taken to create and implement a linear regression model.
2. Create visual aids to correspond and effectively communicate findings
3. Create a model with at minimum 0.80 r2 score

In [ ]:
```
#importing necessary libraries
import pandas as pd #pandas and numpy for data manipulation
import numpy as np
from sklearn.model_selection import train_test_split #train test split to organize data in training and test sets
from sklearn.linear_model import LinearRegression #Linear regression the algorithm i'll be using
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error #each helps us gauge the effectiveness of our model
from sklearn.preprocessing import MinMaxScaler #used to normalize data
import seaborn as sns #seaborn and matplotlib used to visualize data
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

importing dataset and converting to data frame

In [ ]:
```
File_Path = pd.read_excel("C:\\Your\\File\\OneDrive\\Cars_MPG.xlsx")
car_data = File_Path
```

EDA(exploratory data analysis/Cleaning)

In [ ]:
```
print(car_data.shape) #understanding the length of both rows and columns
print(car_data.dtypes) #getting the data type of all columns
print(car_data.describe) #getting a high level overview of the data per column
```

```
(398, 9)
mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight            int64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
<bound method NDFrame.describe of       mpg  cylinders  displacement horsepower  weight  acceleration  \
0    18.0          8         307.0        130    3504          12.0   
1    15.0          8         350.0        165    3693          11.5   
2    18.0          8         318.0        150    3436          11.0   
3    16.0          8         304.0        150    3433          12.0   
4    17.0          8         302.0        140    3449          10.5   
..    ...        ...           ...        ...     ...           ...   
393  27.0          4         140.0         86    2790          15.6   
394  44.0          4          97.0         52    2130          24.6   
395  32.0          4         135.0         84    2295          11.6   
396  28.0          4         120.0         79    2625          18.6   
397  31.0          4         119.0         82    2720          19.4   

     model year  origin                   car name  
0            70       1  chevrolet chevelle malibu  
1            70       1          buick skylark 320  
2            70       1         plymouth satellite  
3            70       1              amc rebel sst  
4            70       1                ford torino  
..          ...     ...                        ...  
393          82       1            ford mustang gl  
394          82       2                  vw pickup  
395          82       1              dodge rampage  
396          82       1                ford ranger  
397          82       1                 chevy s-10  

[398 rows x 9 columns]>
```

Two things stood out, 1 the the difference between the max weight and minimum weight is pretty big ill normalize this. 2 There was an issue with the horsepower column I expected a numeric value but got a object so seems like theres a mixture of strings and numbers perhaps placeholders for missing values. I'll create a function to check for any missing values in the horsepower column.

In [ ]:
```
#function that checks for missing values and adds them to a list
def find_missing_values(df, column):
  missing_values = []
  for value in df[column]:
    if type(value) != int and type(value) != float:
      missing_values.append(value)
    else:
      continue
  return(missing_values)
```

Using our function and finding the missing values in horsepower

In [ ]:
```
horsepower_missing_values = find_missing_values(car_data, 'horsepower')
print(horsepower_missing_values)
```

```
['?', '?', '?', '?', '?', '?']
```

Now that we've found them i'll write a function with the purpose of replacing and dropping them.

In [ ]:
```
def replace_and_drop_missing_values(df, column):
    df[column].replace('?', np.nan, inplace = True)
    df.dropna(subset= column, inplace = True)
    print(df[column].isna().sum())

replace_and_drop_missing_values(car_data, 'horsepower')
```

```
0
```

Splitting data into training and testings sets followed by normalization and fitting

Now that it's cleaned it's time to split the data into training and testing sets. I identify my x and y variables slightly differently using x1 and y1 respectively because I plan on rerunning the model multiple times its easier to keep track of for me.

In [ ]:
```
x1 = car_data[['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model year', 'origin']]
y1 = car_data['mpg']

x1_train, x1_test, y1_train, y1_test = train_test_split(x1, y1, test_size=0.2, random_state=42)
```

Now that the data is split i'll have to normalize the weight column.

In [ ]:
```
scaler = MinMaxScaler()
x1_train[['weight']] = scaler.fit_transform(x1_train[['weight']])
x1_test[['weight']] = scaler.transform(x1_test[['weight']])
```

Moving on to selecting and training the model.

In [ ]:
```
model = LinearRegression()
model.fit(x1_train, y1_train)
y1_pred = model.predict(x1_test)
```

Plugging new data in to test the model

With the model being trained its time to put it to the test. I'll create a separate dataset to use for testing, convert it to a data frame, scale it as well since we scaled the training/test sets and plug it into our model for prediction.

In [ ]:
```
sample_data = {
  'cylinders' : [8],
  'displacement' : [400],
  'horsepower' : [150],
  'weight' : [3761],
  'acceleration' : [9.5],
  'model year' : [71],
  'origin' : [1]
}
sample_data= pd.DataFrame(sample_data)
sample_data[['weight']] = scaler.transform(sample_data[['weight']])
model1 = model.predict(sample_data)
```

```
print(model1)
```

```
[14.96760324]
```

Analyzing results using MAE/MSE/R2Score and feature importance

The model predicted that the mpg for our theoretical dataset would be 14 mpg. To get a general idea of the general accuracy of the model we can get the r2 score, mean squared error and the mean absolute error.

The r2 score is ranged from 0 to 1 I think of it as a percentage taking the first two numbers after the decimal point.

The mean squared error(mse) is used to determine larger errors such as outliers.

the mean absolute error(mae) is useful to see on average how far the model is off by.

In [ ]:

```
test1_r2score = r2_score(y1_test, y1_pred)
test1_mae = mean_absolute_error(y1_test, y1_pred)
test1_mse = mean_squared_error(y1_test, y1_pred)

print('the r2 score for test 1 {}, it was 79% accurate.'.format(test1_r2score))
print('the mae for test 1 {} typically off by 2.'.format(test1_mae))
print('the mse for test 1 {} and some slight outliers skewing predictions.'.format(test1_mse))
```

```
the r2 score for test 1 0.7901500386760347, it was 79% accurate.
the mae for test 1 2.419780249197452 typically off by 2.
the mse for test 1 10.710864418838385 and some slight outliers skewing predictions.
```

Getting the coefficients will show us the magnitude of each feature allowing us to know how each feature impacts the target.

In [ ]:

```
coefficient = model.coef_
features = x1_train.columns
test1_magnitude= dict(zip(features, coefficient)) #combining the two into a dictionary
test1_magnitude = pd.DataFrame({'features': features, 'coefficient': coefficient}) #manually converting our dictionary into a data frame
print(test1_magnitude)
```

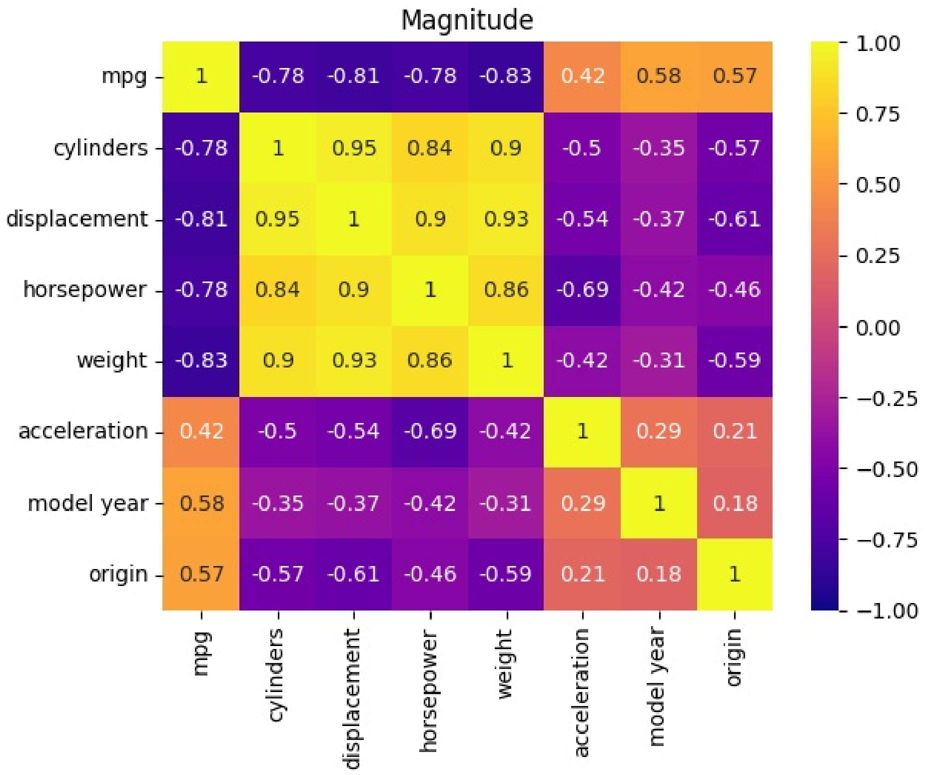
```
       features  coefficient
0     cylinders    -0.345789
1  displacement     0.015109
2    horsepower    -0.021302
3        weight   -21.661512
4  acceleration     0.037950
5    model year     0.767743
6        origin     1.613457
```

Quick view it looks like the weight is the most detrimental to the mpg. The higher the weight the more the mpg decreases. Vice versa, the origin and model year have the most significant positive impact on the mpg. I'll develop a visual aid to further explain the relationship between each feature and the target.

HeatMap to understand feature importance

In [ ]:

```
correlation_graph = car_data.drop(columns=['car name']).corr() #dropping car name since thats a string and wont work for our heat map
plt.title("Magnitude")
plt.xlabel("features")
sns.heatmap(correlation_graph, annot=True, cmap='plasma', vmin=-1, vmax=1, center=0)
plt.show()
```



Retuning model starting with identifying and dropping outliers

Given the mse we can tell we have some outliers, ill remove them and rerun the model.

In [ ]:

```
residual = y1_test - y1_pred
squared_residual = residual ** 2
average_squared_residual = squared_residual.mean()
limit = average_squared_residual + 2 * squared_residual.std()
outliers = squared_residual[squared_residual > limit]
print(outliers)
```

```
394    90.078101
270    78.466818
111    99.138214
Name: mpg, dtype: float64
```

Retraining the model with no outliers

In [ ]:

```
x2_train = x1_train.drop([394, 270, 111], errors='ignore')
x2_test = x1_test.drop([394, 270, 111], errors='ignore')
y2_train = y1_train.drop([394, 270, 111], errors='ignore')
y2_test = y1_test.drop([394, 270, 111], errors='ignore')
```

```
x2 = car_data[['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration']]
y2 = car_data['mpg']

model = LinearRegression()
model.fit(x2_train, y2_train)
y2_pred = model.predict(x2_test)

sample_data2 = {
  'cylinders' : [8],
  'displacement' : [400],
  'horsepower' : [150],
  'weight' : [3761],
  'acceleration' : [9.5],
   'model year' : [71],
  'origin' : [1]
}

sample_data2 = pd.DataFrame(sample_data2)
sample_data2[['weight']] = scaler.transform(sample_data2[['weight']])
model2 = model.predict(sample_data2)
print(model2)
```

```
[14.96760324]
```

Analyzing model 2s performance

No change in the prediction, but going to check the r2_score, mse and mae for a better understanding of what changed.

In [ ]:

```
test2_r2score = r2_score(y2_test, y2_pred)
test2_mae = mean_absolute_error(y2_test, y2_pred)
test2_mse = mean_squared_error(y2_test, y2_pred)

print(test2_r2score)
print(test2_mse)
print(test2_mae)
```

```
0.8372852484237635
7.61151521549652
2.142851632347093
```

The r2 score improved so the predictions are becoming more accurate the mse improved so removing the outliers helped but the mae remained the same there could be some small errors in the dataset throwing the model off. Let's get the coefficients to see if there were any changes there.

In [ ]:

```
model2_coefficient = model.coef_
model2_features = x1_train.columns

model2_magnitude= dict(zip(features, coefficient))
model2_magnitude = pd.DataFrame({'features': features, 'coefficient': coefficient})
print(model2_magnitude)
```

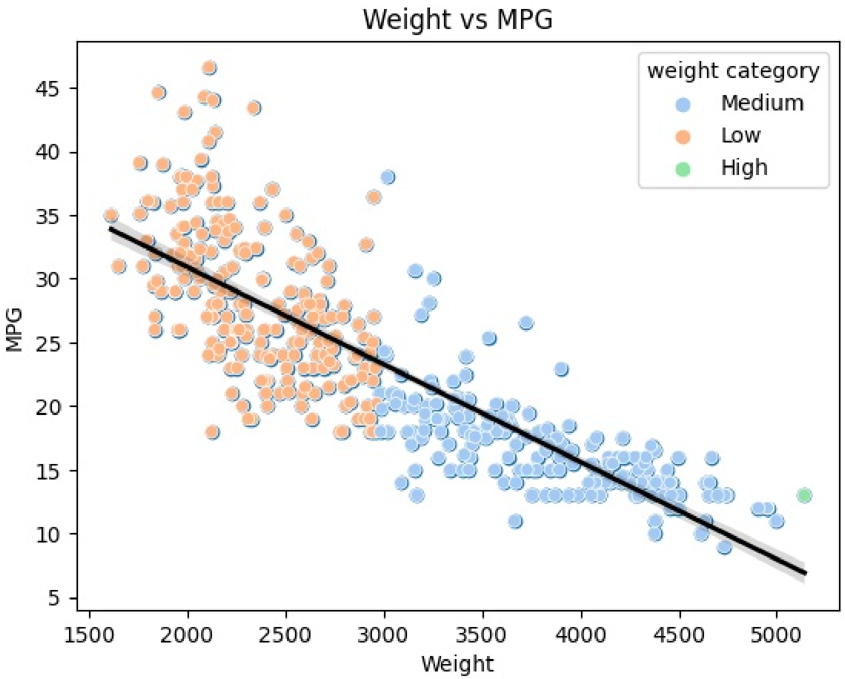
```
        features  coefficient
0      cylinders    -0.345789
1   displacement     0.015109
2     horsepower    -0.021302
3         weight   -21.661512
4   acceleration     0.037950
5     model year     0.767743
6         origin     1.613457
```

Developing a scatter plot to understanding the relationship between weight and mpg

Weight remains the biggest detrimental factor to predicting the mpg we can generate a scatter plot to better visualize the relationship between the two.

In [ ]:

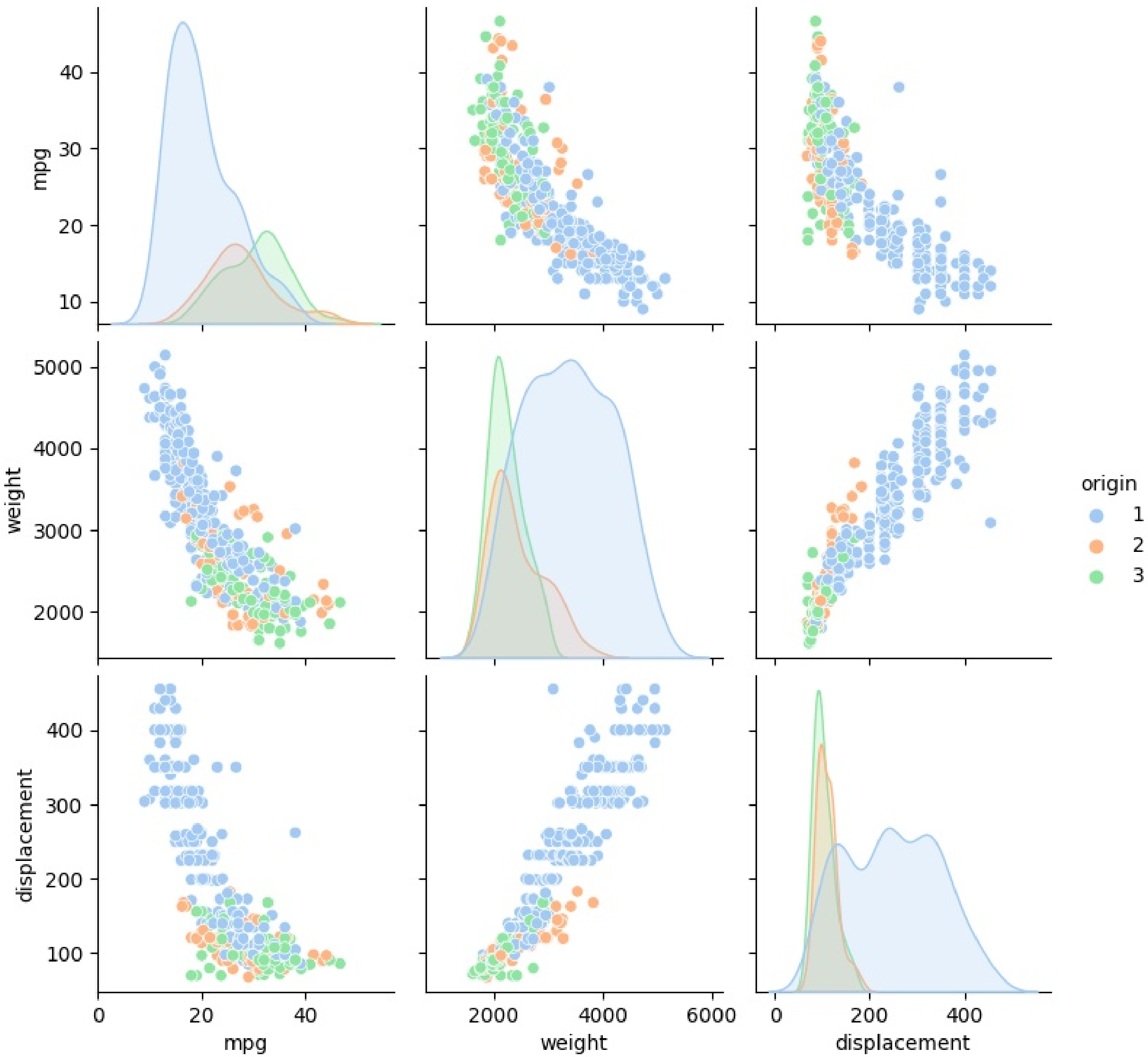
```
plt.scatter(car_data['weight'], car_data['mpg'])

car_data_min_weight = car_data['weight'].min()
car_data_max_weight = car_data['weight'].max()
car_data_avg_weight = car_data['weight'].mean()

conditions = [
    car_data['weight'] <= car_data_avg_weight,
    (car_data['weight'] > car_data_avg_weight) & (car_data['weight'] < car_data_max_weight),
    car_data['weight'] == car_data_max_weight
]
choices = ['Low', 'Medium', 'High']
car_data['weight category'] = np.select(conditions, choices, default='Medium')

custom_palette = {'Low': 'blue', 'Medium': 'green', 'High': 'red'}
sns.scatterplot(x='weight', y='mpg', hue='weight category', data=car_data, palette='pastel')
sns.regplot(x='weight', y='mpg', data=car_data, scatter=False, color='black')
plt.xlabel('Weight')
plt.ylabel('MPG')
plt.title('Weight vs MPG')
plt.show()
```



Two things to consider it looks like we have one weight in our dataset thats considered high which is a bit abnormal but more importantly the linear regression line is moving from left to right in a downward direction which suggests that as the weight increases the mpg decreases.

Developing a pair plot to understand the relationship between our top features and target

In [ ]:
```
sns.pairplot(car_data[['mpg', 'weight', 'displacement', 'origin']],kind='scatter', hue='origin',palette='pastel')
plt.show()
```



I wont go much further with this model because we accomplished the three goals we set out to.

1. Display an understanding of the steps taken to create and implement a linear regression model.

```
 I went through the process step by step.

a. Preprocessing cleaning our data and EDA.

b. Selecting features and targets, splitting into training and testing sets.

c. Fitting and training the model.

d. Testing the model on new data and gauging the results of the model while making adjustments as needed.
```

2. Create visual aids to correspond and effectively communicate findings

a. We used a HeatMap to describe the relationship between all features and the target, the scatter plot was used to dive deeper into understanding the relationship between weight and mpg and lastly the pair plot was used to describe the relationship between our two most impactful features displacement and weight against our target mpg.

3. Create a model with at minimum 0.80 r2 score

a. During the initial run the model was unable to eclipse the 0.80 score I established. Looking at the MSE I was able to tell that there were some outliers skewing the models performance after removing the outliers the r2 score got to a point where I felt comfortable with a score of 0.83. However, the MAE tripled from the original model with a score of 7.61151521549652 which suggests to me that although the outliers have been removed there may be some smaller issues created. I'll be moving on to working with a decision tree model next to see if I can strike a balance between the two and return better results.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js