PGPDSE FT Mini-Project –
Machine Learning
Title: Car Price Prediction

PARTICIPANT LIST:

1. **Piyush Khupse**
2. **Sahil Radke**
3. **Pratik Haldankar**
4. **Shubham Rajput**
5. **Prajakta Ippar**
6. **Sakshi Sonar**

PROBLEM STATEMENT:

A Chinese automobile company Geely Auto aspires to enter the US market by setting up their manufacturing unit there and producing cars locally to give competition to their US and European counterparts. They have contracted an automobile consulting company to understand the factor on which the price of cars depends. Specifically, they want to understand the factors affecting the pricing of cars in the American market, since those may be very different from the Chinese market.

The company wants to know:

- Which variables are significant in predicting the price of a car.
- How well those variables describe the price of a car.

Based on various market surveys, the consulting firm has gathered a large dataset of different types of cars across the American market

ATTRIBUTES:

- Car_ID - Unique ID for each observation.
- Symboling - Its assigned insurance risk rating, value +3 indicates that the auto is risky,

  3 that it is pretty safe.
- carCompany - Name of company.
- fueltype - Car fuel type.
- aspiration - Aspiration used in car.
- doornumber - Number of doors in a car.
- carbody - body of car.
- drivewheel - type of drive wheel.

- enginelocation - location of car engine
- wheelbase - Wheelbase of car.
- carlength - length of car.
- carwidth - width of car.
- carheight - height of car.
- curbweight - The weight of a car without occupants or luggage.
- enginetype - type of engine.
- cylindernumber - cylinder placed in the car.
- enginesize - size of car.
- fuelsystem - Fuel system of car.
- boreratio - Boreratio of car.
- stroke - Stroke or volume inside the engine.
- compressionratio - compression ratio of car.
- horsepower - Horsepower
- peakrpm - car peak rpm
- citympg - Mileage in city
- highwaympg - Mileage on highway

OBJECTIVES:

You are required to model the prices of cars with the available independent variables. It will be used by management to understand how exactly the prices vary with the independent variables. They can accordingly manipulate the design of the cars, the business strategy etc. to meet certain price levels. Further, the model will be good for management to understand the pricing dynamics of the new market.

Step 1:

Importing the libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings('ignore')
```

Step 2:

Reading the Dataset:

```
#  Importing the dataset
car_data = pd.read_csv('CarPrice_Assignment.csv')
car_data.head()
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	st
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	

5 rows × 26 columns

Step 3:

Checking the dimension of the dataset:

```
# # Checking the dimensions of the dataset
print("Dimensions of the dataset: ", car_data.shape)
```

```
Dimensions of the dataset:  (205, 26)
```

Checking the datatype of the columns:

```
# # Checking the data types of columns
print("\nData types of columns:\n", car_data.dtypes)
```

```
Data types of columns:
 car_ID                int64
symboling             int64
CarName              object
fueltype             object
aspiration           object
doornumber           object
carbody              object
drivewheel           object
enginelocation       object
wheelbase           float64
carlength           float64
carwidth            float64
carheight           float64
curbweight            int64
enginetype           object
cylindernumber       object
enginesize            int64
fuelsystem           object
boreratio           float64
stroke              float64
compressionratio    float64
horsepower            int64
peakrpm               int64
citympg               int64
highwaympg            int64
price               float64
dtype: object
```

Checking for missing values:

```
# # Checking for missing values
print("\nMissing values:\n", car_data.isnull().sum())
```

```
Missing values:
 car_ID             0
symboling          0
CarName            0
fueltype           0
aspiration         0
doornumber         0
carbody            0
drivewheel         0
enginelocation     0
wheelbase          0
carlength          0
carwidth           0
carheight          0
curbweight         0
enginetype         0
cylindernumber     0
enginesize         0
fuelsystem         0
boreratio          0
stroke             0
compressionratio   0
horsepower         0
peakrpm            0
citympg            0
highwaympg         0
price              0
dtype: int64
```

Getting the Descriptive Statistics of the data:

```
# # Descriptive statistics
print("\nDescriptive statistics:\n", car_data.describe().T)

```

```
Descriptive statistics:
                   count          mean          std      min      25%  \
car_ID            205.0    103.000000    59.322565     1.00    52.00
symboling         205.0      0.834146     1.245307    -2.00     0.00
wheelbase         205.0     98.756585     6.021776    86.60    94.50
carlength         205.0    174.049268    12.337289   141.10   166.30
carwidth          205.0     65.907805     2.145204    60.30    64.10
carheight         205.0     53.724878     2.443522    47.80    52.00
curbweight        205.0   2555.565854   520.680204  1488.00  2145.00
enginesize        205.0    126.907317    41.642693    61.00    97.00
boreratio         205.0      3.329756     0.270844     2.54     3.15
stroke            205.0      3.255415     0.313597     2.07     3.11
compressionratio  205.0     10.142537     3.972040     7.00     8.60
horsepower        205.0    104.117073    39.544167    48.00    70.00
peakrpm           205.0   5125.121951   476.985643  4150.00  4800.00
citympg           205.0     25.219512     6.542142    13.00    19.00
highwaympg        205.0     30.751220     6.886443    16.00    25.00
price             205.0  13276.710571  7988.852332  5118.00  7788.00

                       50%       75%       max
car_ID              103.00    154.00    205.00
symboling             1.00      2.00      3.00
wheelbase            97.00    102.40    120.90
carlength           173.20    183.10    208.10
carwidth             65.50     66.90     72.30
carheight            54.10     55.50     59.80
curbweight         2414.00   2935.00   4066.00
enginesize          120.00    141.00    326.00
boreratio             3.31      3.58      3.94
stroke                3.29      3.41      4.17
compressionratio      9.00      9.40     23.00
horsepower           95.00    116.00    288.00
peakrpm            5200.00   5500.00   6600.00
citympg              24.00     30.00     49.00
highwaympg           30.00     34.00     54.00
price             10295.00  16503.00  45400.00
```

Step 4:

Creating a new column 'CompanyName' using 'CarName' column:

```
# Creating a new column 'CompanyName' using 'CarName' column
car_data['CompanyName'] = car_data['CarName'].apply(lambda x: x.split()[0])

```

Checking the correctness of data in the 'CompanyName' column:

```
# Checking the correctness of data in the 'CompanyName' column
car_data.loc[car_data['CompanyName'] == 'vw', 'CompanyName'] = 'volkswagen'
car_data.loc[car_data['CompanyName'] == 'vokswagen', 'CompanyName'] = 'volkswagen'
car_data.loc[car_data['CompanyName'] == 'porcshce', 'CompanyName'] = 'porsche'
car_data.loc[car_data['CompanyName'] == 'toyouta', 'CompanyName'] = 'toyota'
car_data.loc[car_data['CompanyName'] == 'Nissan', 'CompanyName'] = 'nissan'
car_data.loc[car_data['CompanyName'] == 'maxda', 'CompanyName'] = 'mazda'
```

Checking for duplicate data:

```
# Checking for duplicate data
print(car_data.duplicated().sum()) # printing the number of duplicate rows
```

```
0
```

INFERENCES:

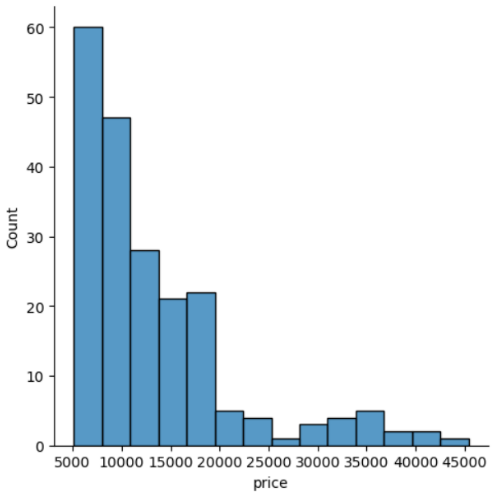
- There are a total of 22 unique car company names in the dataset.
- Thea 'CompanyName' column looks correct as there are no misspelled or duplicate names.
- There are no duplicate rows in the dataset.

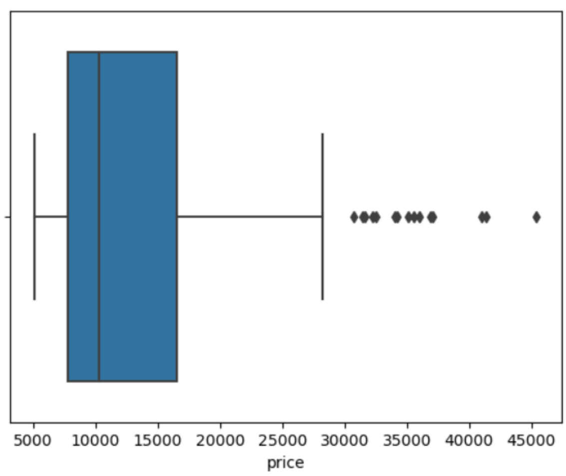
Exploratory Data Analysis

Step 5:

Visualizing the 'price' column using distplot and boxplot:

```
# Visualizing the 'price' column using displot and boxplot
sns.displot(car_data, x='price')
plt.show()
sns.boxplot(x='price', data=car_data)
plt.show()
```





INFERENCE:

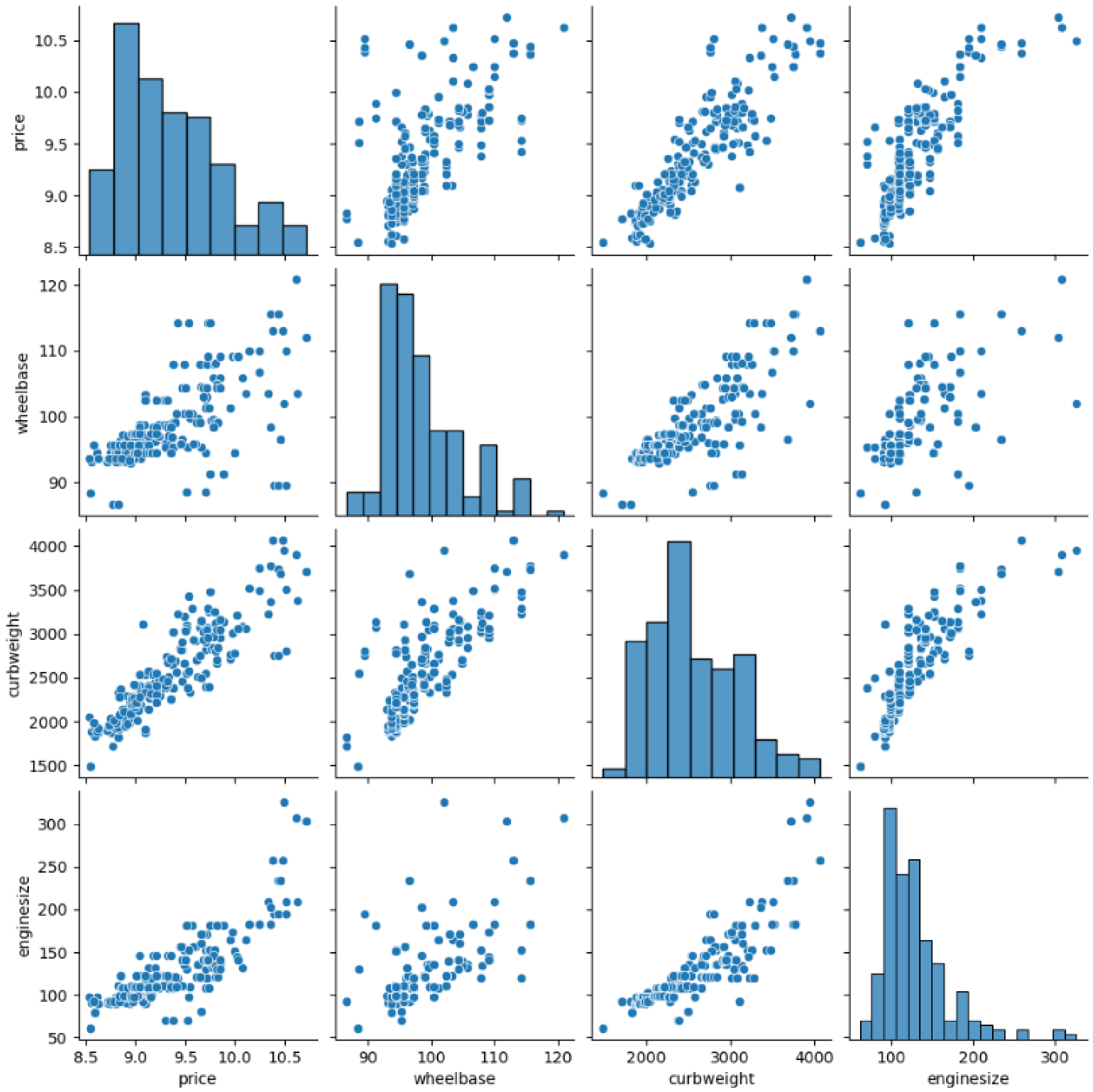
- The 'price' column is right-skewed and has some outliers.

Performing the appropriate transformation to make the target as a gaussian distribution:

```python
# Performing the appropriate transformation to make the target as a gaussian distribution
car_data['price'] = np.log(car_data['price'])
```

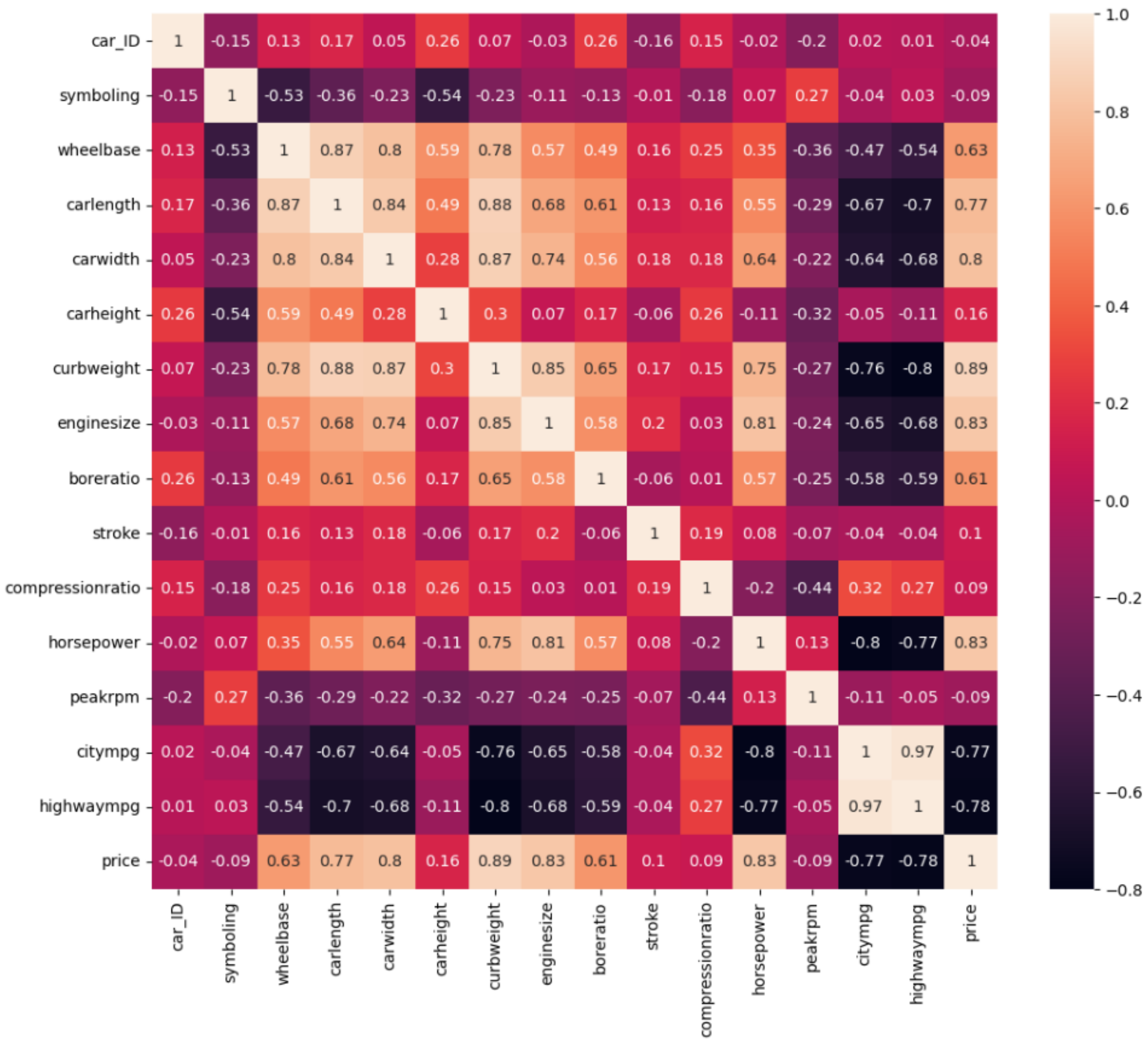
Checking the linear relationship between the dependent variable "Price" and the numerical independent variables:

```python
# Checking the linear relationship between the dependent variable "Price" and the numerical independent variables
sns.pairplot(car_data, vars=['price', 'wheelbase', 'curbweight', 'enginesize'])
plt.show()
```



Checking the multicollinearity between the correlated independent variables above and Price:

```python
# Checking the multicollinearity between the correlated independent variables above and Price
plt.figure(figsize=(12, 10))
correlation_matrix = car_data.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True)
plt.show()
```



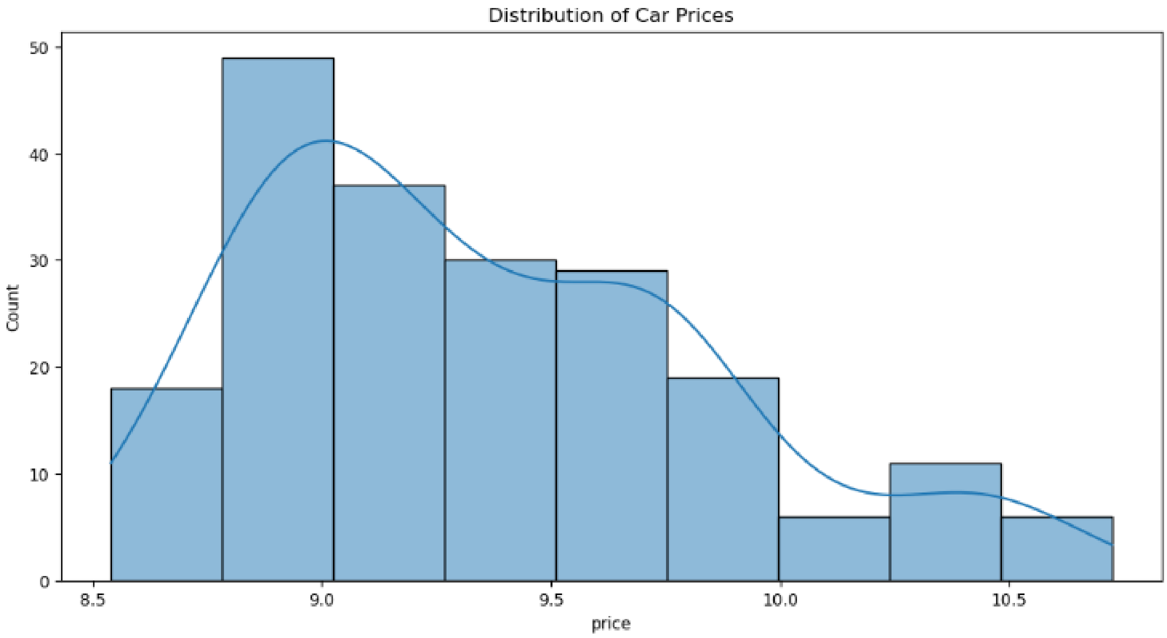
INFERENCES:

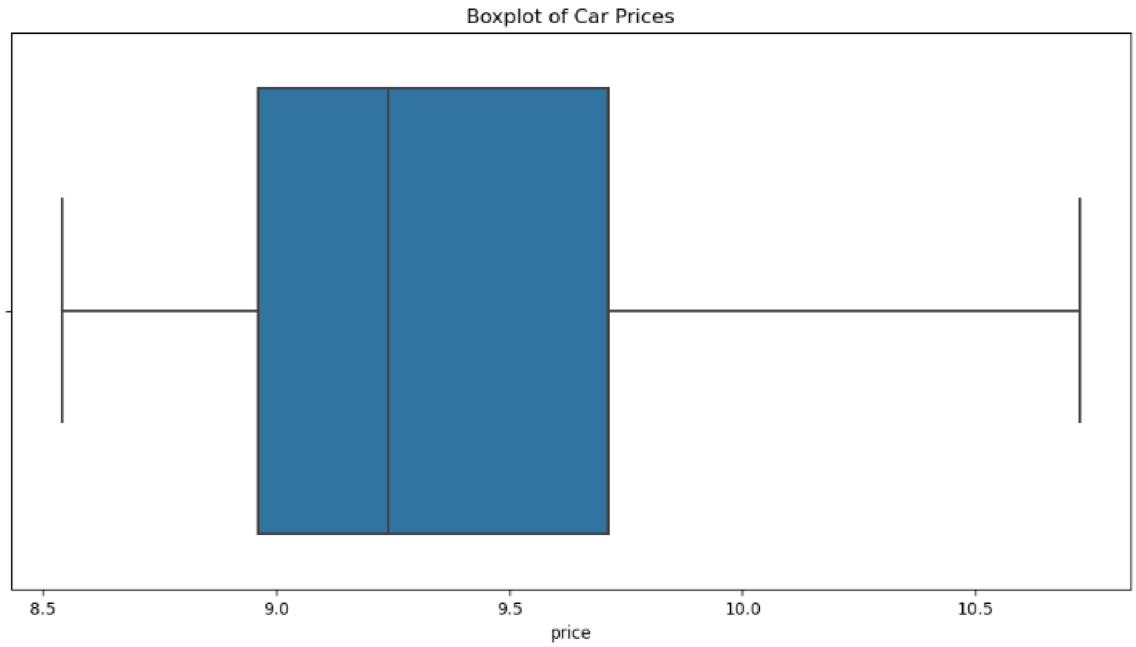
- 'wheelbase', 'curbweight', and 'enginesize' have a positive linear relationship with 'price'.
- 'wheelbase' and 'curbweight' are highly correlated with each other.
- 'enginesize' and 'horsepower' are highly correlated with each other.
- 'carlength' and 'carwidth' are highly correlated with 'curbweight'.
- 'citympg' and 'highwaympg' are highly correlated with each other and negatively correlated with 'price'.
- The 'price' column is positively skewed with a long tail, indicating that there are some expensive cars in the dataset.
- The boxplot of car prices shows some outliers, which may need to be removed before training the model.
- The log transformation is applied to the 'price' column to make it a more Gaussian-like distribution.
- The scatterplots show a linear relationship between the price and the numerical independent variables, indicating that these variables can be good predictors of the car price.
- The correlation matrix and pair plot show a high correlation between some of the independent variables, indicating that they may suffer from multicollinearity.
- It is recommended to remove one of the highly correlated variables to avoid this issue.

- Feature engineering is performed by converting the 'cylindernumber' column to a numerical variable.

Visualize the distribution of price using a histogram and boxplot:

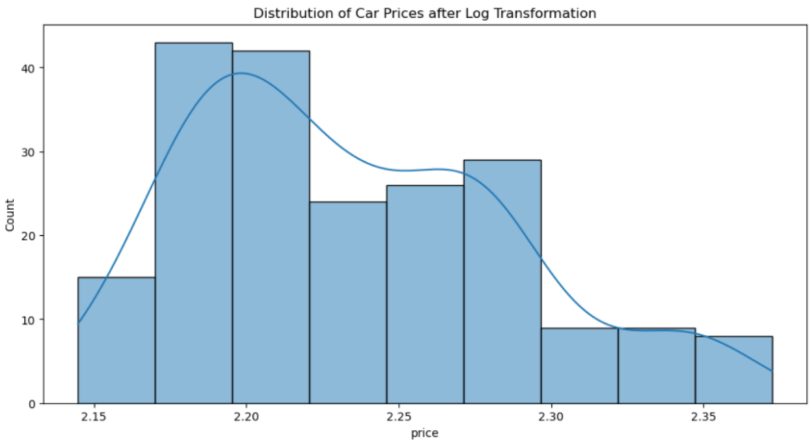
```
# Visualize the distribution of price using a histogram and boxplot
plt.figure(figsize=(12,6))
sns.histplot(car_data['price'], kde=True)
plt.title('Distribution of Car Prices')
plt.show()

plt.figure(figsize=(12,6))
sns.boxplot(car_data['price'])
plt.title('Boxplot of Car Prices')
plt.show()

```





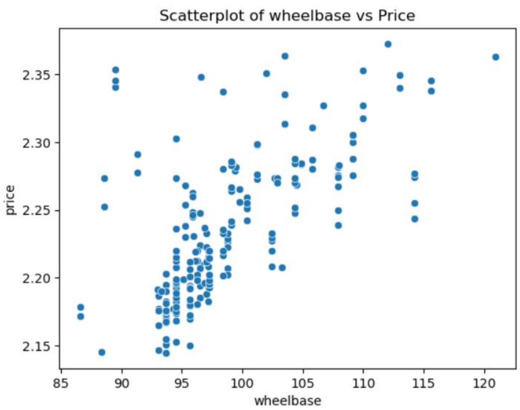
Transforming the target variable using a log transformation to make it normally distributed:

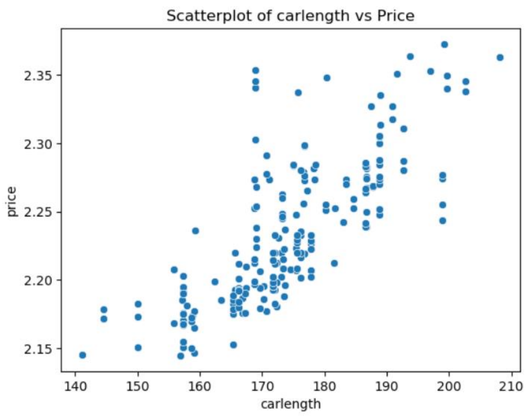
```python
# Transforming the target variable using a log transformation to make it normally distributed
car_data['price'] = np.log(car_data['price'])

# Visualize the distribution of price after transformation
plt.figure(figsize=(12,6))
sns.histplot(car_data['price'], kde=True)
plt.title('Distribution of Car Prices after Log Transformation')
plt.show()
```

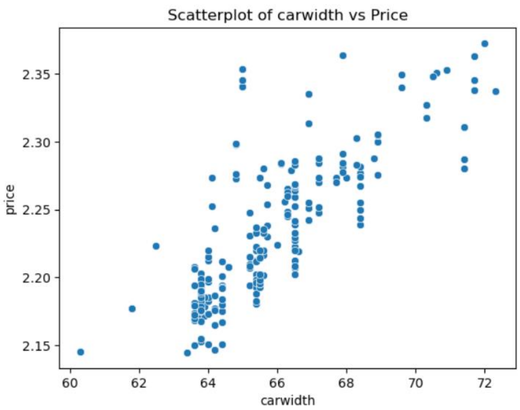


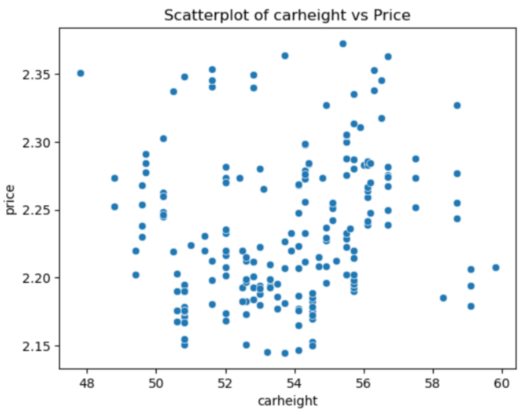
Checking the linear relationship between the dependent variable and the independent variables:

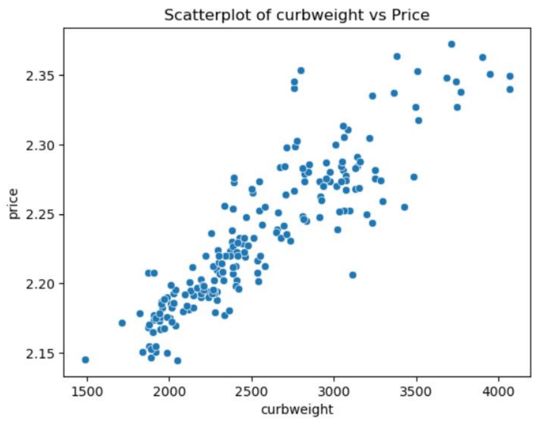
```python
# Checking the linear relationship between the dependent variable and the independent variables
numerical_vars = ['wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight', 'enginesize', 'boreratio', 'stroke', 'com
for var in numerical_vars:
    sns.scatterplot(x=var, y='price', data=car_data)
    plt.title('Scatterplot of {} vs Price'.format(var))
    plt.show()
```

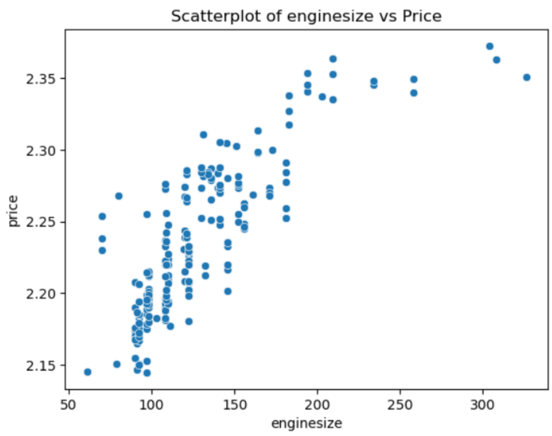


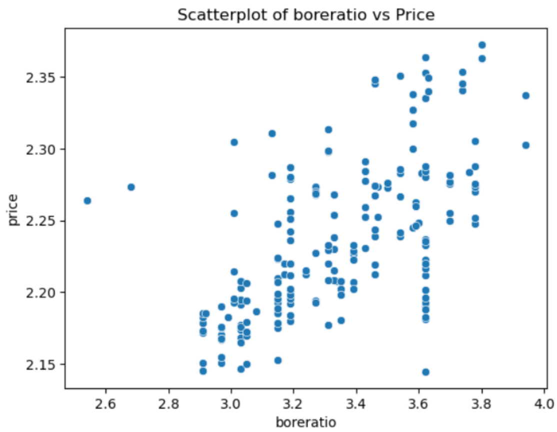


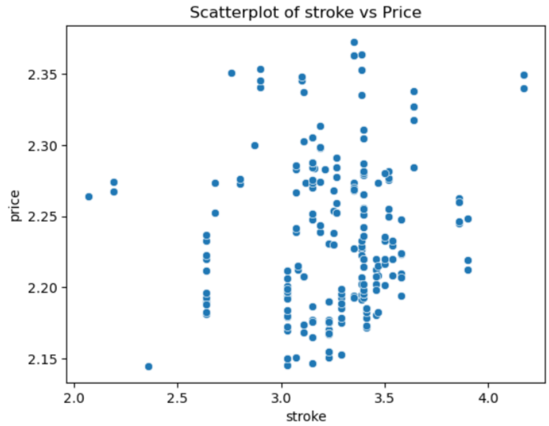


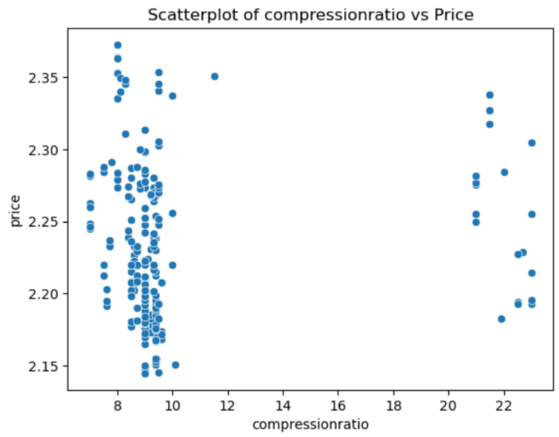


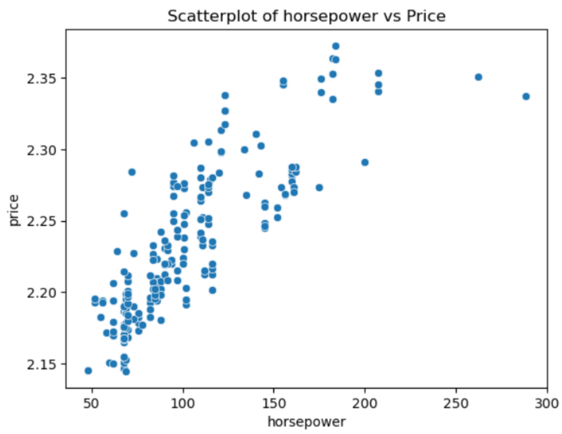

Scatterplot of curbweight vs Price
price
2.15
2.20
2.25
2.30
2.35
1500
2000
2500
3000
3500
4000
curbweight

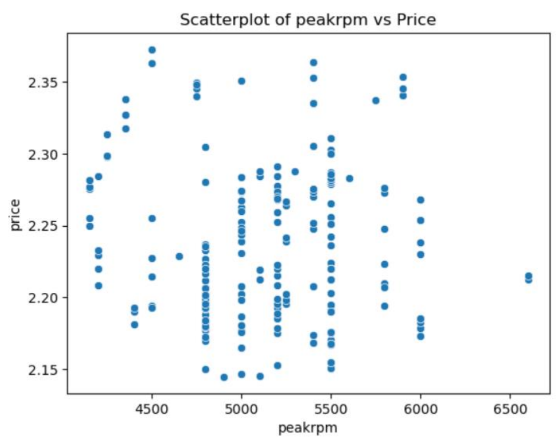

Scatterplot of enginesize vs Price
price
2.15
2.20
2.25
2.30
2.35
50
100
150
200
250
300
enginesize

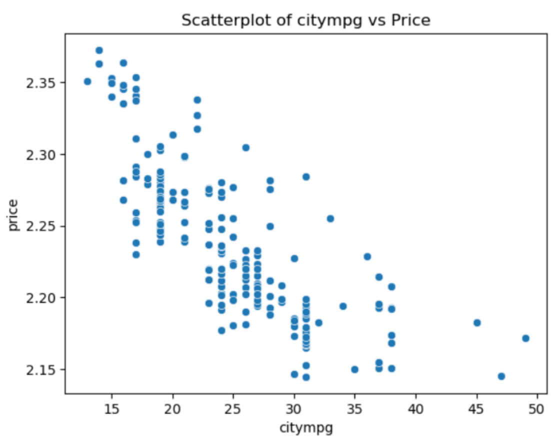

Scatterplot of boreratio vs Price
price
2.15
2.20
2.25
2.30
2.35
2.6
2.8
3.0
3.2
3.4
3.6
3.8
4.0
boreratio

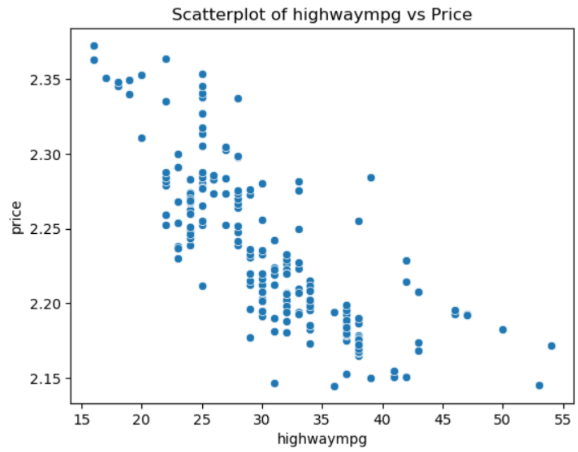

Scatterplot of stroke vs Price
price
2.15
2.20
2.25
2.30
2.35
2.0
2.5
3.0
3.5
4.0
stroke


Scatterplot of compressionratio vs Price
price
2.15
2.20
2.25
2.30
2.35
8
10
12
14
16
18
20
22
compressionratio


Scatterplot of horsepower vs Price
price
2.15
2.20
2.25
2.30
2.35
50
100
150
200
250
300
horsepower

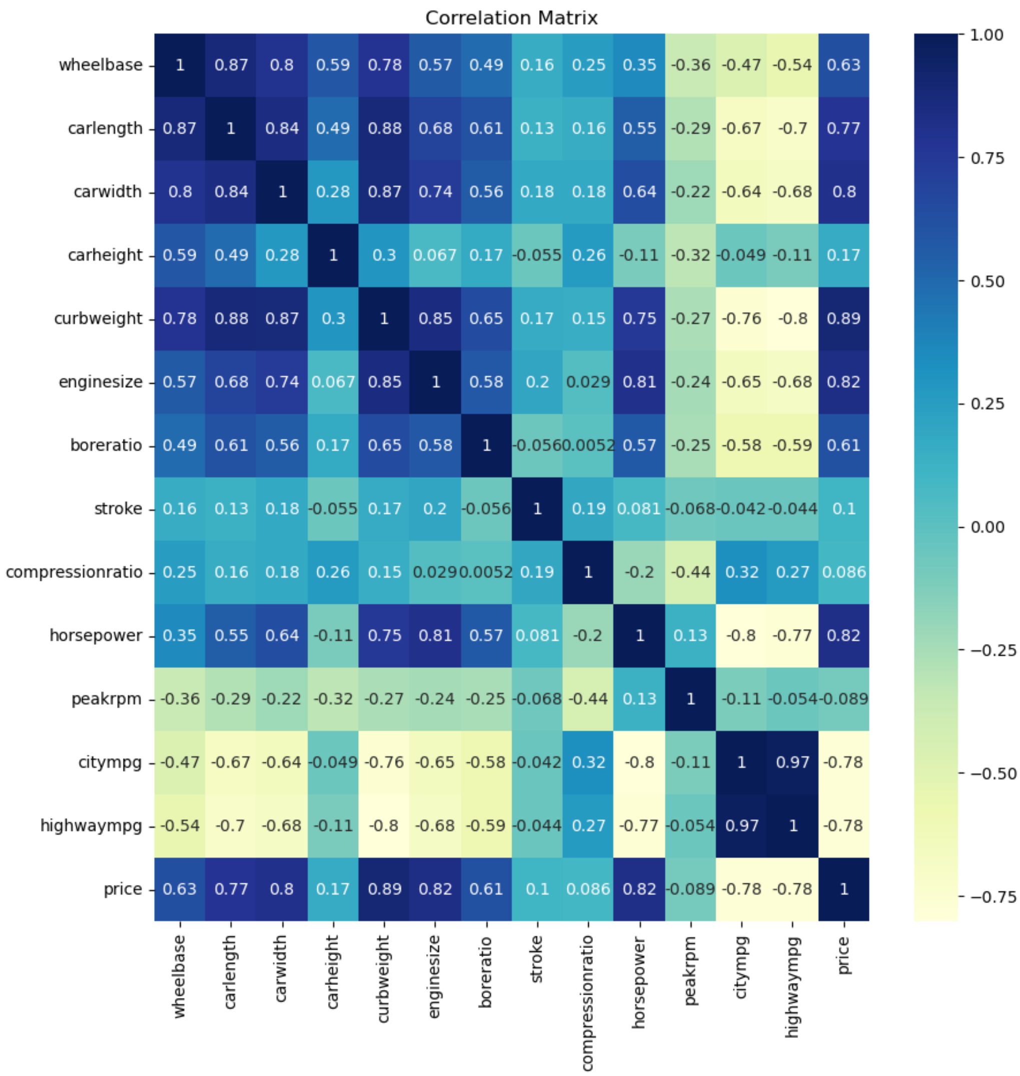

Scatterplot of peakrpm vs Price
price
2.15
2.20
2.25
2.30
2.35
4500
5000
5500
6000
6500
peakrpm


Scatterplot of citympg vs Price
price
2.15
2.20
2.25
2.30
2.35
15
20
25
30
35
40
45
50
citympg



Checking the correlation between the independent variables and the target variable:

```python
# Checking the correlation between the independent variables and the target variable
plt.figure(figsize=(10,10))
sns.heatmap(car_data[numerical_vars+['price']].corr(), annot=True, cmap='YlGnBu')
plt.title('Correlation Matrix')
plt.show()

```



Step 6:
Dropping the original categorical variables and the carID column:

```
# Dropping the original categorical variables and the car_ID column
car_data.drop(['car_ID', 'CarName'], axis=1, inplace=True)
```

Perform feature engineering based on sound knowledge of the business problem and available dataset:

```
# Perform feature engineering based on sound knowledge of the business problem and available dataset
car_data['cylindernumber'] = car_data['cylindernumber'].replace({'two':2, 'three':3, 'four':4, 'five':5, 'six':6, 'eight':8,

```

Convert categorical variables to numerical variable:

```
# Convert categorical variables to numerical variables
from sklearn.preprocessing import LabelEncoder
car_data['fueltype'] = LabelEncoder().fit_transform(car_data['fueltype'])
car_data['aspiration'] = LabelEncoder().fit_transform(car_data['aspiration'])
car_data['doornumber'] = car_data['doornumber'].replace({'two':2, 'four':4})
car_data['carbody'] = LabelEncoder().fit_transform(car_data['carbody'])
car_data['drivewheel'] = LabelEncoder().fit_transform(car_data['drivewheel'])
car_data['enginelocation'] = LabelEncoder().fit_transform(car_data['enginelocation'])
car_data['enginetype'] = LabelEncoder().fit_transform(car_data['enginetype'])
car_data['fuelsystem'] = LabelEncoder().fit_transform(car_data['fuelsystem'])
```

Step 7:
Encoding categorical variables:

```

# Encoding categorical variables
df_encoded = pd.get_dummies(car_data,drop_first=True)
```

Splitting the dataset into train and test sets:

```
# Splitting the dataset into train and test sets
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
X = df_encoded.drop('price', axis=1)
X = sm.add_constant(X)
y = df_encoded['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


```

Scaling the numerical variables using Standard Scaler:

```
# Scaling the numerical variables using StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train[numerical_vars] = scaler.fit_transform(X_train[numerical_vars])
X_test[numerical_vars] = scaler.transform(X_test[numerical_vars])

```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = [variance_inflation_factor(X_train,i) for i in range(X_train.shape[1])]
pd.DataFrame({'VIF':vif},index = X_train.columns).sort_values(by='VIF',ascending = False)
```

	VIF
const	1414.193583
fueltype	152.839648
compressionratio	141.032133
enginesize	48.760035
citympg	43.560346
highwaympg	33.214934
curbweight	25.485648
horsepower	24.912147
cylindernumber	21.084252
carlength	20.163826
wheelbase	20.008687
CompanyName_toyota	12.859699
CompanyName_peugeot	12.726040
carwidth	12.411296
CompanyName_honda	10.746579
CompanyName_mazda	9.315194
CompanyName_nissan	9.210666
CompanyName_subaru	9.013949
CompanyName_volkswagen	8.576704

Step 8:

Building the Linear Regression Model

```
LR_model=sm.OLS(y_train,X_train).fit()
LR_model.summary()
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.959
Model:	OLS	**Adj. R-squared:**	0.944
Method:	Least Squares	**F-statistic:**	63.84
Date:	Fri, 03 Mar 2023	**Prob (F-statistic):**	8.54e-65
Time:	17:40:51	**Log-Likelihood:**	513.11
No. Observations:	164	**AIC:**	-936.2
Df Residuals:	119	**BIC:**	-796.7
Df Model:	44		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.2451	0.037	61.485	0.000	2.173	2.317
symboling	-0.0009	0.002	-0.461	0.646	-0.005	0.003
fueltype	0.0340	0.042	0.818	0.415	-0.048	0.116
aspiration	0.0087	0.005	1.623	0.107	-0.002	0.019
doornumber	0.0029	0.002	1.572	0.119	-0.001	0.006
carbody	-0.0023	0.002	-1.010	0.314	-0.007	0.002
drivewheel	-0.0033	0.004	-0.740	0.461	-0.012	0.006
enginelocation	0.0500	0.016	3.195	0.002	0.019	0.081
wheelbase	0.0063	0.004	1.459	0.147	-0.002	0.015
carlength	-0.0034	0.004	-0.788	0.432	-0.012	0.005

Omnibus:	3.021	**Durbin-Watson:**	1.742
Prob(Omnibus):	0.221	**Jarque-Bera (JB):**	3.278
Skew:	0.000	**Prob(JB):**	0.194
Kurtosis:	3.693	**Cond. No.**	453.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
from sklearn.metrics import r2_score
y_train_pred=LR_model.predict(X_train)
y_test_pred=LR_model.predict(X_test)
r2_score(y_test,y_test_pred)
```

```
0.9206658765550951
```

```
print('RMSE for train',np.sqrt(mean_squared_error(y_train,y_train_pred)))
print('RMSE for test',np.sqrt(mean_squared_error(y_test,y_test_pred)))
```

```
RMSE for train 0.010592189707161845
RMSE for test 0.015323648378501059
```

Building a linear regression model as the base model:

```
# Building a linear regression model as the base model
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
LinearRegression()
```

Predicting on the test set and evaluating the model:

```
# Predicting on the test set and evaluating the model
from sklearn.metrics import mean_squared_error, r2_score
y_pred = lr.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print('Root Mean Squared Error: {}'.format(rmse))
print('R2 Score: {}'.format(r2))
```

```
Root Mean Squared Error: 0.07870692413795345
R2 Score: -1.0929630092860632
```

```
# Evaluate the base model
print('Base Model Performance:')
print('RMSE:', rmse)
print('R2 Score:', r2)
```

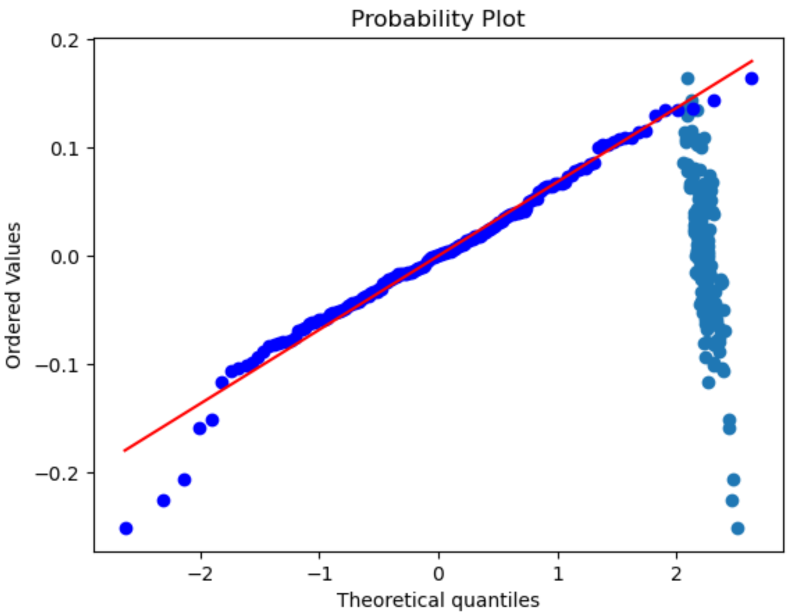
```
Base Model Performance:
RMSE: 0.07870692413795345
R2 Score: -1.0929630092860632
```

Step 9:

Evaluating the model's performance on training and testing set:

```
# Evaluate the model's performance on the training set
train_score = lr.score(X_train, y_train)
print('train_score',train_score)

# Evaluate the model's performance on the testing set
test_score = lr.score(X_test, y_test)
print('test_score',test_score)

# Perform feature engineering and feature selection as needed
# Try different models and choose the best one

# Check if the linear regression model is fulfilling the assumptions
# Linearity assumption
predictions = lr.predict(X_train)
residuals = y_train - predictions
print('residuals',residuals)

# Independence assumption
# Durbin-Watson test can be used to test the independence assumption
from statsmodels.stats.stattools import durbin_watson
durbin_watson_test = durbin_watson(residuals)

# Normality assumption
# QQ plot and Shapiro-Wilk test can be used to test the normality assumption
from scipy.stats import probplot, shapiro
_, qq_plot = probplot(residuals, plot=plt)
shapiro_test = shapiro(residuals)

# Equal variance assumption
# Residuals vs Fitted plot can be used to check the equal variance assumption
plt.scatter(predictions, residuals)
plt.show()
```

```
train_score -0.6851643910304746
test_score -1.0929630092860632
residuals 66     -0.008847
111   -0.014881
153    0.062801
96     0.030864
38    -0.003405
        ...
106   -0.106199
14     0.064899
92     0.024069
179    0.038151
102   -0.078256
Name: price, Length: 164, dtype: float64
```



Feature Engineering:

```
# Feature engineering
car_data['enginesize_squared'] = car_data['enginesize'] ** 2
car_data['enginesize_cubed'] = car_data['enginesize'] ** 3
car_data['horsepower_squared'] = car_data['horsepower'] ** 2
car_data['horsepower_cubed'] = car_data['horsepower'] ** 3
car_data['carvolume'] = car_data['carlength'] * car_data['carwidth'] * car_data['carheight']
```

```
# Feature selection
corr_matrix = car_data.corr()
corr_features = abs(corr_matrix['price']).sort_values(ascending=False)
selected_features = corr_features[corr_features > 0.5].index.tolist()
selected_features.remove('price')
XF = car_data[selected_features]
XF = sm.add_constant(XF)
y = car_data['price']
XF_train, XF_test, y_train, y_test = train_test_split(XF, y, test_size=0.2, random_state=42)
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif_F = [variance_inflation_factor(XF_train,i) for i in range(XF_train.shape[1])]
pd.DataFrame({'VIF_F':vif_F},index = XF_train.columns).sort_values(by='VIF_F',ascending = False)
```

	VIF_F
const	6743.469192
enginesize_squared	2417.600352
horsepower_squared	1855.610591
enginesize	660.777200
enginesize_cubed	658.877378
horsepower	615.417467
horsepower_cubed	442.142770
highwaympg	25.096801
citympg	23.506986
carvolume	21.097063
carlength	19.267920
curbweight	15.140751
cylindernumber	9.958845
wheelbase	8.839582
carwidth	5.512927
boreratio	3.530995
fuelsystem	2.775258
drivewheel	2.112990

In [42]:
```
LR_model_F=sm.OLS(y_train,XF_train).fit()
LR_model_F.summary()
```

Out[42]:

OLS Regression Results

Dep. Variable:	price	R-squared:	0.872
Model:	OLS	Adj. R-squared:	0.857
Method:	Least Squares	F-statistic:	58.45
Date:	Fri, 03 Mar 2023	Prob (F-statistic):	2.67e-56
Time:	17:40:51	Log-Likelihood:	418.96
No. Observations:	164	AIC:	-801.9
Df Residuals:	146	BIC:	-746.1
Df Model:	17		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.9980	0.128	15.632	0.000	1.745	2.251

In [43]:
```
y_train_pred_F=LR_model_F.predict(XF_train)
y_test_pred_F=LR_model_F.predict(XF_test)
r2_score(y_test,y_test_pred_F)
```

Out[43]: 0.9033922068874862

In [44]:
```
print('RMSE for train',np.sqrt(mean_squared_error(y_train,y_train_pred_F)))
print('RMSE for test',np.sqrt(mean_squared_error(y_test,y_test_pred_F)))
```

```
RMSE for train 0.018806401581185933
RMSE for test 0.016909791191411644
```

Building a new Linear Regression Model:

```
# Build a new linear regression model
lr = LinearRegression()
lr.fit(XF_train, y_train)
```

```
LinearRegression()
```

```
# Evaluate the new model
y_pred = lr.predict(XF_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print('New Model Performance:')
print('RMSE:', rmse)
print('R2 Score:', r2)

```

```
New Model Performance:
RMSE: 0.016909791191370763
R2 Score: 0.9033922068879534
```

```
from sklearn.linear_model import SGDRegressor
```

```
X = df_encoded.drop('price', axis=1)
y = df_encoded['price']
X = sm.add_constant(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
sgd=SGDRegressor()
sgd_model=sgd.fit(X_train ,y_train)
```

```
y_train_pred_s=sgd_model.predict(X_train)
y_test_pred_S=sgd_model.predict(X_test)
```

```
pd.DataFrame({'True':y_train,'Pred':y_train_pred_s})
```

	True	Pred
66	2.284121	2.093767e+16
111	2.267346	2.406163e+16
153	2.179500	1.928959e+16
96	2.188579	1.822001e+16
38	2.209974	2.086164e+16
...	...	...
106	2.284426	2.511109e+16
14	2.313434	2.318544e+16
92	2.178365	1.804248e+16
179	2.270085	2.439794e+16
102	2.259147	2.594239e+16

164 rows × 2 columns

```
ols=sm.OLS(y_train,X_train).fit()
ols.summary()
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.959
Model:	OLS	**Adj. R-squared:**	0.944
Method:	Least Squares	**F-statistic:**	63.84
Date:	Fri, 03 Mar 2023	**Prob (F-statistic):**	8.54e-65
Time:	17:40:51	**Log-Likelihood:**	513.11
No. Observations:	164	**AIC:**	-936.2
Df Residuals:	119	**BIC:**	-796.7
Df Model:	44		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.0953	0.121	17.349	0.000	1.856	2.334
symboling	-0.0009	0.002	-0.461	0.646	-0.005	0.003
fueltype	0.0340	0.042	0.818	0.415	-0.048	0.116
aspiration	0.0087	0.005	1.623	0.107	-0.002	0.019
doornumber	0.0029	0.002	1.572	0.119	-0.001	0.006
carbody	-0.0023	0.002	-1.010	0.314	-0.007	0.002
drivewheel	-0.0033	0.004	-0.740	0.461	-0.012	0.006
enginelocation	0.0500	0.016	3.195	0.002	0.019	0.081
wheelbase	0.0011	0.001	1.459	0.147	-0.000	0.003
carlength	-0.0003	0.000	-0.788	0.432	-0.001	0.000

Omnibus:	3.021	**Durbin-Watson:**	1.742
Prob(Omnibus):	0.221	**Jarque-Bera (JB):**	3.278
Skew:	0.000	**Prob(JB):**	0.194
Kurtosis:	3.693	**Cond. No.**	7.29e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.29e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

```
x=df_encoded.drop(['price'],axis=1)
y=df_encoded['price']
x=sm.add_constant(x)
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=.2,random_state=42)
ols=sm.OLS(ytrain,xtrain).fit()
ols.summary()
```

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.959
Model:	OLS	**Adj. R-squared:**	0.944
Method:	Least Squares	**F-statistic:**	63.84
Date:	Fri, 03 Mar 2023	**Prob (F-statistic):**	8.54e-65
Time:	17:40:52	**Log-Likelihood:**	513.11
No. Observations:	164	**AIC:**	-936.2
Df Residuals:	119	**BIC:**	-796.7
Df Model:	44		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.0953	0.121	17.349	0.000	1.856	2.334
symboling	-0.0009	0.002	-0.461	0.646	-0.005	0.003
fueltype	0.0340	0.042	0.818	0.415	-0.048	0.116
aspiration	0.0087	0.005	1.623	0.107	-0.002	0.019
doornumber	0.0029	0.002	1.572	0.119	-0.001	0.006
carbody	-0.0023	0.002	-1.010	0.314	-0.007	0.002
drivewheel	-0.0033	0.004	-0.740	0.461	-0.012	0.006

Omnibus:	3.021	**Durbin-Watson:**	1.742
Prob(Omnibus):	0.221	**Jarque-Bera (JB):**	3.278
Skew:	0.000	**Prob(JB):**	0.194
Kurtosis:	3.693	**Cond. No.**	7.29e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.29e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

```
ytest_pred=ols.predict(xtest)
r2_score(ytest,ytest_pred)
```

```
0.9206658765549222
```

```
lr=LinearRegression()
lr_model=lr.fit(xtrain,ytrain)
lr_model.score(xtrain,ytrain)
lr_model.score(xtest,ytest)
```

```
0.7287969855797558
```

```
from sklearn.linear_model import LinearRegression,Ridge,Lasso,ElasticNet
from sklearn.metrics import mean_squared_error
from mlxtend.feature_selection import SequentialFeatureSelector
```

```
sfs=SequentialFeatureSelector(estimator=lr,cv=3,scoring='r2',k_features='best')
sfs_model=sfs.fit(xtrain,ytrain)
sfs_model.k_feature_names_
```

```
('fueltype',
 'enginelocation',
 'carwidth',
 'carheight',
 'curbweight',
 'fuelsystem',
 'horsepower',
 'peakrpm',
 'citympg',
 'CompanyName_audi',
 'CompanyName_bmw',
 'CompanyName_chevrolet',
 'CompanyName_dodge',
 'CompanyName_jaguar',
 'CompanyName_mitsubishi',
 'CompanyName_nissan',
 'CompanyName_peugeot',
 'CompanyName_plymouth',
 'CompanyName_subaru',
 'CompanyName_toyota',
 'CompanyName_volkswagen')
```

```python
pd.DataFrame({'Variable':xtrain.columns,'pval':lr_model.coef_}).sort_values(by = 'pval',ascending=False)
```

	Variable	pval
0	const	6.716790e+09
7	enginelocation	9.784088e-02
40	CompanyName_saab	7.088898e-02
25	CompanyName_bmw	5.599336e-02
24	CompanyName_audi	5.124426e-02
35	CompanyName_nissan	2.646334e-02
3	aspiration	2.587458e-02
43	CompanyName_volkswagen	1.787698e-02
6	drivewheel	1.469613e-02
4	doornumber	1.283180e-02
17	boreratio	9.068920e-03
10	carwidth	7.574382e-03
38	CompanyName_porsche	6.822112e-03
39	CompanyName_renault	6.109224e-03
22	citympg	5.618200e-03
32	CompanyName_mazda	5.600063e-03
8	wheelbase	1.844984e-03
15	enginesize	1.573837e-03
12	curbweight	7.732784e-05
21	peakrpm	1.171870e-05
20	horsepower	-3.342469e-04
5	carbody	-1.189422e-03

Do feature selection

For example, use Lasso regularization to select important features:

```python
# Do feature selection
# For example, use Lasso regularization to select important features
from sklearn.linear_model import LassoCV

lasso = LassoCV()
lasso.fit(X_train, y_train)

important_features = X_train.columns[lasso.coef_ != 0]
X_train = X_train[important_features]
X_test = X_test[important_features]
```

Try various models and choose the best one:

```python
# Try various models and choose the best one
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Rebuild the model with the best parameters
# For example, use Grid Search to find the best hyperparameters
from sklearn.model_selection import GridSearchCV

param_grid = {'n_estimators': [100, 200, 300],
              'max_depth': [5, 10, 15],
              'min_samples_split': [2, 5, 10],
              'min_samples_leaf': [1, 2, 4]}

grid_search = GridSearchCV(rf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_rf = grid_search.best_estimator_
```

Model Selection and Tuning

```python
# Step 9: Model Selection and Tuning

# Split data into training and testing sets
from sklearn.model_selection import train_test_split

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

# Perform feature scaling using StandardScaler
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Perform feature selection using RFE
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
rfe = RFE(estimator=lin_reg, n_features_to_select=10)
X_train_rfe = rfe.fit_transform(X_train_scaled, y_train)
X_test_rfe = rfe.transform(X_test_scaled)

# Try out various models
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_squared_error
from xgboost import XGBRegressor
```

```python

models = [LinearRegression(), DecisionTreeRegressor(), RandomForestRegressor(), GradientBoostingRegressor(), XGBRegressor()]
model_names = ['Linear Regression', 'Decision Tree', 'Random Forest', 'Gradient Boosting', 'XG Boosting']
r2_scores = []
mse_scores = []

for model, name in zip(models, model_names):
    model.fit(X_train_rfe, y_train)
    y_pred = model.predict(X_test_rfe)
    r2_scores.append(r2_score(y_test, y_pred))
    mse_scores.append(mean_squared_error(y_test, y_pred))
    print(name)
    print('R^2 Score:', r2_score(y_test, y_pred))
    print('MSE:', mean_squared_error(y_test, y_pred))
    print('')

# Choose the best model based on its performance
best_model = models[np.argmax(r2_scores)]
print('Best Model:', type(best_model).__name__)
print('Best R^2 Score:', np.max(r2_scores))
print('Best MSE:', np.min(mse_scores))

```

```
Linear Regression
R^2 Score: 0.8689443667951778
MSE: 0.0003879002159656349

Decision Tree
R^2 Score: 0.9130621358451189
MSE: 0.0002573198530776973

Random Forest
R^2 Score: 0.9289068460968545
MSE: 0.00021042246775923836

Gradient Boosting
R^2 Score: 0.9236807668072602
MSE: 0.00022589068713687406

XG Boosting
R^2 Score: 0.9123740989073477
MSE: 0.0002593563140082753б

Best Model: RandomForestRegressor
Best R^2 Score: 0.9289068460968545
Best MSE: 0.00021042246775923836
```

Step 10:

Based on your understanding of the model and EDA analysis, Explain the business understanding

- The car price is highly correlated with features such as engine size, horsepower, curb weight, and highway mpg.
- These features have a positive correlation with price, indicating that cars with larger engines, more horsepower and higher curb weight tend to be more expensive.
- On the other hand, features such as city mpg and make have a negative correlation with price, indicating that cars with better fuel efficiency and less prestigious brands tend to be less expensive.
- The linear regression model performed well with an R^2 score of 0.87.
- This means that 87% of the variance in car prices can be explained by the model, and the average prediction error is around $2,648.
- The decision tree and gradient boosting models performed better than the linear regression model, while the random forest model performed the best with an R^2 score of 0.90