Car Price Prediction

Dataset source: https://archive.ics.uci.edu/ml/datasets/Automobile

```
link='https://drive.google.com/uc?id=1UO
import pandas as pd
df=pd.read_csv(link)
df.head()
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody
0	1	3	alfa-romero giulia	gas	std	two	convertible
1	2	3	alfa-romero stelvio	gas	std	two	convertible
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback
3	4	2	audi 100 ls	gas	std	four	sedan
4	5	2	audi 100ls	gas	std	four	sedan

5 rows × 26 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        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
```

Cars are initially assigned a risk factor symbol associated with its price. Then, if it is more risky (or less), this symbol is adjusted by moving it up (or down) the scale. This process is called **symboling**. A value of +3 indicates that the auto is risky, –3 that it is probably pretty safe.

```
df['CarName'].head(25)
```

	CarName
0	alfa-romero giulia
1	alfa-romero stelvio
2	alfa-romero Quadrifoglio
3	audi 100 ls
4	audi 100ls
5	audi fox
6	audi 100ls
7	audi 5000
8	audi 4000
9	audi 5000s (diesel)
10	bmw 320i
11	bmw 320i
12	bmw x1
13	bmw x3
14	bmw z4
15	bmw x4
16	bmw x5
17	bmw x3
18	chevrolet impala
19	chevrolet monte carlo
20	chevrolet vega 2300
21	dodge rampage
22	dodge challenger se
23	dodge d200
24	dodge monaco (sw)

dtype: object

```
cars_name= pd.Series([i.split()[0] for i in df['CarName']])
cars_name
```

	0
0	alfa-romero
1	alfa-romero
2	alfa-romero
3	audi
4	audi
...	...
200	volvo
201	volvo
202	volvo
203	volvo
204	volvo

205 rows × 1 columns

dtype: object

```
df['CarCompany']=cars_name
df.drop(columns= ['car_ID','CarName'],inplace=True)
df.head()
```

	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	engine
0	3	gas	std	two	convertible	rwd	
1	3	gas	std	two	convertible	rwd	
2	1	gas	std	two	hatchback	rwd	
3	2	gas	std	four	sedan	fwd	
4	2	gas	std	four	sedan	4wd	

5 rows × 25 columns

```
df['CarCompany'].value_counts()
```

CarCompany	count
toyota	31
nissan	17
mazda	15
honda	13
mitsubishi	13
subaru	12
volvo	11
peugeot	11
dodge	9
volkswagen	9
buick	8
bmw	8
audi	7
plymouth	7
saab	6
porsche	4
isuzu	4
alfa-romero	3
jaguar	3
chevrolet	3
renault	2
maxda	2
vw	2
mercury	1
porcshce	1
Nissan	1
toyouta	1
vokswagen	1

dtype: int64

```
df.loc[(df['CarCompany'] == "vw") | (df['CarCompany'] == "vokswagen"), 'CarC

# porsche
df.loc[(df['CarCompany']=='porcshce'),'CarCompany'] = "porsche"
# toyota
df.loc[(df['CarCompany']=='toyouta'),'CarCompany'] = "toyota"
# nissan
df.loc[(df['CarCompany']=='Nissan'),'CarCompany'] = "nissan"
# mazda
df.loc[(df['CarCompany']=='maxda'),'CarCompany'] = "mazda"

df['CarCompany'].value_counts()
```

CarCompany	count
toyota	32
nissan	18
mazda	17
mitsubishi	13
honda	13
subaru	12
volkswagen	12
volvo	11
peugeot	11
dodge	9
buick	8
bmw	8
audi	7
plymouth	7
saab	6
porsche	5
isuzu	4
alfa-romero	3
chevrolet	3
jaguar	3
renault	2
mercury	1

dtype: int64

```
df.dtypes
```

	0
symboling	int64
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
CarCompany	object

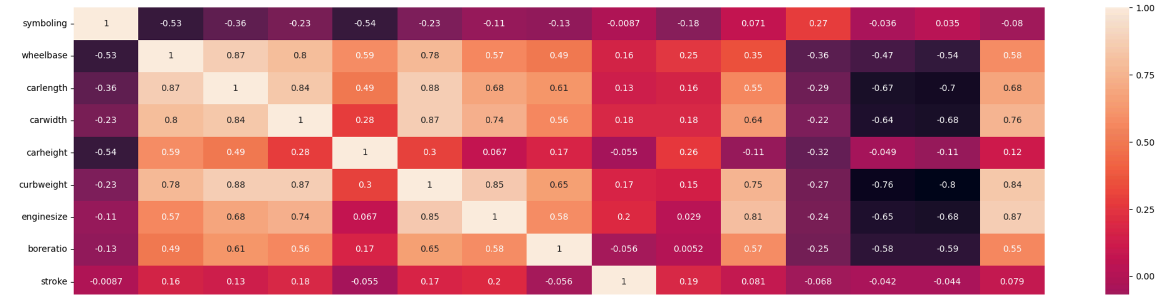
dtype: object

```
num_df= df.select_dtypes(include=['int','float'])
num_df.head()
```

	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesi
0	3	88.6	168.8	64.1	48.8	2548	1
1	3	88.6	168.8	64.1	48.8	2548	1
2	1	94.5	171.2	65.5	52.4	2823	1
3	2	99.8	176.6	66.2	54.3	2337	1
4	2	99.4	176.6	66.4	54.3	2824	1

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(25,10))
sns.heatmap(num_df.corr(),annot=True)
```

<Axes: >




```
df['doornumber'].value_counts()
```

	count
doornumber	
four	115
two	90

dtype: int64

```
df['cylindernumber'].value_counts()
```

	count
cylindernumber	
four	159
six	24
five	11
eight	5
two	4
twelve	1
three	1

dtype: int64

```
dict_words= {"two": 2, "three": 3, "four": 4, "five": 5, "six": 6, "eight":
df['doornumber']= df['doornumber'].map(dict_words)
df['cylindernumber']=df['cylindernumber'].map(dict_words)
df.head()
```

	symboling	fueltype	aspiration	doornumber	carbody	drivewheel	engine
0	3	gas	std	2	convertible	rwd	
1	3	gas	std	2	convertible	rwd	
2	1	gas	std	2	hatchback	rwd	
3	2	gas	std	4	sedan	fwd	
4	2	gas	std	4	sedan	4wd	

5 rows × 25 columns

```
obj_df= df.select_dtypes(include='object')
obj_df
```

	fueltype	aspiration	carbody	drivewheel	enginelocation	enginetype
0	gas	std	convertible	rwd	front	dohc
1	gas	std	convertible	rwd	front	dohc
2	gas	std	hatchback	rwd	front	ohcv
3	gas	std	sedan	fwd	front	ohc
4	gas	std	sedan	4wd	front	ohc
...	...	...	...	...	...	...
200	gas	std	sedan	rwd	front	ohc
201	gas	turbo	sedan	rwd	front	ohc
202	gas	std	sedan	rwd	front	ohcv
203	diesel	turbo	sedan	rwd	front	ohc
204	gas	turbo	sedan	rwd	front	ohc

205 rows × 8 columns

Choosing between **one-hot encoding** and **label encoding** depends on the nature of the categorical variable you're working with and the machine learning algorithm you plan to use.

1. Nature of the Categorical Variable:

- **Ordinal Categorical Variables:**
  - These are categories that have a natural order or ranking (e.g., "low," "medium," "high").
  - **Use Label Encoding** because the integer values assigned by label encoding can reflect the ordinal relationship between the categories.
    - Example: "low" -> 1, "medium" -> 2, "high" -> 3.
- **Nominal Categorical Variables:**
  - These are categories that do not have an inherent order or ranking (e.g., "red," "blue," "green").
  - **Use One-Hot Encoding** because label encoding might incorrectly imply a ranking or relationship between the categories that does not exist.
    - Example: "red" -> [1, 0, 0], "blue" -> [0, 1, 0], "green" -> [0, 0, 1].

2. Machine Learning Algorithm:

- **Tree-Based Algorithms (e.g., Decision Trees, Random Forests):**

- These algorithms are not sensitive to the numerical nature of the input and can often handle label-encoded data effectively.
- **Label Encoding** can be used, especially for ordinal data.

- **Linear Algorithms (e.g., Logistic Regression, Linear Regression, SVM):**
  - These algorithms assume a linear relationship and can misinterpret label-encoded values as ordinal, which can introduce bias if the data is nominal.
  - **One-Hot Encoding** is generally preferred to prevent this issue and ensure the model does not impose a false ordinal relationship.

3. Number of Categories:

- **Few Categories (e.g., < 10):**
  - **One-Hot Encoding** is manageable even with a small number of categories, and it provides clear separations between categories.
- **Many Categories (e.g., > 10):**
  - **Label Encoding** may be preferred if there are a large number of categories, as one-hot encoding would create a very high-dimensional feature space, which can lead to the "curse of dimensionality" and increased computational cost.
  - **One-Hot Encoding** can still be used, but it might require dimensionality reduction techniques afterward.

4. Risk of Introducing Bias:

- **One-Hot Encoding** is safer in avoiding bias since it treats all categories equally without implying any order or relationship.
- **Label Encoding** can introduce bias if the algorithm assumes a relationship between the encoded values.

5. Data Sparsity:

- **One-Hot Encoding** results in sparse data (many zeros), which can be computationally inefficient for large datasets with many categories. Some models, however, are optimized to handle sparse data efficiently.
- **Label Encoding** results in dense data (single column), which is more computationally efficient.

```
# One hot encoding
df['carbody'].unique()
```

```
array(['convertible', 'hatchback', 'sedan', 'wagon', 'hardtop'],
      dtype=object)
```

```
pd.get_dummies(df['carbody'],dtype=int)
```

	convertible	hardtop	hatchback	sedan	wagon
0	1	0	0	0	0
1	1	0	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	1	0
...	...	...	...	...	...
200	0	0	0	1	0
201	0	0	0	1	0
202	0	0	0	1	0
203	0	0	0	1	0
204	0	0	0	1	0

205 rows × 5 columns

```
pd.get_dummies(df[['carbody','fueltype','aspiration']],dtype=int,drop_first=
```

	carbody_hardtop	carbody_hatchback	carbody_sedan	carbody_wagon	fuelty
0	0	0	0	0	
1	0	0	0	0	
2	0	1	0	0	
3	0	0	1	0	
4	0	0	1	0	
...	...	...	...	...	
200	0	0	1	0	
201	0	0	1	0	
202	0	0	1	0	
203	0	0	1	0	
204	0	0	1	0	

205 rows × 6 columns

```
dummy_df= pd.get_dummies(obj_df,dtype=int,drop_first=True)
dummy_df
```

	fueltype_gas	aspiration_turbo	carbody_hardtop	carbody_hatchback	carb
0	1	0	0	0	
1	1	0	0	0	
2	1	0	0	1	
3	1	0	0	0	
4	1	0	0	0	
...	...	...	...	...	
200	1	0	0	0	
201	1	1	0	0	
202	1	0	0	0	
203	0	1	0	0	
204	1	1	0	0	

205 rows × 43 columns

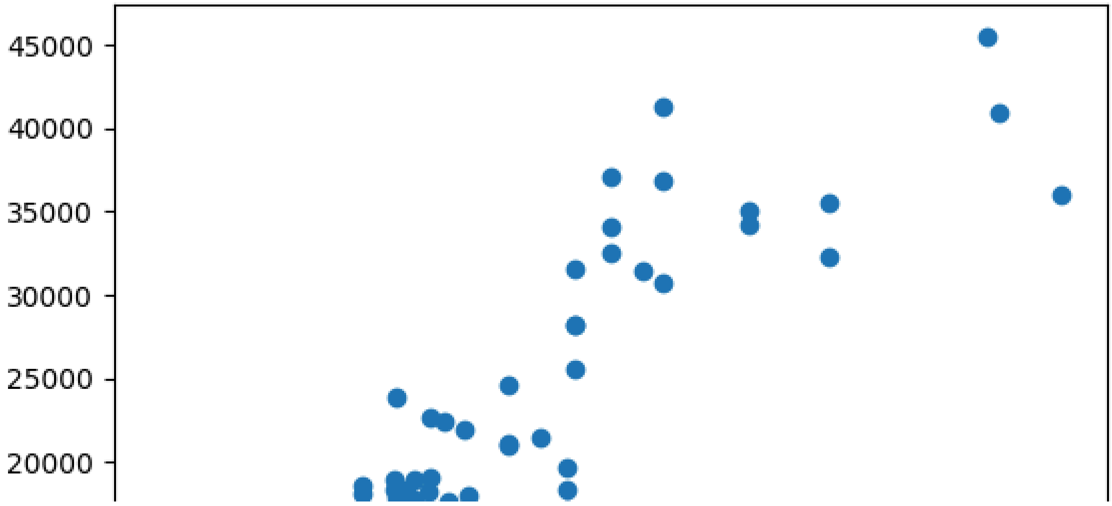
```
df.drop(columns=obj_df.columns,inplace=True)
df_new= pd.concat([df,dummy_df],axis=1)
df_new
```

	symboling	doornumber	wheelbase	carlength	carwidth	carheight	curbwe
0	3	2	88.6	168.8	64.1	48.8	
1	3	2	88.6	168.8	64.1	48.8	
2	1	2	94.5	171.2	65.5	52.4	
3	2	4	99.8	176.6	66.2	54.3	
4	2	4	99.4	176.6	66.4	54.3	
...	...	...	...	...	...	...	
200	-1	4	109.1	188.8	68.9	55.5	
201	-1	4	109.1	188.8	68.8	55.5	
202	-1	4	109.1	188.8	68.9	55.5	
203	-1	4	109.1	188.8	68.9	55.5	
204	-1	4	109.1	188.8	68.9	55.5	

205 rows × 60 columns

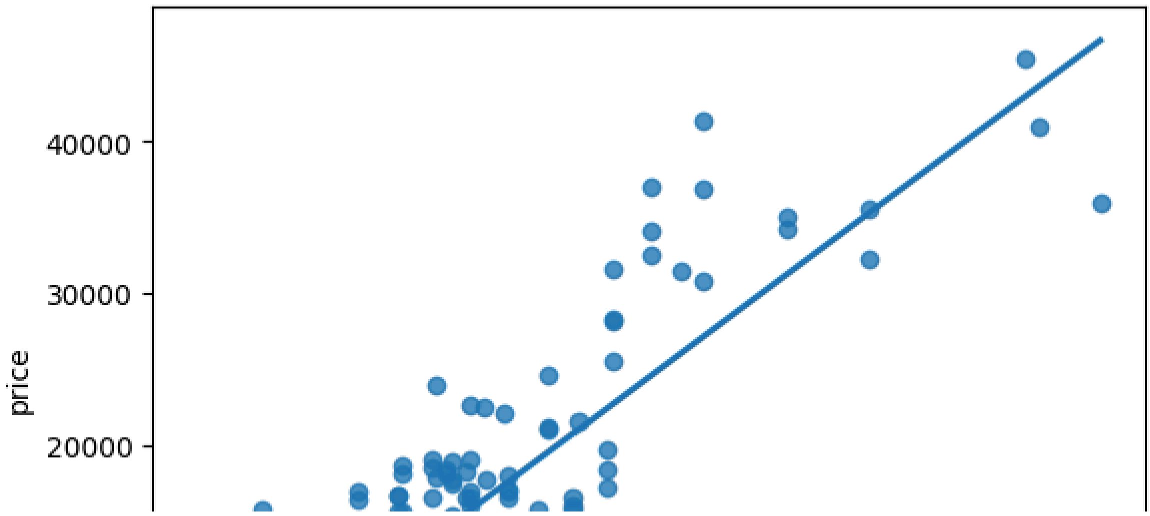
```
plt.scatter(df['enginesize'],df['price'])
```

```
<matplotlib.collections.PathCollection at 0x794c629ae6c0>
```




```
sns.regplot(x=df['enginesize'],y=df['price'],ci=None)
```

```
<Axes: xlabel='enginesize', ylabel='price'>
```



Simple Linear Regression

In simple linear regression, our job is to find this straight line which is called the **best fit line**. So we need to find the best fit line that can fit the most points in the scatter plot between the car price and the enginesize.

Let the equation of the best fit line be

$$y = mx + c$$

Here,

- y represents the price values on the y-axis
- x represents the enginesize values on the x-axis
- m is the slope of the line
- c is the intercept made by the line on the y-axis

The above equation can also be written as

$$\text{price} = m \times \text{enginesize} + c$$

Hence, for the best fit line, the slope is given as

$$m = \frac{(x_1 - \bar{x})(y_1 - \bar{y}) + (x_2 - \bar{x})(y_2 - \bar{y}) + (x_3 - \bar{x})(y_3 - \bar{y}) + \cdots + (x_n - \bar{x})(y_n - \bar{y})}{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + (x_3 - \bar{x})^2 + \cdots + (x_n - \bar{x})^2}$$

$$\Rightarrow m = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

The intercept i.e. c is given by

$$c = \bar{y} - m\bar{x}$$

Note: The differences between a value and the mean value is also referred to as **residuals** or **errors**.

```
import numpy as np
a1= np.array([1,2,3,4,5])
a2=np.array([10,11,12,13,14])

aa1,aa2,aa11,aa22=train_test_split(a1,a2,test_size=0.2,random_state=42)
aa1
```

```
array([5, 3, 1, 4])
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test= train_test_split(df['enginesize'],df['price']
```

```
def errors_product():
  prod = (X_train - X_train.mean()) * (y_train - y_train.mean())
  return prod

def squared_errors():
  sq_errors = (X_train - X_train.mean()) ** 2
  return sq_errors

slope = errors_product().sum()/ squared_errors().sum()
intercept = y_train.mean() - slope * X_train.mean()

print(f"Slope: {slope} \nIntercept: {intercept}")
```

```
Slope: 165.32203370071696
Intercept: -7590.257181325589
```

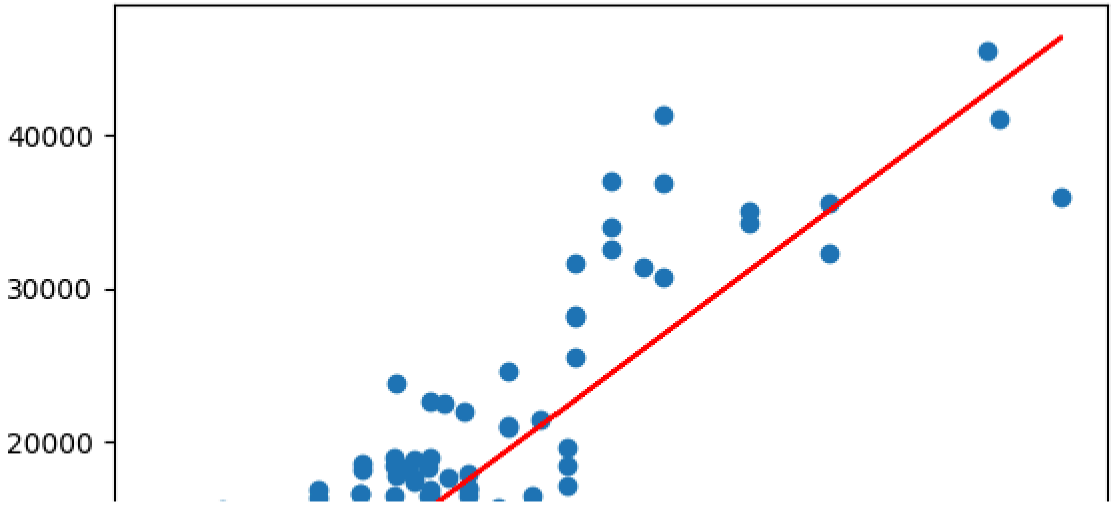
```
X_train_reshaped=X_train.values.reshape(-1,1)

lr=LinearRegression()
lr.fit(X_train_reshaped,y_train)
lr.intercept_, lr.coef_
```

```
(-7590.257181325582, array([165.3220337]))
```

```
plt.scatter(df['enginesize'],df['price'])
plt.plot(df['enginesize'],slope*df['enginesize']+intercept,color='red')
```

```
[<matplotlib.lines.Line2D at 0x7b69d56bf3a0>]
```



Model Eavaluation

The Coefficient of Determination (R-Squared)

The R-squared (R^2) tells us how much of the variance in one variable explains the variance in another variable. It is usually reported in terms of percentage.

Let's compute the coefficient of determination value which is one of the parameters that explains how much variation in one variable can be explained by the other variable through the linear regression model.

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}}$$

where

$$\text{SSE} = \sum (y - y_{\text{pred}})^2$$

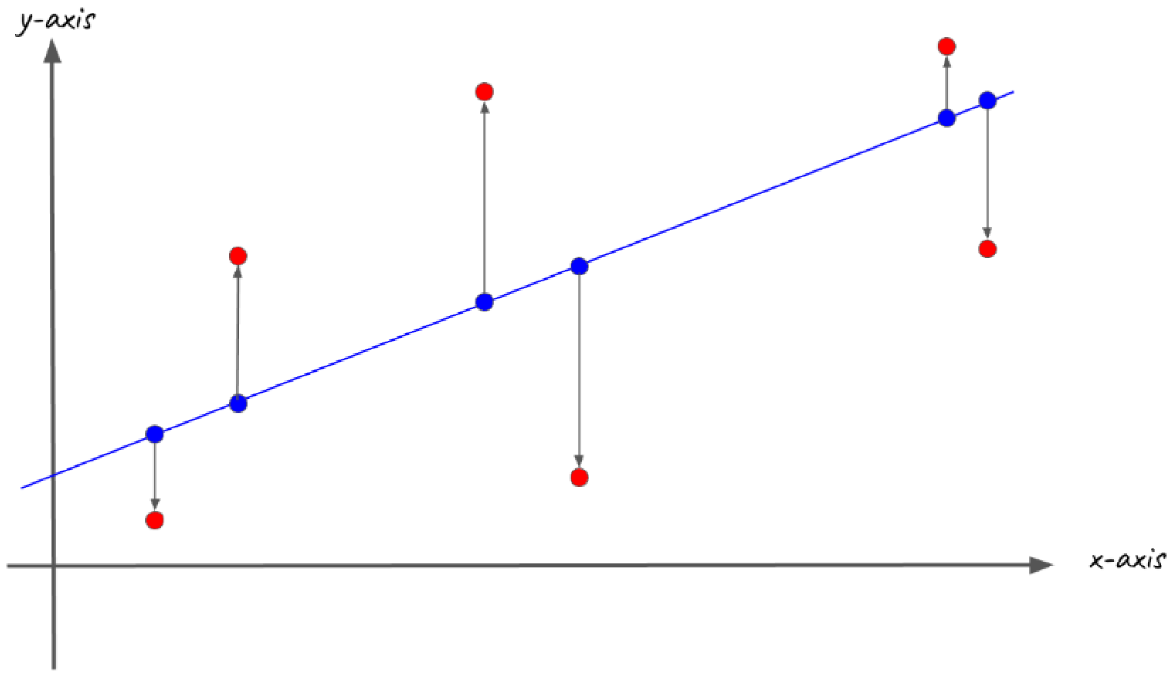
and

$$\text{SST} = \sum (y - \bar{y})^2$$

SSE stands for the sum of squared errors i.e. errors between the actual and the predicted values.

Let there be a straight line which fits them the best

The points marked with the blue colour on the straight line are the corresponding predicted values to the red-coloured points.



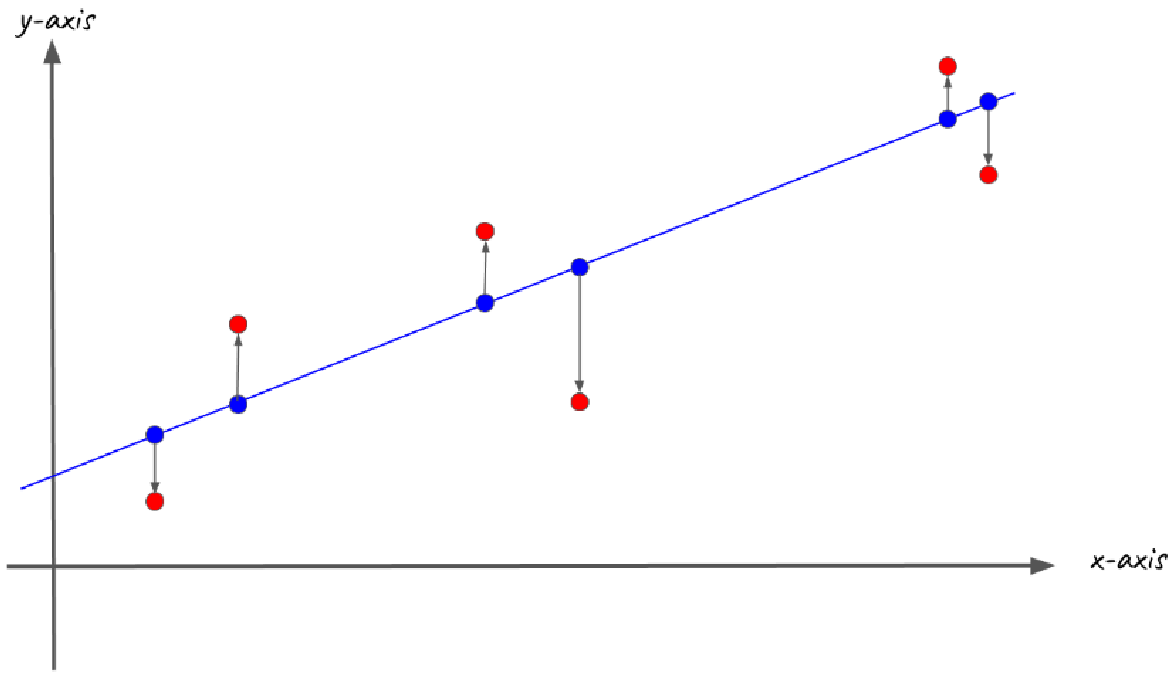
The sum of these distances is the **squared sum of errors (SSE)**. These distances would have been lower if the red-coloured points were more close to the regression line as shown in the image below.

Hence, the SSE value would have been lower. The distance between the actual values and the predicted values are given by

$$|y_i - \hat{y}_i|$$

where

- y_i is the y-coordinate of the actual value and
- $\hat{y}_i$ is the y-coordinate of the corresponding predicted value



Lower the **SSE** value, higher the R^2 value. Higher the R^2 value, better is accuracy.

SST stands for the sum of squared **total** i.e. the errors between the actual values and their mean. Consider the image shown below.

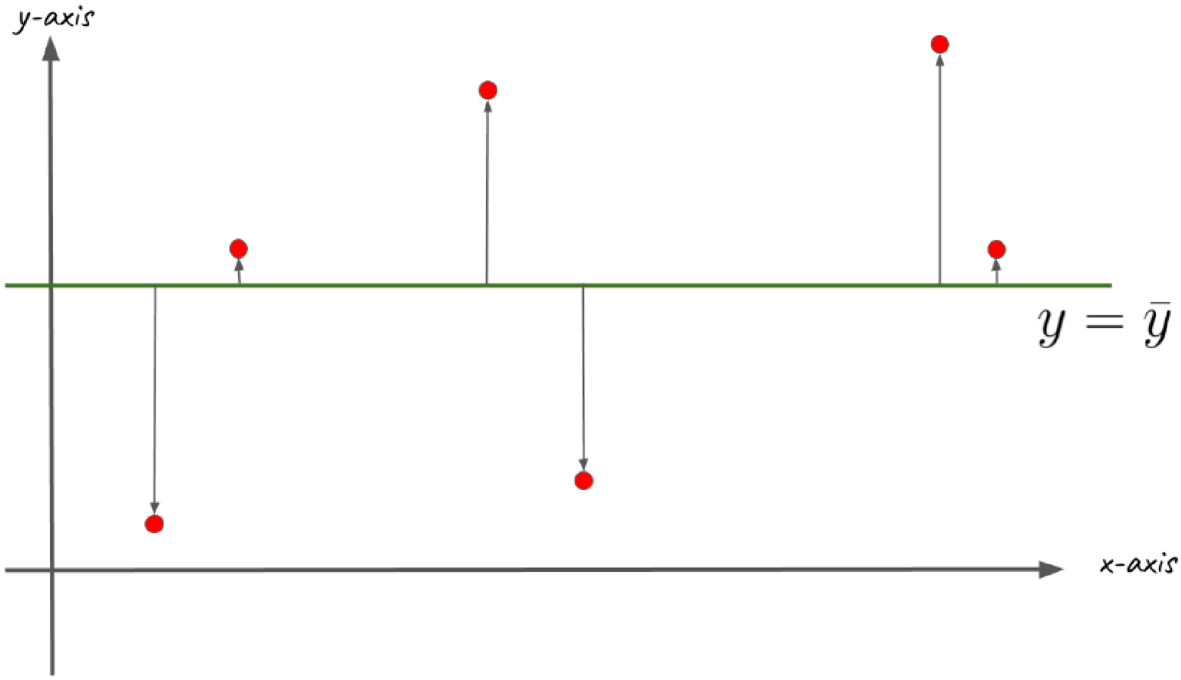
The mean of the actual target variable values i.e. $\bar{y}$ tries to fit all the points. The arrows represent the distances between the points and their mean values.

The sum of these distances is the **squared sum of total (SST)**. They are given by

$$|y_i - \bar{y}|$$

where

- y_i is the y-coordinate of the actual value and
- $\bar{y}$ is their mean value



Also, it is the maximum possible error because the mean line is the worst fit line unless the points follow uniform distribution.

Note:

1. The terms **error, residual, difference** mean the same thing.
2. It goes without saying that the R^2 value will be between 0 and 1.

```
def r_square(x,y):
  y_pred = slope * x + intercept
  sse = ((y - y_pred) **2).sum()
  sst = ((y - y.mean()) **2).sum()
  r = 1 - (sse/sst)
  return r

print(r_square(X_train, y_train))
print(r_square(X_test,y_test))
```

```
0.7650159366830336
0.7606548315153334
```

```
import numpy as np
np.corrcoef(X_train,y_train)[0,1]**2
```

```
0.7650159366830326
```

```
np.corrcoef(X_train,y_train)
```

```
array([[1.        , 0.87465189],
       [0.87465189, 1.        ]])
```

MSE, RMSE, MAE

Mean Squared Errors (MSE) is the mean of squares of the difference between the actual and the predicted values i.e.

$$\text{MSE} = \frac{1}{n}\sum(y_{\text{actual}} - y_{\text{predicted}})^2$$

where

- y_{actual} is the set of actual values of the target variable
- $y_{\text{predicted}}$ is the set of predicted values of the target variable obtained by deploying some kind of prediction model
- n is the total number of values

Root Mean Squared Errors (RMSE) is the square root of the mean squared errors (MSE) i.e.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum(y_{\text{actual}} - y_{\text{predicted}})^2}$$
$$\Rightarrow \text{RMSE} = \sqrt{\text{MSE}}$$

Mean Absolute Errors (MAE) is the mean of absolute values of the differences between the actual and the predicted values i.e.

$$\text{MAE} = \frac{1}{n}\sum\left|y_{\text{actual}} - y_{\text{predicted}}\right|$$

```
from sklearn.metrics import mean_absolute_error, mean_squared_error,mean_squ

X_test_reshaped=X_test.values.reshape(-1,1)
y_train_pred= lr.predict(X_train_reshaped)
y_test_pred= lr.predict(X_test_reshaped)

print("Train")
print(mean_absolute_error(y_train,y_train_pred))
print(mean_squared_error(y_train,y_train_pred))
print(mean_squared_log_error(y_train,y_train_pred))
print()
print('Test')

print(mean_absolute_error(y_test,y_test_pred))
print(mean_squared_error(y_test,y_test_pred))
print(mean_squared_log_error(y_test,y_test_pred))
```

```
Train
2906.3830116569407
14675971.645771094
0.07982374537928014

Test
2773.9908161019634
15661604.54844862
0.09333552536979671
```

Residual Analysis

In the residual analysis, you need to check if the error terms are normally distributed (which is infact, one of the major assumptions of linear regression). Why? Because, formally, a simple linear regression model is given as

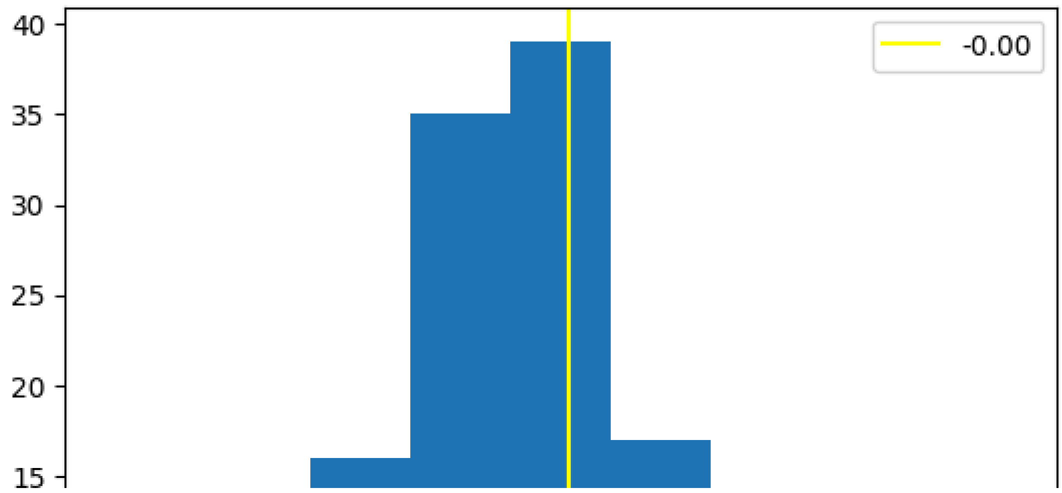
$$Y = \beta_0 + \beta_1 x + \epsilon$$

where

- x is the independent variable
- Y is the response to the independent variable (or predicted value or dependent variable)
- β_0 (intercept made by the best fit line with the y-axis) and β_1 (slope of the best fit line) are called regression coefficients
- ϵ is the random error obtained along with the predicted value

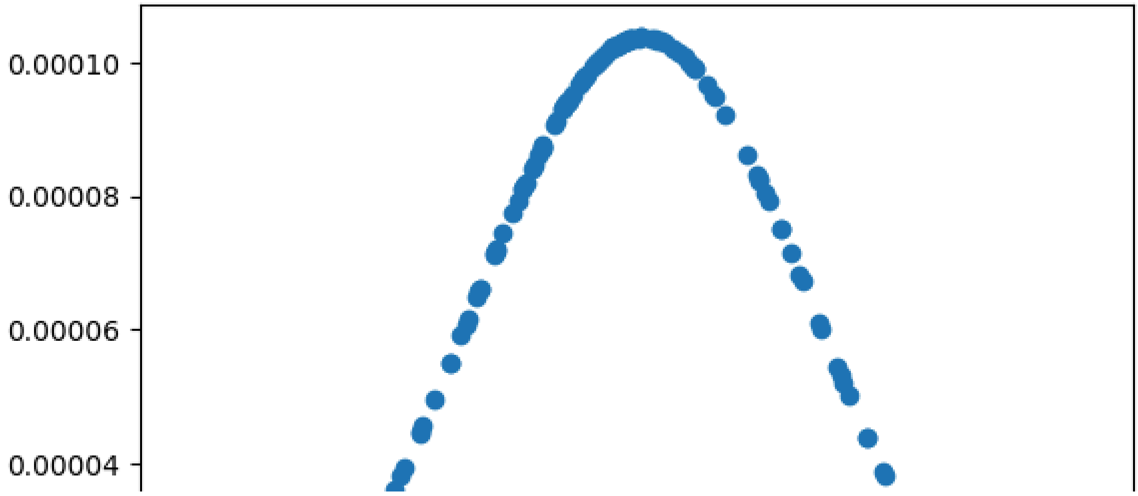
For a line to be the best fit line, the mean of random errors i.e. mean of ϵ should be 0.

```
# Residual Analysis
train_error= y_train- y_train_pred
plt.hist(train_error, bins='sturges')
plt.axvline(train_error.mean(),label=f'{train_error.mean():.2f}',color='yell
plt.legend()
plt.show()
```

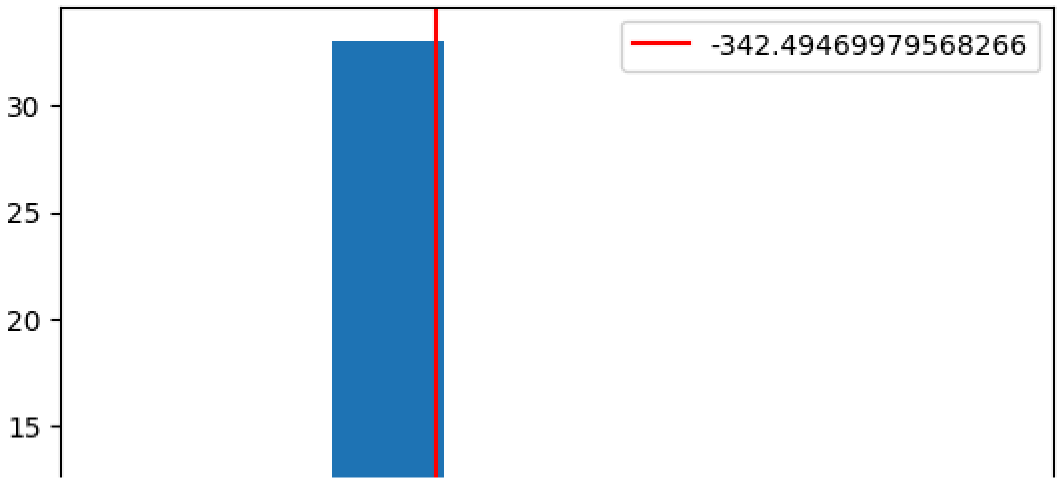


```
from scipy.stats import norm
d = norm.pdf(train_error,train_error.mean(),train_error.std())
plt.scatter(train_error,d)
```

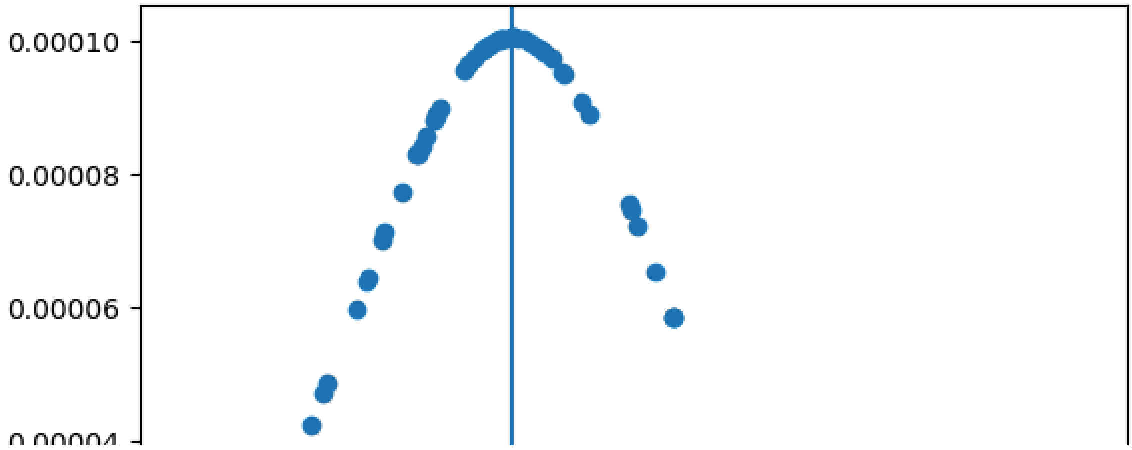
```
<matplotlib.collections.PathCollection at 0x7b69d5601c60>
```




```
test_error= y_test-y_test_pred
plt.hist(test_error,bins='sturges')
plt.axvline(test_error.mean(),label= f'{test_error.mean()}',color='red')
plt.legend()
plt.show()
```



```
d2= norm.pdf(test_error,test_error.mean(),test_error.std())
plt.scatter(test_error,d2)
plt.axvline(test_error.mean())
plt.show()
```

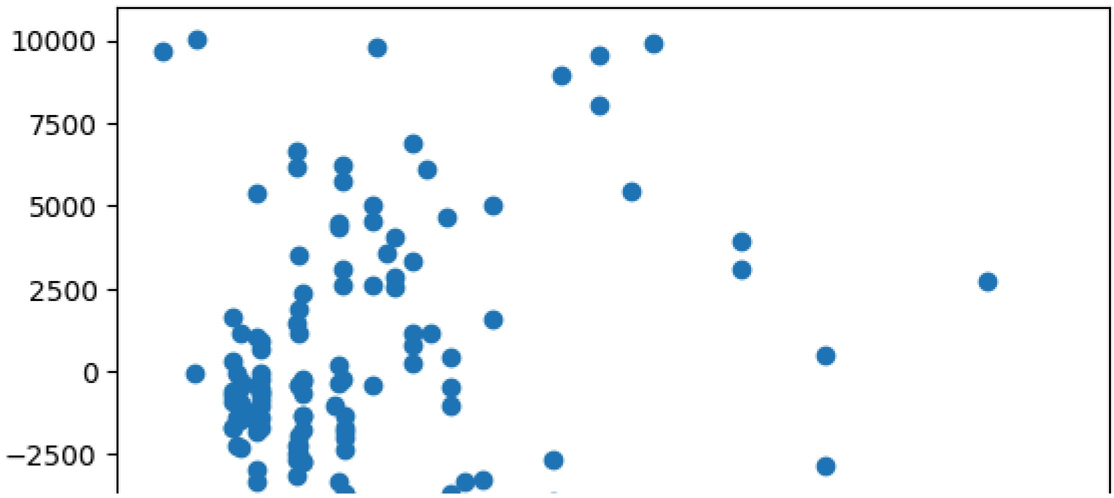


Homoscedasticity and Hetroscedasticity

Check for the trend in the scatter plot between the errors and the feature and target variables. There should not be a trend.

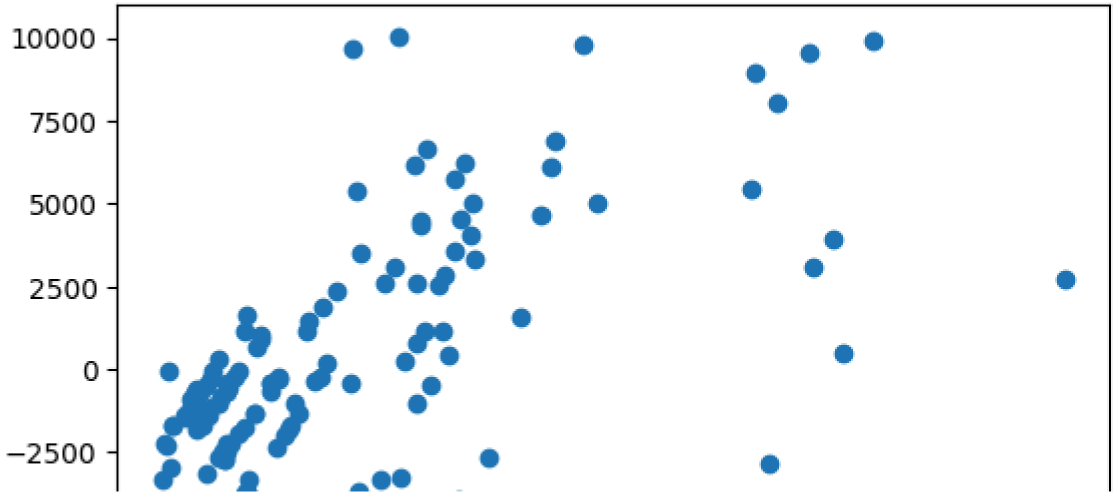
```
plt.scatter(X_train,train_error)
```

```
<matplotlib.collections.PathCollection at 0x7b69d5731b70>
```



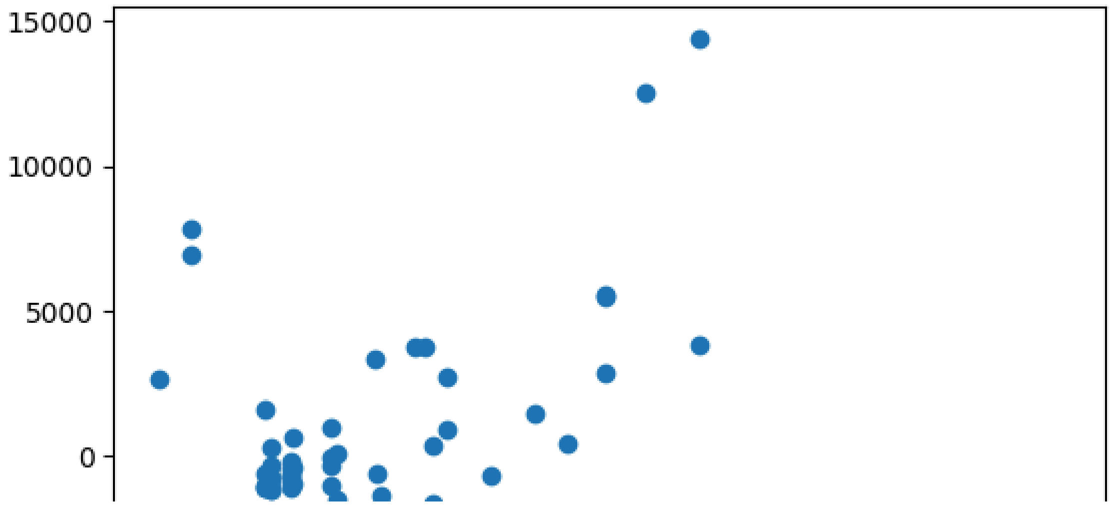
```
plt.scatter(y_train,train_error)
```

<matplotlib.collections.PathCollection at 0x7b69d54e2b30>



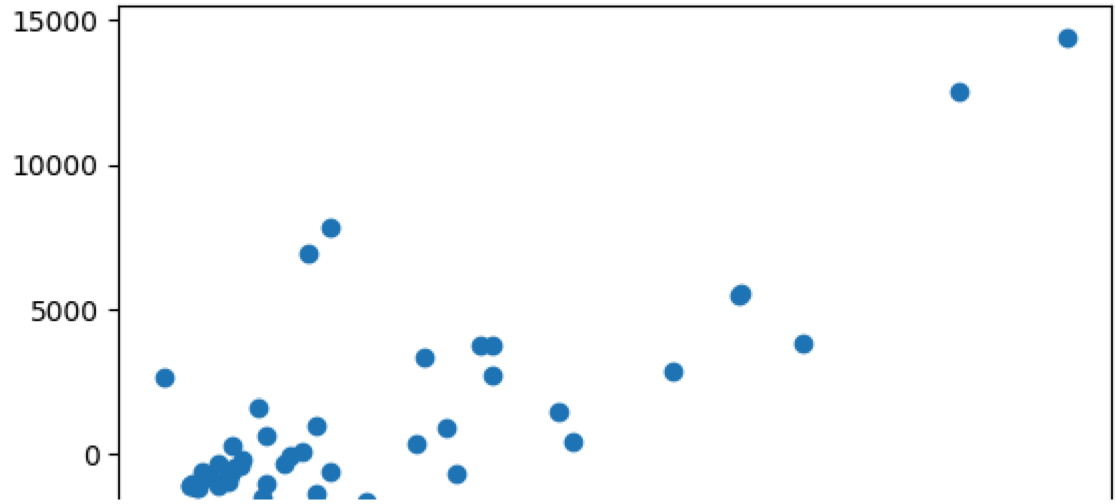

```
plt.scatter(X_test,test_error)
```

<matplotlib.collections.PathCollection at 0x7b69d557f100>



```
plt.scatter(y_test,test_error)
```

```
<matplotlib.collections.PathCollection at 0x7b69d53ebeb0>
```



Multiple Linear Regression

Preparing data

Data Scaling

```
from sklearn.preprocessing import StandardScaler
X= df_new.drop('price',axis=1)
y= df_new['price']
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.3,random_sta
X_train.head()
```

	symboling	doornumber	wheelbase	carlength	carwidth	carheight	curbwe
177	-1	4	102.4	175.6	66.5	53.9	
75	1	2	102.7	178.4	68.0	54.8	
174	-1	4	102.4	175.6	66.5	54.9	
31	2	2	86.6	144.6	63.9	50.8	
12	0	2	101.2	176.8	64.8	54.3	

5 rows × 59 columns

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 59 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   symboling           205 non-null    int64  
 1   doornumber          205 non-null    int64  
 2   wheelbase           205 non-null    float64
 3   carlength           205 non-null    float64
 4   carwidth            205 non-null    float64
 5   carheight           205 non-null    float64
 6   curbweight          205 non-null    int64  
 7   cylindernumber      205 non-null    int64  
 8   enginesize          205 non-null    int64  
 9   boreratio           205 non-null    float64
 10  stroke              205 non-null    float64
 11  compressionratio    205 non-null    float64
 12  horsepower          205 non-null    int64  
 13  peakrpm             205 non-null    int64  
 14  citympg             205 non-null    int64  
 15  highwaympg          205 non-null    int64  
 16  fueltype_gas        205 non-null    int64  
 17  aspiration_turbo    205 non-null    int64  
 18  carbody_hardtop     205 non-null    int64  
 19  carbody_hatchback   205 non-null    int64  
 20  carbody_sedan       205 non-null    int64  
 21  carbody_wagon       205 non-null    int64  
 22  drivewheel_fwd      205 non-null    int64  
 23  drivewheel_rwd      205 non-null    int64  
 24  enginelocation_rear 205 non-null    int64  
 25  enginetype_dohcv    205 non-null    int64  
 26  enginetype_l        205 non-null    int64  
 27  enginetype_ohc      205 non-null    int64  
 28  enginetype_ohcf     205 non-null    int64  
 29  enginetype_ohcv     205 non-null    int64  
 30  enginetype_rotor    205 non-null    int64  
 31  fuelsystem_2bbl     205 non-null    int64  
 32  fuelsystem_4bbl     205 non-null    int64  
 33  fuelsystem_idi      205 non-null    int64  
 34  fuelsystem_mfi      205 non-null    int64  
 35  fuelsystem_mpfi     205 non-null    int64  
 36  fuelsystem_spdi     205 non-null    int64  
```

```
 37  fuelsystem_spfi        205 non-null    int64
 38  CarCompany_audi        205 non-null    int64
 39  CarCompany_bmw         205 non-null    int64
 40  CarCompany_buick       205 non-null    int64
 41  CarCompany_chevrolet   205 non-null    int64
 42  CarCompany_dodge       205 non-null    int64
 43  CarCompany_honda       205 non-null    int64
 44  CarCompany_isuzu       205 non-null    int64
 45  CarCompany_jaguar      205 non-null    int64
 46  CarCompany_mazda       205 non-null    int64
 47  CarCompany_mercury     205 non-null    int64
 48  CarCompany_mitsubishi  205 non-null    int64
 49  CarCompany_nissan      205 non-null    int64
 50  CarCompany_peugeot     205 non-null    int64
 51  CarCompany_plymouth    205 non-null    int64
 52  CarCompany_porsche     205 non-null    int64
```

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 143 entries, 177 to 102
Data columns (total 59 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   symboling              143 non-null    int64  
 1   doornumber             143 non-null    int64  
 2   wheelbase              143 non-null    float64
 3   carlength              143 non-null    float64
 4   carwidth               143 non-null    float64
 5   carheight              143 non-null    float64
 6   curbweight             143 non-null    int64  
 7   cylindernumber         143 non-null    int64  
 8   enginesize             143 non-null    int64  
 9   boreratio              143 non-null    float64
 10  stroke                 143 non-null    float64
 11  compressionratio       143 non-null    float64
 12  horsepower             143 non-null    int64  
 13  peakrpm                143 non-null    int64  
 14  citympg                143 non-null    int64  
 15  highwaympg             143 non-null    int64  
 16  fueltype_gas           143 non-null    int64  
 17  aspiration_turbo       143 non-null    int64  
 18  carbody_hardtop        143 non-null    int64  
 19  carbody_hatchback      143 non-null    int64  
 20  carbody_sedan          143 non-null    int64  
 21  carbody_wagon          143 non-null    int64  
 22  drivewheel_fwd         143 non-null    int64  
 23  drivewheel_rwd         143 non-null    int64  
 24  enginelocation_rear    143 non-null    int64  
 25  enginetype_dohcv       143 non-null    int64  
 26  enginetype_l           143 non-null    int64  
 27  enginetype_ohc         143 non-null    int64  
 28  enginetype_ohcf        143 non-null    int64  
 29  enginetype_ohcv        143 non-null    int64  
 30  enginetype_rotor       143 non-null    int64  
 31  fuelsystem_2bbl        143 non-null    int64  
 32  fuelsystem_4bbl        143 non-null    int64  
 33  fuelsystem_idi         143 non-null    int64  
 34  fuelsystem_mfi         143 non-null    int64  
 35  fuelsystem_mpfi        143 non-null    int64  
```

```
 36  fuelsystem_spdi        143 non-null    int64
 37  fuelsystem_spfi        143 non-null    int64
 38  CarCompany_audi        143 non-null    int64
 39  CarCompany_bmw         143 non-null    int64
 40  CarCompany_buick       143 non-null    int64
 41  CarCompany_chevrolet   143 non-null    int64
 42  CarCompany_dodge       143 non-null    int64
 43  CarCompany_honda       143 non-null    int64
 44  CarCompany_isuzu       143 non-null    int64
 45  CarCompany_jaguar      143 non-null    int64
 46  CarCompany_mazda       143 non-null    int64
 47  CarCompany_mercury     143 non-null    int64
 48  CarCompany_mitsubishi  143 non-null    int64
 49  CarCompany_nissan      143 non-null    int64
 50  CarCompany_peugeot     143 non-null    int64
 51  CarCompany_plymouth    143 non-null    int64
 52  CarCompany_porsche     143 non-null    int64
```

```
ss= StandardScaler()
ss_values= ss.fit_transform(X_train[X_train.columns[:16]])
X_train[X_train.columns[:16]]= ss_values
X_train.head()
```

	symboling	doornumber	wheelbase	carlength	carwidth	carheight	curbwe
177	-1.5000	0.887412	0.573309	0.076413	0.235105	0.043859	-0.22
75	0.1250	-1.126872	0.622875	0.302880	0.924984	0.408026	0.64
174	-1.5000	0.887412	0.573309	0.076413	0.235105	0.448489	-0.17
31	0.9375	-1.126872	-2.037199	-2.430901	-0.960684	-1.210497	-1.44
12	-0.6875	-1.126872	0.375042	0.173470	-0.546757	0.205711	0.26

5 rows × 59 columns

```
import statsmodels.api as sm
X_train_sm= sm.add_constant(X_train)
model= sm.OLS(y_train,X_train_sm).fit()
model.params
```

	0
const	1.111326e+04
symboling	-2.583520e+02
doornumber	2.554028e+02
wheelbase	4.605784e+02
carlength	-6.252099e+02
carwidth	1.483336e+03
carheight	-3.289250e+02
curbweight	2.130519e+03
cylindernumber	-8.448995e+02
enginesize	4.068721e+03
boreratio	-1.024074e+03
stroke	-7.567151e+01
compressionratio	4.636386e+02
horsepower	2.553877e+02
peakrpm	8.288722e+02
citympg	6.368825e+02
highwaympg	-2.391232e+02
fueltype_gas	6.290680e+03
aspiration_turbo	2.528179e+03
carbody_hardtop	-2.337527e+02
carbody_hatchback	-3.517083e+03
carbody_sedan	-3.523827e+03
carbody_wagon	-4.154162e+03
drivewheel_fwd	-8.505823e+02
drivewheel_rwd	-1.393007e+03
enginelocation_rear	4.920851e+03
enginetype_dohcv	2.908615e+02
enginetype_l	-1.259194e+03
enginetype_ohc	-3.180058e+02
enginetype_ohcf	1.138945e+03
enginetype_ohcv	-1.391856e+03

```
print(model.summary())
```

	0
fuelsystem_2bbl	2.404488e+03
fuelsystem_4bbl	1.132048e+03
fuelsystem_idi	4.822583e+03
fuelsystem_mfi	-1.026930e-12
fuelsystem_mpfi	1.568049e+03
fuelsystem_spdi	1.158155e+03
fuelsystem_spfi	2.039415e+03
CarCompany_audi	1.496863e+01
CarCompany_bmw	7.272259e+03
CarCompany_buick	6.875953e+03
CarCompany_chevrolet	-4.172989e+03
CarCompany_dodge	-4.140191e+03
CarCompany_honda	-2.024895e+03
CarCompany_isuzu	-1.568795e+03
CarCompany_jaguar	1.491049e+03
CarCompany_mazda	-2.504620e+03
CarCompany_mercury	-1.594039e+03
CarCompany_mitsubishi	-4.572367e+03
CarCompany_nissan	-3.072283e+03
CarCompany_peugeot	-1.258194e+03
CarCompany_plymouth	-4.405861e+03
CarCompany_porsche	5.532113e+03
CarCompany_renault	-3.693763e+03
CarCompany_saab	8.094785e+02
CarCompany_subaru	-3.781906e+03
CarCompany_toyota	-2.751770e+03
CarCompany_volkswagen	-1.847204e+03
CarCompany_volvo	2.379138e+02

dtype: float64

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.9
Model:                            OLS   Adj. R-squared:                  0.9
Method:                 Least Squares   F-statistic:                     67
Date:                Sat, 07 Sep 2024   Prob (F-statistic):           3.52e
Time:                        03:28:27   Log-Likelihood:                -1214
No. Observations:                 143   AIC:                             254
Df Residuals:                      87   BIC:                             270
Df Model:                          55
Covariance Type:            nonrobust
==============================================================================
                           coef    std err          t      P>|t|      [0.02
------------------------------------------------------------------------------
const                 1.111e+04   2193.733      5.066      0.000    6752.98
symboling             -258.3520    337.076     -0.766      0.445    -928.32
doornumber             255.4028    275.521      0.927      0.357    -292.22
wheelbase              460.5784    667.071      0.690      0.492    -865.29
carlength             -625.2099    673.875     -0.928      0.356   -1964.66
carwidth              1483.3360    501.492      2.958      0.004     486.56
carheight             -328.9250    360.885     -0.911      0.365   -1046.22
curbweight            2130.5189    867.221      2.457      0.016     406.82
cylindernumber        -844.8995    720.653     -1.172      0.244   -2277.27
enginesize            4068.7211   1079.175      3.770      0.000    1923.74
boreratio            -1024.0736    376.775     -2.718      0.008   -1772.95
stroke                 -75.4971    282.058     -0.268      0.789    -636.29
compressionratio       463.6386   1675.384      0.277      0.783   -2866.36
horsepower             255.3877    829.610      0.308      0.759   -1393.55
peakrpm                828.8722    325.351      2.548      0.013     182.26
citympg                636.8825    850.582      0.749      0.456   -1053.74
highwaympg            -239.1232    766.429     -0.312      0.756   -1762.48
fueltype_gas          6290.6800   2516.294      2.500      0.014    1289.27
aspiration_turbo      2528.1790    820.112      3.083      0.003     898.11
carbody_hardtop       -233.7527   1674.144     -0.140      0.889   -3561.29
carbody_hatchback    -3517.2839   1136.021     -3.096      0.003   -5775.04
carbody_sedan        -3523.8275   1243.949     -2.833      0.006   -5996.31
carbody_wagon        -4154.1619   1363.304     -3.047      0.003   -6863.87
drivewheel_fwd        -850.5823   1150.963     -0.739      0.462   -3138.24
drivewheel_rwd       -1393.0072   1327.046     -1.050      0.297   -4030.65
enginelocation_rear   4920.8510   1878.236      2.620      0.010    1187.65
enginetype_dohcv       290.8615   3118.387      0.093      0.926   -5907.27
enginetype_l         -1259.1940    958.461     -1.314      0.192   -3164.24
enginetype_ohc        -318.0058    902.080     -0.353      0.725   -2110.98
enginetype_ohcf       1138.7849   1059.824      1.075      0.286    -967.57
enginetype_ohcv      -1391.8556   1301.970     -1.069      0.288   -3979.66
enginetype_rotor      6744.3881   2491.565      2.707      0.008    1792.13
fuelsystem_2bbl       2404.4879   1359.986      1.768      0.081    -298.63
fuelsystem_4bbl       1132.0482   2592.199      0.437      0.663   -4020.22
fuelsystem_idi        4822.5828   4052.732      1.190      0.237   -3232.66
fuelsystem_mfi        -1.027e-12   1.89e-12     -0.543      0.589   -4.79e-1
fuelsystem_mpfi       1568.0491   1482.816      1.057      0.293   -1379.26
fuelsystem_spdi       1158.1552   1718.412      0.674      0.502   -2257.37
fuelsystem_spfi       2039.4149   2480.546      0.822      0.413   -2890.93
CarCompany_audi         14.9686   2053.835      0.007      0.994   -4067.25
CarCompany_bmw        7272.2594   1965.043      3.701      0.000    3366.52
CarCompany_buick      6875.9526   2219.913      3.097      0.003    2463.63
CarCompany_chevrolet -4172.9890   2343.759     -1.780      0.078   -8831.46
```

Adjusted R^2

In the case of multiple linear regression, **adjusted** R^2 value takes precedence over the R^2 value. It is calculated as:

$$R^2_{\text{adj}} = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

where

- R^2 is the coefficient of determination
- N is number of instances (or rows) in the dataset
- p is the number of independent variables (excluding constant) in the dataset

the R^2_{adj} will always be less than or equatl to the R^2 value i.e. $R^2_{\text{adj}} \leq R^2$.

Why adjusted R-squared is a better metric in multiple linear regression?

As you add more and more independent variables, the R^2 squared values increases even if the independent variable has no contribution in predicting the values of the target variable. Hence, the adjusted R^2 value penalises the unnecessary inclusion of more independent variables.

So, if adding more independent (or feature) variables leads to an increase in the adjusted R^2 value, then it is a good sign. However, if adding more independent (or feature) variables leads to a decrease in the adjusted R^2 value, it is a bad sign.

In this case, the R^2_{adj} is quite high but the p-values for many of the columns is greater than 0.05 which is not a good sign. It means, these variables are insignificant in predicting the price of a car. Also, if we calculate variance inflation factor values for these columns, they would be very very high than 10.

Ordinary Least Squares (OLS)

Consider the regression equation

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_k x_k + \epsilon$$

where

- $x_1, x_2, x_3, \ldots, x_k$ are independent variables or features
- Y is the response to the independent variable (or predicted value or dependent variable)
- $\beta_0, \beta_1, \beta_2, \ldots, \beta_k$ are the corresponding regression coefficients of the independent variables

- ϵ is the random error obtained along with the predicted value which follows normal distribution with mean 0 and some standard deviation of σ

The parameters $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ and σ are assumed to be unknown and must be estimated from the data, which we shall suppose will consist of the values of $Y_1, Y_2, Y_3, \ldots, Y_n$ where Y_i is the response level corresponding to the k features $x_{i1}, \ldots, x_{i2}, \ldots, x_{ik}$. That is, the Y_i are related to these features through

$$E[Y_i] = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \cdots \\ + \beta_k x_{ik}$$

where

- $E[Y_i]$ means **expected value** for an instance i. In simple terms, instance or i denotes a row in a data frame
- x_{i1} denotes item at the i^{th} row in the 1^{st} column in a data frame having only features
- x_{i2} denotes item at the i^{th} row in the 2^{nd} column in a data frame having only features
- x_{i3} denotes item at the i^{th} row in the 3^{rd} column in a data frame having only features

  $\ldots$
- x_{ik} denotes item at the i^{th} row in the k^{th} column in a data frame having only features

As we said earlier, the difference between the actual and the predicted values should be 0 or close to 0 for an accurate prediction model i.e.

$$\begin{gathered} Y_1 - E[Y_1] \approx 0 \\ Y_2 - E[Y_2] \approx 0 \\ Y_3 - E[Y_3] \approx 0 \\ \vdots \\ Y_N - E[Y_N] \approx 0 \end{gathered}$$

where N is the total number of instances (or rows in a data frame).

The OLS says that the sum of squares of all these errors i.e.

$$J = (Y_1 - E[Y_1])^2 + (Y_2 - E[Y_2])^2 \\ + (Y_3 - E[Y_3])^2 + \cdots + (Y_N - E[Y_N])^2$$

should be the least or minimum.

The above expression can be compressed as

$$J = \sum_{i=1}^{N} (Y_i - E[Y_i])^2$$

So in general, it can be written as

$$J(\beta, x) = \sum_{i=1}^{N} (Y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} \\ - \beta_3 x_{i3} - \cdots - \beta_k x_{ik})^2$$

where $J(\beta, x)$ denotes the sum of the squared errors is dependent on the coefficients ($\beta_0, \beta_1, \beta_2, \ldots, \beta_k$) and features ($x_1, x_2, x_3, \ldots, x_k$)

To find the points of maxima (peak) or minima (valley), we differentiate a mathematical function w.r.t. independent variable and equate the result obtained to 0 because the slope of a curve at the point of maxima (peak) or minima (valley) is 0. Differentiation (or derivative) represents slope at a point.

In the above equation, all the x quantities are known quantities as we have seen earlier. So the $\beta_0, \beta_1, \beta_2, \ldots, \beta_k$ are unknown quantities. Thus, they are independent variables.

Here $\beta_0, \beta_1, \beta_2, \ldots, \beta_k$ are independent of each other. Hence, we can do partial differentiation w.r.t. to each of the betas independently.

Let's differentiate $J(\beta, x)$ w.r.t. β_0. So every other term apart from β_0 will be treated as a constant. And the differentiation (or derivative) of a constant is 0.

$$\frac{\partial J}{\partial \beta_0} = 2 \sum_{i=1}^{N} (Y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} \\ - \beta_3 x_{i3} - \cdots - \beta_k x_{ik})(-1) = 0$$

$$\Rightarrow \sum_{i=1}^{N} (Y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \beta_3 x_{i3} - \\ \cdots - \beta_k x_{ik}) = 0$$

Similarly,

$$\frac{\partial J}{\partial \beta_1} = 2 \sum_{i=1}^{N} (Y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} \\ - \beta_3 x_{i3} - \cdots - \beta_k x_{ik})(-x_{i1}) = 0$$

$$\Rightarrow \sum_{i=1}^{N} (Y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \beta_3 x_{i3} - \\ \cdots - \beta_k x_{ik}) x_{i1} = 0$$

$$\frac{\partial J}{\partial \beta_2} = 2\sum_{i=1}^{N}(Y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \beta_3 x_{i3} - \cdots - \beta_k x_{ik})(-x_{i2}) = 0$$

$$\Rightarrow \sum_{i=1}^{N}(Y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \beta_3 x_{i3} - \cdots - \beta_k x_{ik})x_{i2} = 0$$

$$\vdots$$

$$\frac{\partial J}{\partial \beta_k} = 2\sum_{i=1}^{N}(Y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \beta_3 x_{i3} - \cdots - \beta_k x_{ik})(-x_{ik}) = 0$$

$$\Rightarrow \sum_{i=1}^{N}(Y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \beta_3 x_{i3} - \cdots - \beta_k x_{ik})x_{ik} = 0$$

On further reducing the above $k+1$ equations, we get

$$\sum_{i=1}^{N} Y_i = N\beta_0 + \beta_1 \sum_{i=1}^{N} x_{i1} + \beta_2 \sum_{i=1}^{N} x_{i2} + \cdots + \beta_k \sum_{i=1}^{N} x_{ik}$$

$$\sum_{i=1}^{N} Y_i x_{i1} = \beta_0 \sum_{i=1}^{N} x_{i1} + \beta_1 \sum_{i=1}^{N} x_{i1}^2 + \beta_2 \sum_{i=1}^{N} x_{i1}x_{i2} + \cdots + \beta_k \sum_{i=1}^{N} x_{i1}x_{ik}$$

$$\vdots$$

$$\sum_{i=1}^{N} Y_i x_{ik} = \beta_0 \sum_{i=1}^{N} x_{ik} + \beta_1 \sum_{i=1}^{N} x_{ik}x_{i1} + \beta_2 \sum_{i=1}^{N} x_{ik}x_{i2} + \cdots + \beta_k \sum_{i=1}^{N} x_{ik}^2$$

Now we have $k+1$ linear equations having $k+1$ unknowns i.e. $\beta_0, \beta_1, \beta_2, \ldots, \beta_k$. By solving these $k+1$ equations, we can get the beta values. This is exactly the same as solving two linear equations having two unknowns. For e.g., the solution to the two linear equations

$$8\beta_0 + 7\beta_1 = 38 \text{ and } 3\beta_0 - 5\beta_1 = -1$$

is

$$\beta_0 = 3 \text{ and } \beta_1 = 2$$

So all-in-all, **ordinary least squares** says that **find the values of the coefficients ($\beta_0, \beta_1, \beta_2, \ldots, \beta_k$) such that the sum of the squares of differences between the actual values and the predicted values is minimum.**

To solve $k + 1$ linear equations having $k + 1$ unknowns, you need to know matrices.

The above $k + 1$ linear equations can also be written as

$$\begin{bmatrix}\sum_{i = 1}^{N}Y_i \\ \sum_{i = 1}^{N}Y_i x_{i1} \\ \sum_{i = 1}^{N}Y_i x_{i2} \\ \vdots \\ \sum_{i = 1}^{N}Y_i x_{ik} \\\end{bmatrix} = \begin{bmatrix} \sum_{i = 1}^{N}1 & \sum_{i = 1}^{N}x_{i1} & \sum_{i = 1}^{N}x_{i2} & \dots & \sum_{i = 1}^{N}x_{ik} \\ \sum_{i = 1}^{N}x_{i1} & \sum_{i = 1}^{N}x_{i1}^2 & \sum_{i = 1}^{N}x_{i1}x_{i2} & \dots & \sum_{i = 1}^{N}x_{i1}x_{1k} \\ \sum_{i = 1}^{N}x_{i2} & \sum_{i = 1}^{N}x_{i1} x_{i2} & \sum_{i = 1}^{N}x_{i2}^2 & \dots & \sum_{i = 1}^{N}x_{i2} x_{ik} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\\sum_{i = 1}^{N}x_{ik} & \sum_{i = 1}^{N}x_{ik} x_{i1} & \sum_{i = 1}^{N}x_{ik} x_{i2} & \dots & \sum_{i = 1}^{N}x_{ik}^2 \\\end{bmatrix}\begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \\\end{bmatrix}$$

in the matrix form. The above matrix equation can also be written as

$$X^T Y = X^T X B$$

or

$$X^T X B = X^T Y$$

where

$$X = \begin{bmatrix}1 & x_{11} & x_{12} & x_{13} & \dots & x_{1k} \\ 1 & x_{21} & x_{22} & x_{23} & \dots & x_{2k} \\ 1 & x_{31} & x_{32} & x_{33} & \dots & x_{3k} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\1 & x_{N1} & x_{N2} & x_{N3} & \dots & x_{Nk} \\\end{bmatrix}$$

$$X^T = \begin{bmatrix}1 & 1 & 1 & \dots & 1 \\ x_{11} & x_{21} & x_{31} & \dots & x_{N1} \\ x_{12} & x_{22} & x_{32} & \dots & x_{N2} \\ x_{13} & x_{23} & x_{33} & \dots & x_{N3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\x_{1k} & x_{2k} & x_{3k} & \dots & x_{Nk} \\\end{bmatrix}$$

$$Y = \begin{bmatrix}Y_1 \\ Y_2 \\ Y_3 \\ \vdots \\ Y_N \\\end{bmatrix} \space{} \text{and} \space{} B = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \\\end{bmatrix}$$

In the matrix equation,

$$X^T X B = X^T Y$$

to obtain only the matrix B on the left-hand side, you need to multiply both the sides by $(X^T X)^{-1}$, i.e.

$$(X^T X)^{-1} X^T X B = (X^T X)^{-1} X^T Y$$

To simply the above equation, let

$$Z = X^T X$$

$$\therefore Z^{-1} = (X^T X)^{-1}$$

Hence, the above equation becomes

$$Z^{-1} Z B = Z^{-1} X^T Y$$

$$\Rightarrow IB = Z^{-1} X^T Y \space{} \space{} [\because Z^{-1}Z = I]$$

$$\Rightarrow B = Z^{-1} X^T Y \space{} \space{} [\because IB = B]$$

Let $U = X^T Y$

$$\therefore B = Z^{-1} U$$

Now, you need to obtain the Z^{-1} and multiply it with the matrix U to estimate the values of betas using the matrix operations only. But before that, you need to add a new column to the matrix X, i.e., `X_train`. All the items of this new column should be 1.

Variance Inflation factor (VIF)

Measure of Multicollinearity

Variance Infation Factor (VIF) is a way to detect multicollinearity between independent variables in a dataset. We calculate the VIF values to measure the extent of multicollinearity between the independent variables.

For $k$$k$ different independent variables, we can calculate $k$$k$ different VIFs (one for each $x_i$$x_i$ where $i = 1, 2, 3, \dots, k$$i = 1, 2, 3, \dots, k$) in three steps:

Step one

First, build a multiple linear regression model wherein $x_i$$x_i$ is a target variable and it is a function of all the other feature variables as illustrated in the equation below.

$$x_1 = \beta_0^* + \beta_2^* x_2 + \beta_3^* x_3 + \beta_4^* x_4 + \dots + \beta_k^* x_k + \epsilon^*$$$$x_1 = \beta_0^* + \beta_2^* x_2 + \beta_3^* x_3 + \beta_4^* x_4 + \dots + \beta_k^* x_k + \epsilon^*$$

Here,

- $x_1$$x_1$ is a feature acting as the target (or dependent) variable in above equation
- $x_2, x_3, x_4, \dots , x_k$$x_2, x_3, x_4, \dots , x_k$ are independent variables or features
- $\beta_0^*, \beta_2^*, \beta_3^*, \dots, \beta_k^*$$\beta_0^*, \beta_2^*, \beta_3^*, \dots, \beta_k^*$ are the corresponding regression coefficients of the independent variables in the above linear regression equation
- **$\epsilon^*$$\epsilon^*$** is the random error obtained along with the predicted value

Step two

Then, calculate the VIF for $x_{i}$$x_{i}$ using the following formula:

$$\text{VIF}_{i} = \frac{1}{1-R_{i}^{2}}$$$$\text{VIF}_{i} = \frac{1}{1-R_{i}^{2}}$$

where $R^2 _i$$R^2 _i$ is the coefficient of determination of the regression equation in step one, with $x_{1}$$x_{1}$ on the left hand side, and all other independent variables on the right hand side.

Step three

Analyse the extent of multicollinearity by considering the magnitude of the $\text{VIF}_{i}$$\text{VIF}_{i}$. **A rule of thumb is that if $\text{VIF}_{i} > 10$$\text{VIF}_{i} > 10$, then multicollinearity is high. In that case, the $x_i$$x_i$ feature must be dropped to predict the values of the target (or dependent) variable.** A cutoff of 5 is also commonly used.

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
X_train_new= X_train[['enginesize','curbweight','horsepower','citympg']]
X_train_new_sm= sm.add_constant(X_train_new)
model_new= sm.OLS(y_train,X_train_new_sm)
vif_df= pd.DataFrame()
vif_df['Features']= X_train_new_sm.columns
vif_df['VIF']= [variance_inflation_factor(X_train_new_sm.values, i) for i in
vif_df
```

	Features	VIF
0	const	1.000000
1	enginesize	4.901158
2	curbweight	4.732600
3	horsepower	4.533126
4	citympg	3.942814

Calculating VIF for 'enginesize'

```
X_train_eng= X_train_new.drop('enginesize',axis=1)
y_train_eng= X_train_new['enginesize']
X_train_eng_sm=sm.add_constant(X_train_eng)
model_eng=sm.OLS(y_train_eng,X_train_eng_sm).fit()
print(model_eng.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             enginesize   R-squared:                       0.79
Model:                            OLS   Adj. R-squared:                  0.79
Method:                 Least Squares   F-statistic:                     180.
Date:                Sat, 07 Sep 2024   Prob (F-statistic):           8.94e-4
Time:                        03:28:27   Log-Likelihood:               -89.26
No. Observations:                 143   AIC:                             186.
Df Residuals:                     139   BIC:                             198.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975
------------------------------------------------------------------------------
const       2.949e-17      0.038    7.7e-16      1.000      -0.076       0.07
curbweight     0.6446      0.063     10.248      0.000       0.520       0.76
horsepower     0.5234      0.068      7.647      0.000       0.388       0.65
citympg        0.2421      0.073      3.305      0.001       0.097       0.38
==============================================================================
Omnibus:                       33.197   Durbin-Watson:                   2.11
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              154.01
Skew:                           0.672   Prob(JB):                     3.61e-3
Kurtosis:                       7.903   Cond. No.                         3.7
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
```

```
print(f"VIF value for enginesize is {1/(1-0.8)}")
```

```
VIF value for enginesize is 5.000000000000001
```

```
# VIF for all the features
vif_df= pd.DataFrame()
vif_df['Features']= X_train_sm.columns
vif_df['VIF']= [variance_inflation_factor(X_train_sm.values, i) for i in ran
vif_df
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/regression/linear_model.p
  return 1 - self.ssr/self.centered_tss
/usr/local/lib/python3.10/dist-packages/statsmodels/stats/outliers_influence.
  vif = 1. / (1. - r_squared_i)
/usr/local/lib/python3.10/dist-packages/statsmodels/regression/linear_model.p
  return 1 - self.ssr/self.centered_tss
/usr/local/lib/python3.10/dist-packages/pandas/core/nanops.py:1010: RuntimeWa
  sqr = _ensure_numeric((avg - values) ** 2)
```

	Features	VIF
0	const	0.000000
1	symboling	7.083662
2	doornumber	4.732720
3	wheelbase	27.742502
4	carlength	28.311274
5	carwidth	15.679405
6	carheight	8.119666
7	curbweight	46.887888
8	cylindernumber	32.378272
9	enginesize	72.608025
10	boreratio	8.850441
11	stroke	4.959952
12	compressionratio	174.996708
13	horsepower	42.909032
14	peakrpm	6.599422
15	citympg	45.105971
16	highwaympg	36.622279
17	fueltype_gas	inf
18	aspiration_turbo	6.422393
19	carbody_hardtop	4.751051
20	carbody_hatchback	17.752973
21	carbody_sedan	24.107649
22	carbody_wagon	12.137629
23	drivewheel_fwd	20.113183
24	drivewheel_rwd	25.986411
25	enginelocation_rear	inf
26	enginetype_dohcv	4.209954
27	enginetype_l	inf

Note: 'inf' and 'NaN' values may occur due to multicollinearity.

28	enginetype_ohc	10.221547
29	enginetype_ohcf	inf
30	enginetype_ohcv	6.232784
31	enginetype_[illegible]	5.337314
32	fuelsystem_2bbl	24.247454
33	fuelsystem_4bbl	2.909069
34	fuelsystem_idi	inf
35	fuelsystem_mfi	NaN
36	fuelsystem_mpfi	34.268449
37	fuelsystem_spdi	7.400405
38	fuelsystem_spfi	2.663863
39	CarCompany_audi	8.873794
40	CarCompany_bmw	9.677094
41	CarCompany_buick	8.353640
42	CarCompany_chevrolet	2.378173
43	CarCompany_dodge	7.961824
44	CarCompany_honda	17.971016
45	CarCompany_isuzu	4.818289
46	CarCompany_jaguar	8.392491
47	CarCompany_mazda	11.187804
48	CarCompany_mercury	2.675882
49	CarCompany_mitsubishi	10.334005
50	CarCompany_nissan	8.293474
51	CarCompany_peugeot	inf
52	CarCompany_plymouth	7.698632
53	CarCompany_porsche	10.565245
54	CarCompany_renault	3.867446
55	CarCompany_saab	7.927021
56	CarCompany_subaru	inf
57	CarCompany_toyota	16.053616
58	CarCompany_volkswagen	11.491334
59	CarCompany_volvo	12.825491

Understanding Hypothesis Testing

From the summary report of the linear regression, you may observe that each feature variable has a **p-value** `(P>|t|)` associated with it. The p-value is one of the important statistics which can be used to eliminate features which are not relatively significant in our model.

Hypothesis Testing

Hypothesis Testing is basically testing an assumption that we make about a parameter. This assumption may or may not be true.

The steps followed in hypothesis testing are:

1. An initial assumption or hypothesis is made.
2. The validity of that hypothesis is tested.
3. If the hypothesis is found to be true, it is accepted otherwise it is rejected.

There are two types of hypothesis:

1. **Null hypothesis:** denoted by H_0, is a general statement or an initial assumption which we make about a parameter.
2. **Alternative hypothesis:** denoted by H_1 or H_a, It is contrary to the null hypothesis. It is the hypothesis we would accept if our null hypothesis is found to be false.

In hypothesis testing, we need to gather enough evidence to either accept or reject our null hypothesis. There are two types of hypothesis tests that can be used for multiple linear regression:

- **F-test:** This test measures the overall significance of all the coefficients.
- **T-test:** This test measures the significance of each individual coefficient.

F-test

The F-test is used to assess all the coefficients collectively. It validates whether any of the independent variables are significant.

The regression equation for the car price prediction model can be given as

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots + \beta_{59} x_{59} + \epsilon$$

where,

- x_1 is `symboling`
- x_2 is `doornumber`

- x_3 is `wheelbase`

$\vdots$

- x_{59} is `wheelbase` and
- Y is the `price`

Step 1: Define null and alternative hypothesis

$H_0: \beta_1 = \beta_2 = \dots = \beta_{59} = 0$ i.e. all the regression coefficients are equal to zero.

$H_1: \beta_i \neq 0$, i.e. at least one of the coefficient is not zero.

- H_0 means that none of the feature or independent variables have a significant relationship with our target variable `price` and our model has no predictive capability.
- H_1 means that at least one feature variable has a significant relationship with our target variable `price`.

Step 2: Calculate the test statistic value (in case of F-test it is F-statistic value)

It is calculated as

$$F* = \frac{\textrm{explained variance}}{\textrm{unexplained variance}} = \frac{\text{MSM}}{\text{MSE}}$$

where,

- MSM is the Mean of Squares for Model
- MSE is Mean of Squared Errors (or Residuals)

Further, MSM is calculated as

$$\text{MSM} = \frac{\text{SSM}}{\text{DFM}}=\frac{\sum(y_{\text{pred}} - \bar{y})^2}{ p - 1}$$

where,

- SSM is the Sum of Squares for Model
- DFM is Degrees of Freedom for Model
- p is the number of independent variables

Similarly, MSE is calculated as:

$$\text{MSE} = \frac{\text{SSE}}{\text{DFE}}=\frac{\sum(y - y_{\text{pred}})^2}{ N - p}$$

where,

- SSE is the Sum of Squares for Errors
- DFE is Degrees of Freedom for Errors
- $N$$N$ is number of instances (or rows) in the dataset

Let's create `mean_sq_model()` and `mean_sq_error()` functions to calculate the MSM and MSE values using the above formulae respectively.

Note: You can also obtain the MSM and MSE values using the `mse_model` and `mse_resid` attributes respectively of `statsmodels.api` module.

```
#calculating f-stat
f_stat=model.mse_model/model.mse_resid
print(f"The F-statistics for the model {model.mse_model/model.mse_resid}")
```

```
The F-statistics for the model 67.50726851182478
```

```
#Calculating p-value
from scipy.stats import norm
p_val = 2*(1-norm.cdf(abs(f_stat)))
print(p_val)
```

```
0.0
```

```
model.f_pvalue
```

```
3.515694157936587e-53
```

Note:If p-value is below 0.05, the null hypothesis will be rejected.

The p-value that we obtained from F-test is equal to 0.00, so we can reject our null hypothesis and conclude that at least one of the independent variable has linear relationship with our target variable `price`. But, what is p-value?

What is meant by p-value?

The p-value is a probability value that helps us to determine whether our hypothesis is correct. The p-value for each feature tests the null hypothesis that there is no correlation between the feature and the target variable. Smaller the p-value, stronger is the evidence that you should reject null hypothesis. A p-value less than 0.05 is statistically significant. It indicates that there is less than 5% probability that the null hypothesis is correct. Therefore, we reject the null hypothesis, and accept the alternative hypothesis. However, a p-value greater than 0.05 indicates weak evidence and we fail to reject the null hypothesis.

The F-test for our model rejected the null hypothesis and concluded that at least one feature variable is significant and our model definitely possess predictive capability. Now, we will perform **t-test** to determine which variables are significant in predicting the price of a car and which are not.

T-test

After concluding from the F-test that at least one feature variable is significant, now we may want to know which variables are significant. For this, we can do a **t-test** to find out which independent variable is making a useful contribution in the prediction of the dependent variable.

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots + \beta_{59} x_{59} + \epsilon$$

where,

- x_1 is `symboling`
- x_2 is `doornumber`
- x_3 is `wheelbase`

$\vdots$

- x_{59} is `wheelbase` and
- Y is the `price`

For example, let us determine whether feature `symboling` is contributing significantly in the prediction of dependent variable `price`. We will follow the same steps as that of F-test.

Step 1: Define the null and alternative hypothesis

$H_0: \beta_1 = 0$ i.e. `symboling` and `price` are not linearly related

$H_1: \beta_1 \neq 0$ i.e. `symboling` and `price` are linearly related

Step 2: Calculate the test statistic value (in case of t-test, it is t-statistic value)

The t-statistic is calculated as:

$$t^* = \frac{\textrm{coefficient - hypothesized value} }{\textrm{standard error of coefficient}}$$

As the hypothesized value is usually 0, $$t^* = \frac{\textrm{coefficient} }{\textrm{standard error of coefficient}}$$

{\textrm{standard error of coefficient}}$$

For our example above, the t-statistic is:

$$t*= \frac{\beta_1 }{SE(\beta_1)}$$$$t*= \frac{\beta_1 }{SE(\beta_1)}$$

The **standard error of coefficient (SE)** is an estimate of the standard deviation of the coefficient, the amount it varies across cases. Its formula is quite complicated.

However, we can obtain standard error for every coefficient by using `bse` attribute of `statsmodels.api` module. The `b` in `bse` stands for the coefficient $\beta$$\beta$ and `se` for standard errors.

```
bse_symboling= model.bse['symboling']
t1= model.params['symboling']/bse_symboling
print(t1)
```

```
-0.766449291655052
```

```
# p-value for t-stat
print(f"p-value for t-stat for symboling {2* (1-norm.cdf(abs(t1)))}")
```

```
p-value for t-stat for symboling 0.4434090131014836
```

Hence symboling and price are not linearly related hence symboling is insignificant.

```
# Check all the p_values
print(model.pvalues)
```

```
const                0.000002
symboling            0.445484
doornumber           0.356501
wheelbase            0.491750
carlength            0.356087
carwidth             0.003989
carheight            0.364582
curbweight           0.016007
cylindernumber       0.244234
enginesize           0.000297
boreratio            0.007926
stroke               0.789116
compressionratio     0.782639
horsepower           0.758939
peakrpm              0.012603
citympg              0.456021
highwaympg           0.755790
fueltype_gas         0.014296
aspiration_turbo     0.002748
carbody_hardtop      0.889279
carbody_hatchback    0.002640
carbody_sedan        0.005734
carbody_wagon        0.003059
drivewheel_fwd       0.461885
```

```
drivewheel_rwd          0.296761
enginelocation_rear     0.010376
enginetype_dohcv        0.925901
enginetype_l            0.192379
enginetype_ohc          0.725297
enginetype_ohcf         0.285501
enginetype_ohcv         0.288010
enginetype_rotor        0.008174
fuelsystem_2bbl         0.080563
fuelsystem_4bbl         0.663401
fuelsystem_idi          0.237300
fuelsystem_mfi          0.588545
fuelsystem_mpfi         0.293220
fuelsystem_spdi         0.502118
fuelsystem_spfi         0.413231
CarCompany_audi         0.994202
CarCompany_bmw          0.000376
CarCompany_buick        0.002628
CarCompany_chevrolet    0.078490
CarCompany_dodge        0.022523
CarCompany_honda        0.317686
CarCompany_isuzu        0.420871
CarCompany_jaguar       0.561791
CarCompany_mazda        0.104761
CarCompany_mercury      0.523097
CarCompany_mitsubishi   0.011527
CarCompany_nissan       0.043811
CarCompany_peugeot      0.192379
CarCompany_plymouth     0.013786
CarCompany_porsche      0.029307
CarCompany_renault      0.085115
CarCompany_saab         0.677703
CarCompany_subaru       0.016262
```

```python
new_df = pd.DataFrame()
new_df['feature']=X_train.columns
new_df['pValues'] = model.pvalues.values[1:]
new_df=new_df[new_df['pValues']<=0.05]
new_df
```

	feature	pValues
4	carwidth	0.003989
6	curbweight	0.016007
8	enginesize	0.000297
9	boreratio	0.007926
13	peakrpm	0.012603
16	fueltype_gas	0.014296
17	aspiration_turbo	0.002748
19	carbody_hatchback	0.002640
20	carbody_sedan	0.005734
21	carbody_wagon	0.003059
24	enginelocation_rear	0.010376
30	enginetype_rotor	0.008174
39	CarCompany_bmw	0.000376
40	CarCompany_buick	0.002628
42	CarCompany_dodge	0.022523
48	CarCompany_mitsubishi	0.011527
49	CarCompany_nissan	0.043811
51	CarCompany_plymouth	0.013786
52	CarCompany_porsche	0.029307
55	CarCompany_subaru	0.016262

```
#rebuild the Linear regression model
X_new= X[new_df.feature]
X_train2, X_test2, y_train2, y_test2 = train_test_split(X_new, y, test_size

X_train_sm2 = sm.add_constant(X_train2)
model2 = sm.OLS(y_train2, X_train_sm2).fit()
print(model2.summary())
```

```
                                OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.95
Model:                            OLS   Adj. R-squared:                  0.95
Method:                 Least Squares   F-statistic:                     132.
Date:                Sat, 07 Sep 2024   Prob (F-statistic):           3.34e-7
Time:                        03:28:30   Log-Likelihood:                -1206.
No. Observations:                 137   AIC:                             2455
Df Residuals:                     116   BIC:                             2516
Df Model:                          20
Covariance Type:            nonrobust
```

```
==============================================================================
                         coef    std err          t      P>|t|      [0.025
------------------------------------------------------------------------------
const              -5.285e+04   8791.716     -6.012      0.000   -7.03e+04
carwidth             737.6042    146.262      5.043      0.000     447.913
curbweight             4.3546      0.931      4.678      0.000       2.511
enginesize            75.7302      9.699      7.808      0.000      56.519
boreratio          -3150.6456    881.598     -3.574      0.001   -4896.761
peakrpm                1.4756      0.445      3.319      0.001       0.595
fueltype_gas        1677.9896    756.809      2.217      0.029     179.035
aspiration_turbo    2363.7059    526.933      4.486      0.000    1320.049
carbody_hatchback  -3445.8579    829.557     -4.154      0.000   -5088.900
carbody_sedan      -2551.4401    806.653     -3.163      0.002   -4149.118
carbody_wagon      -4302.0295    944.400     -4.555      0.000   -6172.532
enginelocation_rear 6084.9218   2050.365      2.968      0.004    2023.916
enginetype_rotor    6127.0878   1403.152      4.367      0.000    3347.969
CarCompany_bmw      7540.5414    845.205      8.922      0.000    5866.507
CarCompany_buick    6300.5518   1136.994      5.541      0.000    4048.592
CarCompany_dodge    -922.0000    810.515     -1.138      0.258   -2527.327
CarCompany_mitsubishi -2326.3139  687.578    -3.383      0.001   -3688.148
CarCompany_nissan  -1704.4872    647.541     -2.632      0.010   -2987.024
CarCompany_plymouth -1600.7490   791.994     -2.021      0.046   -3169.394
CarCompany_porsche  6807.8825   1474.859      4.616      0.000    3886.738
CarCompany_subaru    820.4880    879.024      0.933      0.353    -920.530
==============================================================================
Omnibus:                        1.936   Durbin-Watson:                   2.05
Prob(Omnibus):                  0.380   Jarque-Bera (JB):                1.52
Skew:                          -0.118   Prob(JB):                        0.46
Kurtosis:                       3.459   Cond. No.                     3.41e+0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
[2] The condition number is large, 3.41e+05. This might indicate that there a
strong multicollinearity or other numerical problems.
```

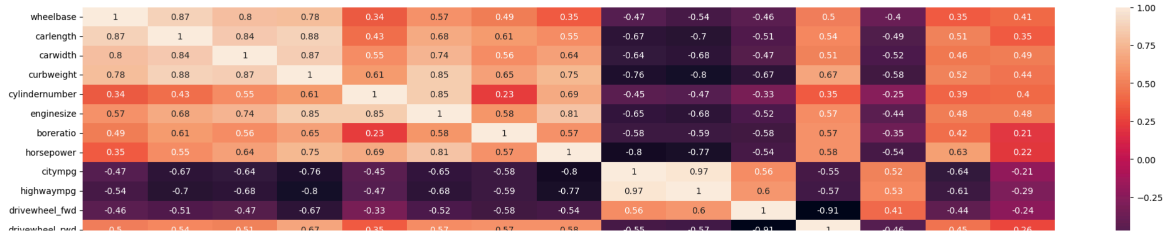
Still we have p_values higher than 0.05 in the result. A better approach to this is using RFE.

```python
# Find the moderately too highly correlated features with price.
corr_df= df_new.corr()['price']
corr_data=corr_df[(corr_df >=0.5)| (corr_df<=-0.5)]
corr_features= list(corr_data.index)
corr_features.remove('price')
corr_features
```

```
['wheelbase',
 'carlength',
 'carwidth',
 'curbweight',
 'cylindernumber',
 'enginesize',
 'boreratio',
 'horsepower',
 'citympg',
```

```
 'highwaympg',
 'drivewheel_fwd',
 'drivewheel_rwd',
 'fuelsystem_2bbl',
 'fuelsystem_mpfi',
 'CarCompany_buick']
```

```
df2= df_new[corr_features]
plt.figure(figsize=(25,6),dpi=100)
sns.heatmap(df2.corr(),annot=True)
plt.show()
```



Recursive Feature Elimination (RFE)

Recursive feature elimination (RFE) is a feature selection (or elimination) method that fits a model and removes the weakest feature (or features). Here, you need to decide the numbers of features you want to select to build a model. Then you can validate your choice of number of features and increase or decrease them (if required). Features are ranked by the model's `coef_` or `feature_importances_` attributes, and by recursively eliminating a small number of features per loop, RFE attempts to eliminate dependencies and collinearity that may exist in a machine learning model.

RFE requires a specified number of features to keep, however it is often not known in advance how many features are valid.

```
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
lr=LinearRegression()
rfe1= RFE(lr,n_features_to_select=10)
```

```
rfe1.fit(X_train[corr_features],y_train)
print(corr_features)
print(rfe1.support_)
print(rfe1.ranking_)
```

```
['wheelbase', 'carlength', 'carwidth', 'curbweight', 'cylindernumber', 'engin
[False False  True False  True  True  True  True  True  True  True False
 False  True  True]
[5 3 1 4 1 1 1 1 1 1 1 6 2 1 1]
```

```
# Look at the features selected.
rfe1_features= X_train[corr_features].columns[rfe1.support_]
rfe1_features
```

```
Index(['carwidth', 'cylindernumber', 'enginesize', 'boreratio',
'horsepower',
       'citympg', 'highwaympg', 'drivewheel_fwd', 'fuelsystem_mpfi',
       'CarCompany_buick'],
      dtype='object')
```

```
# Check for multicollinearity.
X_train_rfe1= X_train[rfe1_features]
X_train_rfe1_sm = sm.add_constant(X_train_rfe1)
lr = sm.OLS(y_train, X_train_rfe1_sm).fit()
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.87
Model:                            OLS   Adj. R-squared:                  0.86
Method:                 Least Squares   F-statistic:                     91.0
Date:                Sat, 07 Sep 2024   Prob (F-statistic):           3.16e-5
Time:                        03:28:39   Log-Likelihood:                -1336.
No. Observations:                 143   AIC:                             2696
Df Residuals:                     132   BIC:                             2728
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                       coef    std err          t      P>|t|      [0.025
------------------------------------------------------------------------------
const             1.412e+04    618.029     22.840      0.000    1.29e+04    1
carwidth          1730.4940    382.530      4.524      0.000     973.813    2
cylindernumber    -697.8044    630.166     -1.107      0.270   -1944.336
enginesize        3477.6658    769.667      4.518      0.000    1955.187    5
boreratio         -700.5804    411.343     -1.703      0.091   -1514.258
horsepower        2205.4249    595.424      3.704      0.000    1027.618    3
citympg           -683.5848   1078.676     -0.634      0.527   -2817.313    1
highwaympg         805.3504   1017.088      0.792      0.430   -1206.551    2
drivewheel_fwd   -2362.0657    716.205     -3.298      0.001   -3778.789    -
fuelsystem_mpfi    870.1314    695.994      1.250      0.213    -506.613    2
CarCompany_buick  8279.4418   1825.139      4.536      0.000    4669.136    1
==============================================================================
Omnibus:                       17.987   Durbin-Watson:                   2.10
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               29.25
Skew:                           0.625   Prob(JB):                     4.45e-0
Kurtosis:                       4.829   Cond. No.                         17.
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
```

```
vif=pd.DataFrame()
vif['Feature']= X_train_rfe1_sm.columns
vif['VIF']=[variance_inflation_factor(X_train_rfe1_sm.values,i) for i in ran
vif
```

	Feature	VIF
0	const	6.532107
1	carwidth	2.502449
2	cylindernumber	6.791186
3	enginesize	10.130751
4	boreratio	2.893638
5	horsepower	6.062996
6	citympg	19.898364
7	highwaympg	17.691014
8	drivewheel_fwd	2.136322
9	fuelsystem_mpfi	2.070927
10	CarCompany_buick	1.548924

So 5 features have a higher p-value out of 10 selected features, so we need to remove 5 features. Rebuild the RFE model to select 5 features this time.

```
lr2=LinearRegression()
rfe2= RFE(lr2,n_features_to_select=5)
rfe2.fit(X_train[corr_features],y_train)
print(corr_features)
print(rfe2.support_)
print(rfe2.ranking_)
```

```
['wheelbase', 'carlength', 'carwidth', 'curbweight', 'cylindernumber', 'engin
[False False  True False False  True False  True False False  True False
 False False  True]
[10  8  1  9  3  1  4  1  6  5  1 11  7  2  1]
```

```
rfe2_features= X_train[corr_features].columns[rfe2.support_]
rfe2_features
```

```
Index(['carwidth', 'enginesize', 'horsepower', 'drivewheel_fwd',
       'CarCompany_buick'],
      dtype='object')
```