Name: Kelly Joseph Calvadores
Name: Robvic Matthew C. Gonan
Section:CPE32S3
Activity: Prelim Exam Project
Instructor: Engr. Roman M. Richard

Importing Libraries and Database

```
#Importing Libraries
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
# Importing Database
data = pd.read_csv('/content/drive/MyDrive/EmTech02 - Prelim Exam/scrap-price.csv')
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Database Information

```
print(data.columns.tolist())
```

```
['ID', 'symboling', 'name', 'fueltypes', 'aspiration', 'doornumbers', 'carbody', 'drivewheels', 'enginelocation', 'wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype', 'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke', 'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg', 'price']
```

```
#category
ds = data[(data.fueltypes    == 'diesel')]
gs = data[(data.fueltypes    == 'gas')]
Dtest = data
```

```
data.head()
```

	ID	symboling	name	fueltypes	aspiration	doornumbers	carbody	drivewheels	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg	price
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495.0
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500.0
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500.0
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950.0
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450.0

5 rows × 26 columns

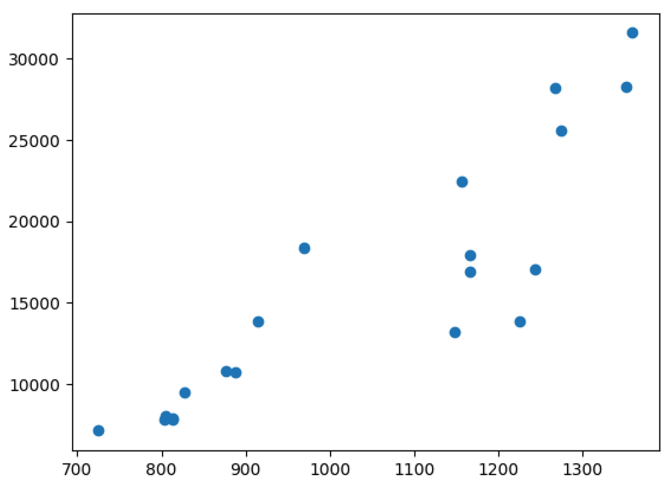
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ID                205 non-null    int64  
 1   symboling         205 non-null    int64  
 2   name              205 non-null    object 
 3   fueltypes         205 non-null    object 
 4   aspiration        205 non-null    object 
 5   doornumbers       205 non-null    object 
 6   carbody           205 non-null    object 
 7   drivewheels       205 non-null    object 
 8   enginelocation    205 non-null    object 
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64  
 14  enginetype        205 non-null    object 
 15  cylindernumber    205 non-null    object 
 16  enginesize        205 non-null    int64  
 17  fuelsystem        205 non-null    object 
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64  
 22  peakrpm           205 non-null    int64  
 23  citympg           205 non-null    int64  
 24  highwaympg        205 non-null    int64  
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

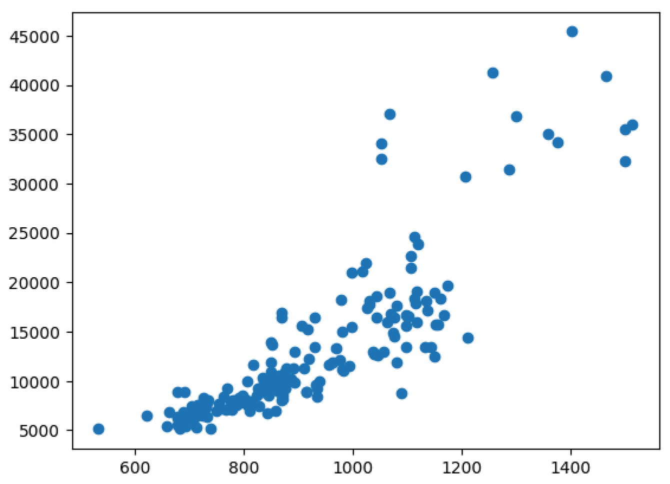
```
data.describe()
```

	ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreratio	stroke	compressionratio	horsepower	peakrpm	citympg	highwaympg	price
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329756	3.255415	10.142537	104.117073	5125.121951	25.219512	30.751220	13276.710571
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313597	3.972040	39.544167	476.985643	6.542142	6.886443	7988.852332
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.000000	4150.000000	13.000000	16.000000	5118.000000
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.600000	70.000000	4800.000000	19.000000	25.000000	7788.000000
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.000000	5200.000000	24.000000	30.000000	10295.000000
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410000	9.400000	116.000000	5500.000000	30.000000	34.000000	16503.000000
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	288.000000	6600.000000	49.000000	54.000000	45400.000000

```
dsMean = ds[["enginesize", "curbweight", "horsepower"]].mean(axis=1)
plt.scatter(dsMean, ds["price"])
plt.show()
```

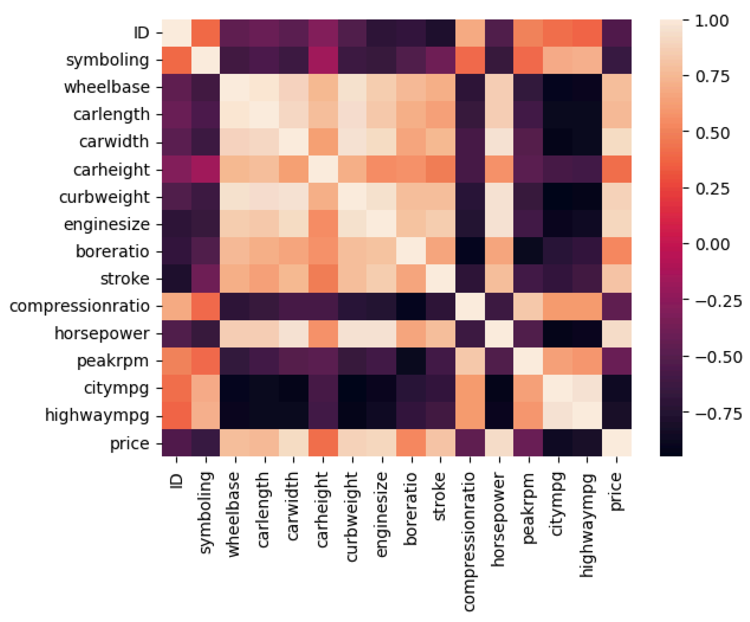



```
gsMean = gs[["enginesize", "curbweight", "horsepower"]].mean(axis=1)
plt.scatter(gsMean, gs["price"])
plt.show()
```



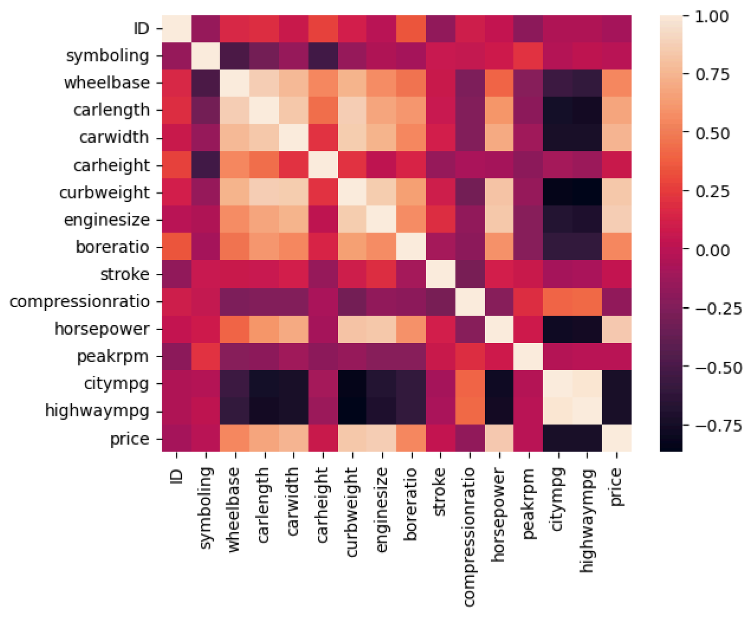

```
sns.heatmap(ds.corr())
```

```
<ipython-input-39-e18bdda33173>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(ds.corr())
<Axes: >
```




```
sns.heatmap(gs.corr())
```

```
<ipython-input-40-0b4f747c0ff5>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(gs.corr())
<Axes: >
```

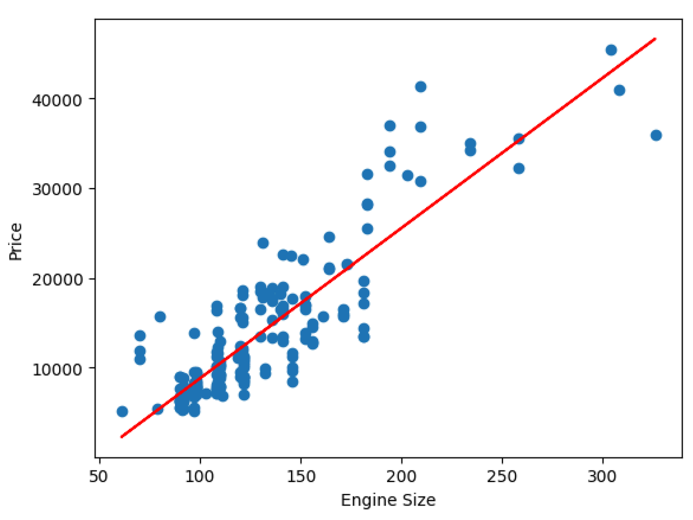


Linear Regression

Singular Linear Regression

```
#category
ds = data[(data.fueltypes    == 'diesel')]
gs = data[(data.fueltypes    == 'gas')]
Dtest = data
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.linear_model import LinearRegression

# Singular Linear Regression
x = Dtest['enginesize']
y = Dtest['price']
plt.scatter(x, y)
plt.xlabel('Engine Size')
plt.ylabel('Price')

# Convert the data into arrays
x = np.array(x).reshape(-1, 1)
y = np.array(y)

# Fit linear regression model
model = LinearRegression()
model.fit(x, y)

# Calculate the predictions
y_pred = model.predict(x)

# Calculate the coefficients
a1 = model.coef_[0]
a0 = model.intercept_

# Add the regression line
plt.plot(x, y_pred, color='red')
plt.show()

# R-squared
r_squared = r2_score(y, y_pred)

# Mean Absolute Error
mae = mean_absolute_error(y, y_pred)

# Mean Squared Error
mse = mean_squared_error(y, y_pred)

# Root Mean Squared Error
rmse = np.sqrt(mse)

# Print Result
print("R-squared:", r_squared)
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
```




```
R-squared: 0.7641291357806176
Mean Absolute Error: 2815.022353836411
Mean Squared Error: 14980261.40555132
Root Mean Squared Error: 3870.4342657576967
```

Evaluation

R-squared:

- Based on the result of the R-squared the value is approximately 0.7641 which indicates that around 76.41% of the "Price" variable is explained by the independent variable which is "Engine Size". Hence, the model explained how a significant portion of the Price is based on engine size.

Mean Absolute Error(MAE):

- The result of MAE is approximately 2815.02 meaning that the models's prediction is off by $2815.02.

Mean Squared Error(MSE):

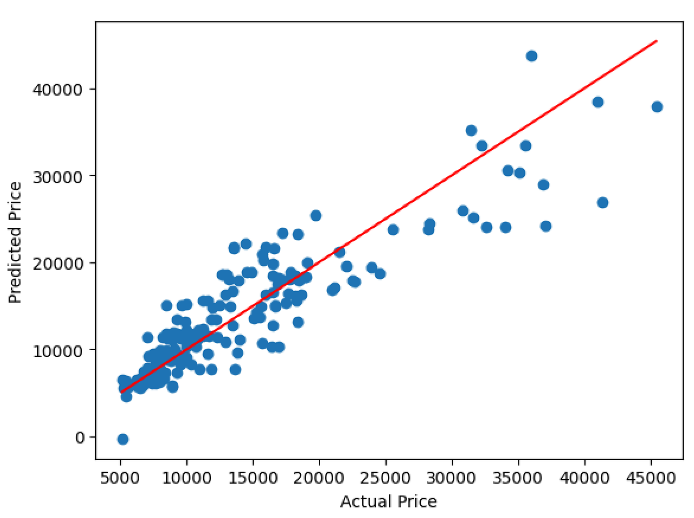
- The result of the MSE is approximately 14,980, 261.41 meaning the average squared difference between the predicted value and the actual value has large errors.

Root Mean Squared Error(RMSE):

- RMSE has an approximate 3,870.43 that represents the models prediction error in the same field as the target variable. which means that the model's prediction is from the actual price is $3,870.43

Multiple Linear Regression

```
# Multiple Linear Regression
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

x = Dtest[['enginesize', 'curbweight', 'horsepower', 'carlength', 'carwidth', 'citympg', 'highwaympg']]
y = Dtest['price']

# Convert the data into arrays
x = np.array(x)
y = np.array(y)

# Multiple Linear Regression model
model = LinearRegression()
model.fit(x, y)

# Prediction model
y_pred = model.predict(x)

# Plot the actual data
plt.scatter(y, y_pred)
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red')
plt.show()

# Mean Absolute Error
mae = mean_absolute_error(y, y_pred)

# Mean Squared Error
mse = mean_squared_error(y, y_pred)

# Root Mean Squared Error
rmse = np.sqrt(mse)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
```




```
Mean Absolute Error: 2384.153342535963
Mean Squared Error: 11402892.00079167
Root Mean Squared Error: 3376.81684442489
```

Evaluation

Mean Absolute Error (MAE):

- The result of MAE is at approximate 2384.15 which indicates that the models predictions are off by $2384.15 from the actual price.

Mean Squared Error (MSE):

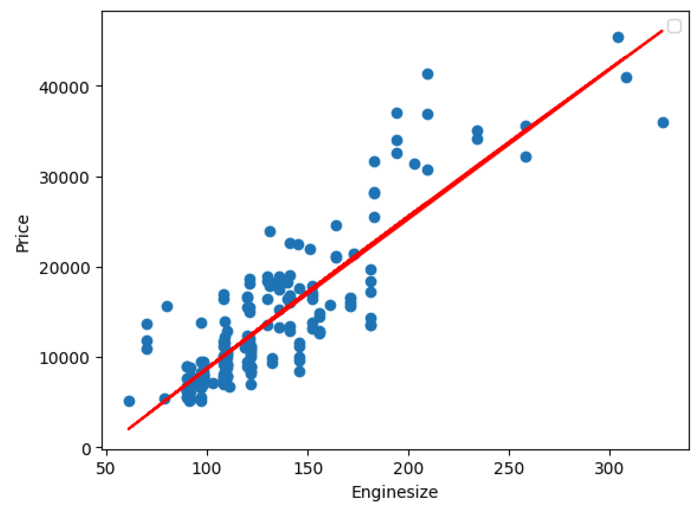
- The MSE has an approximate of 11,402,892 which indicates that the average diefference between the predicted price and the actual price is higher and has a higher percentage of giving large errors.

Root Mean Squared Error(RMSE):

- RMSE has an approximate 3376.82. This can be interpretted that the model's prediction error has an error of $3376.82.

```
# Polynomial Linear Regression

import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Polynomial Linear Regression
x = Dtest['enginesize']
y = Dtest['price']

x = np.array(x).reshape(-1, 1)

# Degree of the polynomial
degree = 2

# Prediction
poly_features = PolynomialFeatures(degree=degree)
x_poly = poly_features.fit_transform(x)

# Model
model = LinearRegression()
model.fit(x_poly, y)

# Prediction model
y_pred = model.predict(x_poly)

# Plot the actual data
plt.scatter(x, y)
plt.xlabel('Enginesize')
plt.ylabel('Price')

# Plot the fitted polynomial curve
plt.plot(x, y_pred, color='red'.format(degree))
plt.legend()
plt.show()

# Calculate Mean Absolute Error
mae = mean_absolute_error(y, y_pred)

# Calculate Mean Squared Error
mse = mean_squared_error(y, y_pred)

# Calculate Root Mean Squared Error
rmse = np.sqrt(mse)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
```




```
Mean Absolute Error: 2809.772875118241
Mean Squared Error: 14973637.600078523
Root Mean Squared Error: 3869.578478345997
```

Evaluation

Mean Absolute Error (MAE):

- The result of MAE is at approximate 2809.77 which means that the model's prediction are off by $2809.77.

Mean Squared Error (MSE):

- The MSE has an approximate of 14,973,637.60 meaning that the model's prediction is more prone to large erros rather than smaller ones.

Root Mean Squared Error(RMSE):

- RMSE has an approximate 3869.58. This can be interpretted that the model's prediction error has an error of $3869.58 from the actual price.

Logistic Regression

```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler

#Import data
OrigData = pd.read_csv('scrap price.csv')
LLR = OrigData


#Replace all columns that has a string into integer
LLR['fueltypes'] = LLR['fueltypes'].astype('category')
LLR['fueltypes'] = LLR['fueltypes'].cat.codes

LLR['name'] = LLR['name'].astype('category')
LLR['name'] = LLR['name'].cat.codes

LLR['aspiration'] = LLR['aspiration'].astype('category')
LLR['aspiration'] = LLR['aspiration'].cat.codes

LLR['doornumbers'] = LLR['doornumbers'].astype('category')
LLR['doornumbers'] = LLR['doornumbers'].cat.codes

LLR['carbody'] = LLR['carbody'].astype('category')
LLR['carbody'] = LLR['carbody'].cat.codes

LLR['drivewheels'] = LLR['drivewheels'].astype('category')
LLR['drivewheels'] = LLR['drivewheels'].cat.codes

LLR['enginelocation'] = LLR['enginelocation'].astype('category')
LLR['enginelocation'] = LLR['enginelocation'].cat.codes

LLR['enginetype'] = LLR['enginetype'].astype('category')
LLR['enginetype'] = LLR['enginetype'].cat.codes

LLR['cylindernumber'] = LLR['cylindernumber'].astype('category')
LLR['cylindernumber'] = LLR['cylindernumber'].cat.codes

LLR['fuelsystem'] = LLR['fuelsystem'].astype('category')
LLR['fuelsystem'] = LLR['fuelsystem'].cat.codes


#Logistic Regression
X = LLR.drop(columns = 'price')
y = LLR['fueltypes']


#Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 52)


#Initialize Standard Scale
Scale = StandardScaler()


#Model
X_train_scaled = Scale.fit_transform(X_train)
X_test_scaled = Scale.fit_transform(X_test)
LogReg = LogisticRegression(random_state = 0).fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
#Data Predict
Pred = LogReg.predict(X_train_scaled)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```

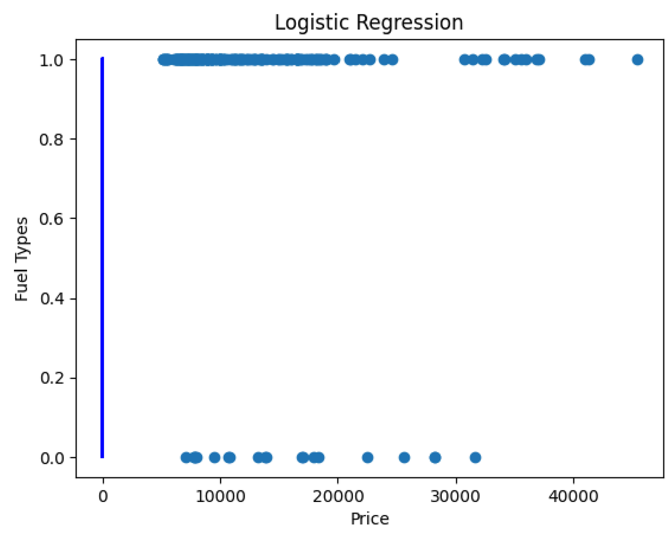
```
#Score of X training and X test
LogReg.score(X_train_scaled, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
0.7062937062937062
```

```
LogReg.score(X_test_scaled, y_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
0.6935483870967742
```

```
#Plot
plt.scatter(LLR['price'], LLR['fueltypes'])
plt.xlabel('Price')
plt.ylabel('Fuel Types')
plt.title('Logistic Regression')
plt.plot(X_train_scaled, Pred, color = 'Blue')
plt.show()
```



```
#Evaluation Model
MAE = mean_absolute_error(y_train, Pred)
MSE = mean_squared_error(y_train, Pred)
RMSE = np.sqrt(MSE)

print("Mean Squared Error:", MAE * 100, "%")
print("Median Squared Error:", MSE * 100, "%")
print("Mode Squared Error:", RMSE * 100, "%")
```

```
    Mean Squared Error: 29.37062937062937 %
    Median Squared Error: 29.37062937062937 %
    Mode Squared Error: 54.19467627971346 %
```

Evaluation

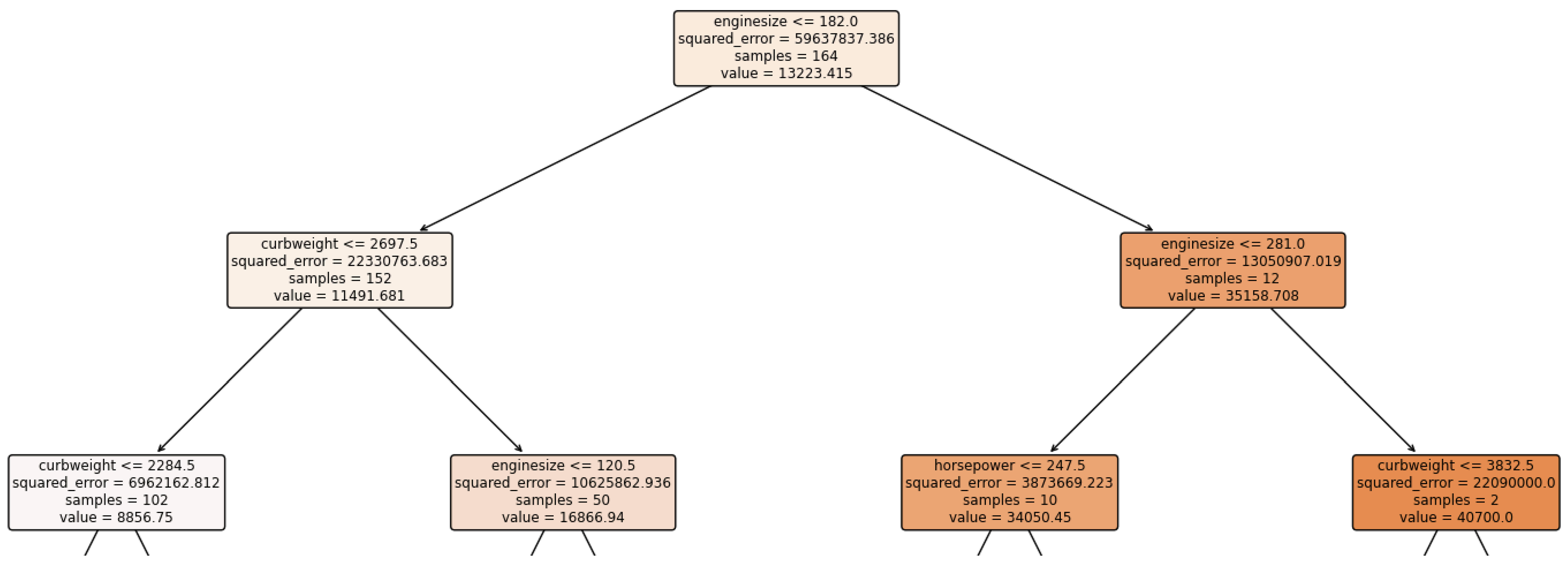
- Mean Absolute Error (MAE) - The Absolute Error value of the prediction in price for fuel is 29.37%.
- Median Squared Error (MSE) - The Squared Error value of the prediction in price with fuel, it has 29.37% and it was a low chance to occur.
- Mode Squared Error (RMSE) - 54.19% are the Squared Error, this prediction is in half percent therfore, there is a chance that the price will be identical either gas or diesel fuel.

Decision Tree

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

# Column Variable
X = data[['curbweight', 'enginesize', 'horsepower']]
y = data['price']

# Training set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Depth of decision tree
model = DecisionTreeRegressor(max_depth=3)

# Model for data
model.fit(X_train, y_train)

# Prediction of data
y_pred = model.predict(X_test)

# Evaluating model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
    Mean Squared Error: 11168485.300630387
```

```
# Decision Tree image
from sklearn.tree import plot_tree
plt.figure(figsize=(20,10))
plot_tree(model, feature_names=X.columns, filled=True, rounded=True)
plt.show()
```



Evaluation

As shown in the decision tree, the price of the vehicle is very dependent on the size of the engine. The dicision tree is split into 14 parts and it starts if the price is at lesser or equal than 182. The model shows that the engine size does have great factor/contribution in increasing the price of the said car.

Random Forest

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

X = data[['wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight', 'enginesize', 'boreratio', 'stroke', 'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg']]
y = data['price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=30)

# Model
model = RandomForestRegressor(n_estimators=100, random_state=30)
model.fit(X_train, y_train)

# Data prediction
ypred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, ypred)
print("Mean Squared Error:", mse)
```

```
    Mean Squared Error: 4508973.818539319
```

```
# Random Tree image
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 10))
plot_tree(model.estimators_[0], max_depth=2, feature_names=X.columns, filled=True)
plt.show()
```