Dhruvi Hiteshkumar Suthar
21162101005
CBA, Class: 7A, Batch: 71
Machine Learning

Practical - 4 - Implement linear regression for given dataset and find model which has highest r2 score and minimum MSE

Instructions:
Understand the problem statement properly
Clean dataset assigned to you
List of important attributes with proper justification
Read sample linear regression code

Answer following questions in a pdf file:

1. **List down all the important attributes in the dataset**
2. **Write down the models you have compared.**
3. **Write down the model which has highest r2 score and minimum MSE**

Linear Regression is statistical method used to model the relationship between a dependent variable(target) and one or more independent variables(features or predictors). The goal is to find the best fitting straight line that predicts the dependent variable based on the independent variables.

For example: W = B0 + B1 X H
here B0 is *intercept* (the value of W when H is 0), B1 is *slope* (how much W changes for a unit change in H)

R2 score measures how well the regression model fits the data. It ranges from 0 to 1:
0 : the model does not explain any of the variability in the target variable.
1 : the model perfectly explains all the variability in the target value.

Mean Squared Error (MSE) measures the average squared difference between the actual values and the predicted values. The lower the MSE, the better the model.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```
df=pd.read_csv("autos_csv.csv")
df.head(10)
```

Show hidden output

```
import numpy as np
```

We will first replace all the NaN values with True

```
df.replace('?',np.nan,inplace=True)
```

Now as for the columns with float values we can find the mean and replace them with True values.

```
mean_normalized_losses=df['normalized-losses'].astype(float).mean()
df['normalized-losses'].fillna(mean_normalized_losses,inplace=True)
df['normalized-losses']=df['normalized-losses'].astype(float)
```

```
mean_bore=df['bore'].astype(float).mean()
df['bore'].fillna(mean_bore,inplace=True)
df['bore']=df['bore'].astype(float)
```

```
mean_stroke=df['stroke'].astype(float).mean()
df['stroke'].fillna(mean_stroke,inplace=True)
df['stroke']=df['stroke'].astype(float)
```

```
mean_price=df['price'].astype(float).mean()
df['price'].fillna(mean_price,inplace=True)
df['price']=df['price'].astype(float)
```

Any other True values left like in columns with string values that row is deleted.

```
df.dropna(inplace=True)
df.head(10)
```

Show hidden output

Now we will find the important attributes using ANOVA and Correlation.

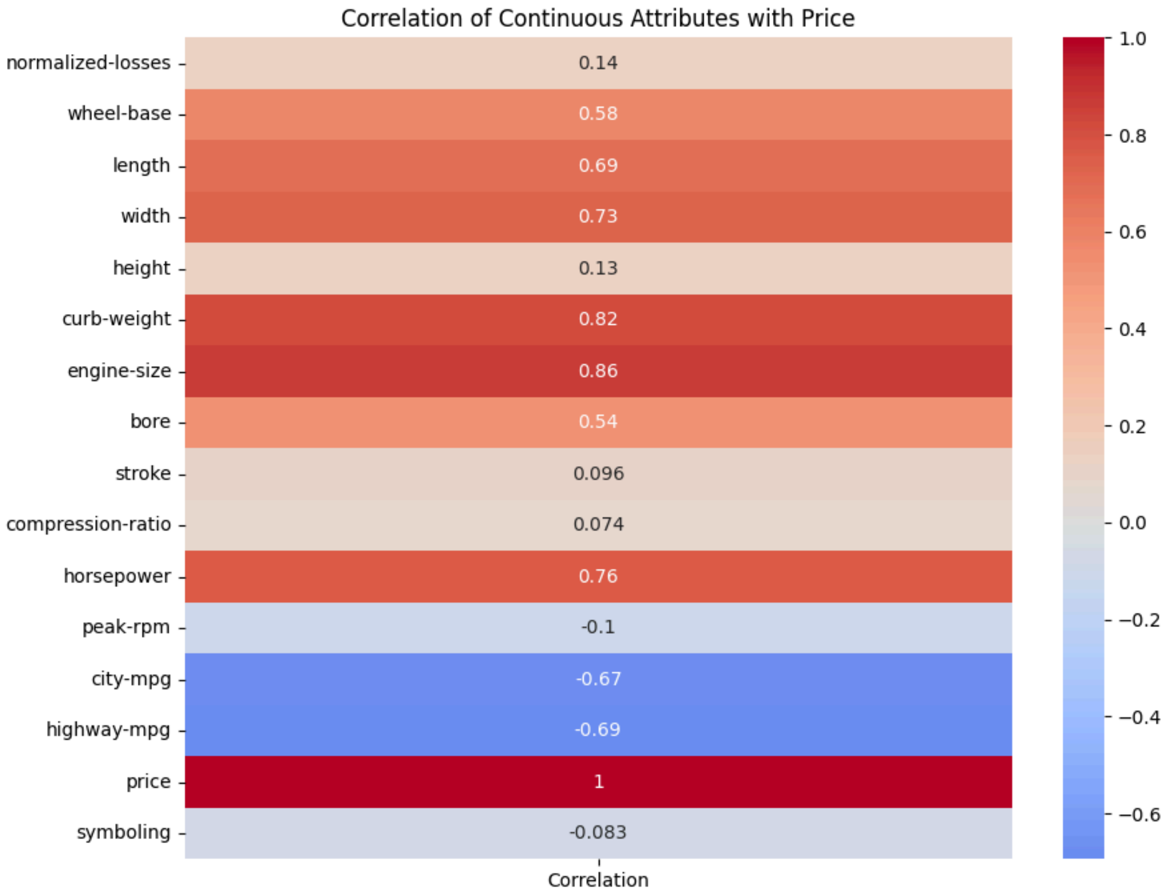
```
import seaborn as sns
import matplotlib.pyplot as plt
```

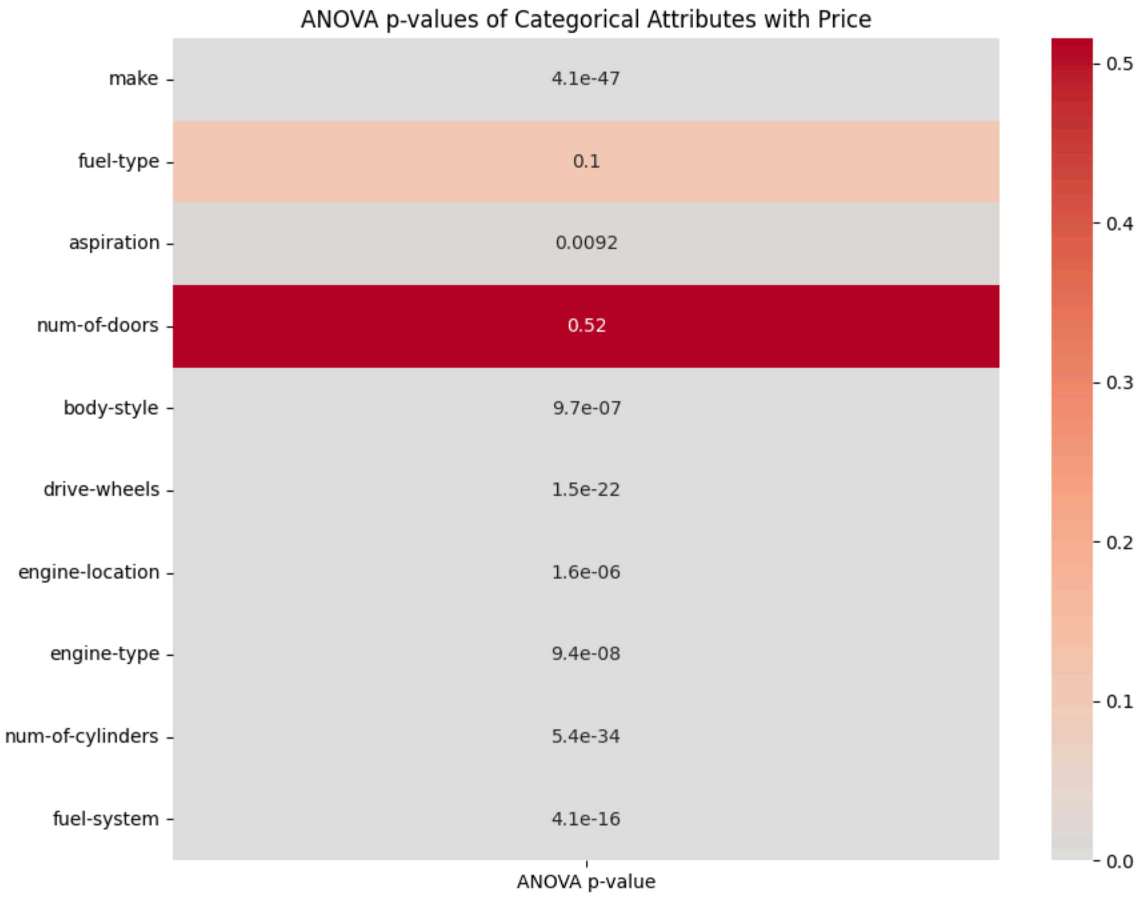
```
continuous=[]
categorical=[]
for i in df.columns:
  if df[i].dtype=="object":
    categorical.append(i)
  else:
    continuous.append(i)
print("Continuous Attributes : ", continuous)
print("Categorical Attributes : ", categorical)
```

```
Continuous Attributes :  ['normalized-losses', 'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-size', 'bore', 'stroke', 'compres
Categorical Attributes :  ['make', 'fuel-type', 'aspiration', 'num-of-doors', 'body-style', 'drive-wheels', 'engine-location', 'engine-type', 'num
```

```
from scipy.stats import f_oneway
```

```
important_continuous = []
important_categorical = []

correlation_dict = {}
for i, attribute in enumerate(continuous):
    correlation = df[attribute].corr(df['price'])
    correlation_dict[attribute] = correlation
    if abs(correlation) > 0.5 and attribute!='price':
        important_continuous.append(attribute)

correlation_df = pd.DataFrame.from_dict(correlation_dict, orient='index', columns=['Correlation'])
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_df, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation of Continuous Attributes with Price')
plt.show()

anova_dict = {}
for i, attribute in enumerate(categorical):
    cat_grp = []
    for cat in df[attribute].unique():
        cat_grp.append(df[df[attribute] == cat]["price"])

    f_statistic, p_value = f_oneway(*cat_grp)
    anova_dict[attribute] = p_value
    if p_value < 0.05:
        important_categorical.append(attribute)

print("")
anova_df = pd.DataFrame.from_dict(anova_dict, orient='index', columns=['ANOVA p-value'])
plt.figure(figsize=(10, 8))
sns.heatmap(anova_df, annot=True, cmap='coolwarm', center=0)
plt.title('ANOVA p-values of Categorical Attributes with Price')
plt.show()

print("Important Continuous Attributes:", important_continuous)
print("Important Categorical Attributes:", important_categorical)
```






```
Important Continuous Attributes: ['wheel-base', 'length', 'width', 'curb-weight', 'engine-size', 'bore', 'horsepower', 'city-mpg', 'highway-mpg']
```

Now as we have filtered out important attributes, we will only consider them for training and testing, this can help in reducing overfiiting, will make data more interpretable as there will be less and most relevant features.

Now we will first filter the data.

```
df_filtered = df[important_continuous + important_categorical + ['price']]
```

Now we will encode the categorical values.

```
df_encoded = pd.get_dummies(df_filtered, columns=important_categorical, drop_first=True)
```

Simple Linear Regression

Now we will split the data in training and testing sets.

```
x = df_encoded[['engine-size']]
y = df_encoded['price']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
print("x train shape:",x_train.shape)
print("y train shape:",y_train.shape)
print("x test shape:",x_test.shape)
print("y test shape:",y_test.shape)
```

```
x train shape: (140, 1)
y train shape: (140,)
x test shape: (61, 1)
y test shape: (61,)
```

First we will train the data.

```
model=LinearRegression()
model.fit(x_train,y_train)
print("Intercept:", model.intercept_)
print("Slope/Coefficient:", model.coef_)
```

```
Intercept: -5846.593293886866
Slope/Coefficient: [148.27657052]
```

Now we will predict the prices.

```
y_pred=model.predict(x_test)
pd.DataFrame({"Actual": y_test, "Predicted": y_pred})
```

	Actual	Predicted
97	7999.0	8536.234047
15	30760.0	25143.209946
31	6855.0	7794.851194
162	9258.0	8684.510617
132	11850.0	12094.871740
...	...	...
144	9233.0	10167.276323
100	9549.0	11946.595169
177	11248.0	12243.148310
98	8249.0	8536.234047
174	10698.0	10463.829464

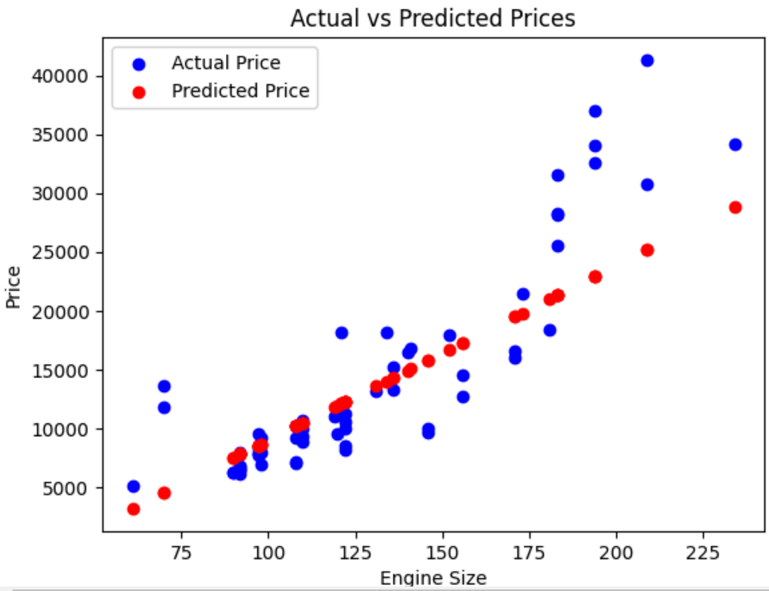
61 rows × 2 columns

Now to evaluate the model we will calculate R2 Score and MSE.

```
print("Simple Linear regression model:")
print('R2: ', r2_score(y_test, y_pred))
print('MSE: ', mean_squared_error(y_test, y_pred))
```

```
Simple Linear regression model:
R2:  0.7173106311223689
MSE:  22590620.56443966
```

```
plt.scatter(x_test, y_test, color='blue', label='Actual Price')
plt.scatter(x_test, y_pred, color='red', label='Predicted Price')
plt.xlabel('Engine Size')
plt.ylabel('Price')
plt.title('Actual vs Predicted Prices')
plt.legend()
plt.show()
```



Multiple Linear Regression

```
x = df_encoded[important_continuous]
y = df_encoded['price']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
print("x train shape:",x_train.shape)
print("y train shape:",y_train.shape)
print("x test shape:",x_test.shape)
print("y test shape:",y_test.shape)
```

```
x train shape: (140, 9)
y train shape: (140,)
x test shape: (61, 9)
y test shape: (61,)
```

```
model=LinearRegression()
model.fit(x_train,y_train)
print("Intercept:", model.intercept_)
print("Slope/Coefficient:", model.coef_)
```

```
Intercept: -36937.664080081224
Slope/Coefficient: [ 1.43459820e+02 -3.23165901e+01  6.51084708e+02  1.17695746e+00
  1.02505237e+02 -2.78632295e+03 -1.43538542e+01 -7.71865712e+01
 -1.60006079e+02]
```

```
y_pred=model.predict(x_test)
pd.DataFrame({"Actual": y_test, "Predicted": y_pred})
```

	Actual	Predicted
97	7999.0	6918.338032
15	30760.0	23131.316321
31	6855.0	6316.550177
162	9258.0	8416.678376
132	11850.0	12533.952551
...	...	...
144	9233.0	10761.899108
100	9549.0	10668.617694
177	11248.0	13026.046103
98	8249.0	7222.351560
174	10698.0	11814.486680

61 rows × 2 columns

```
print("Multiple Linear regression model:")
print('R2: ', r2_score(y_test, y_pred))
print('MSE: ', mean_squared_error(y_test, y_pred))
```

```
Multiple Linear regression model:
R2:  0.6559407241729635
MSE:  27494888.055903472
```

Polynomial Linear Regression (using pipeline)

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
```

```
pipe_inp = [('scale', StandardScaler()),
            ('polynomial', PolynomialFeatures(degree=2)),
            ('model', LinearRegression())]
pipe = Pipeline(pipe_inp)
pipe
```

Show hidden output

```
x = df_encoded[important_continuous]
y = df_encoded['price']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
print("x train shape:",x_train.shape)
print("y train shape:",y_train.shape)
print("x test shape:",x_test.shape)
print("y test shape:",y_test.shape)
```

Show hidden output

```
pipe.fit(x_train,y_train)
```

Show hidden output

```
y_pred = pipe.predict(x_test)
pd.DataFrame({"Actual": y_test, "Predicted": y_pred})
```

	Actual	Predicted
97	7999.0	8524.755787
15	30760.0	17896.363817
31	6855.0	5505.858806
162	9258.0	7234.016819
132	11850.0	15673.802439
...	...	...
144	9233.0	11174.850655
100	9549.0	9783.038071
177	11248.0	11734.996062
98	8249.0	7440.118060
174	10698.0	10640.588117

61 rows × 2 columns

```
print("Polynomial Linear regression model:")
print('R2: ', r2_score(y_test, y_pred))
print('MSE: ', mean_squared_error(y_test, y_pred))
```

```
Polynomial Linear regression model:
R2:  0.3327446268763994
MSE:  53322532.126575634
```

Simple Linear Regression has the highest R2 score and lowest MSE, indicating it explains the most variance and gas the least error among the