Linear Regression :

In [447…

```
#importing the libraries and csv file

import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
import math

file_automob=pd.read_csv('./Automobile_data.csv')
```

In [448…

```
#displaying the top 5 data
file_automob.head(5)
```

Out[448]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136

5 rows × 26 columns

In [449…

```
file_automob.describe()
```

Out[449]:

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	2
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	10.142537	
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	3.972040	
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	7.000000	
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	8.600000	
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	9.000000	
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	9.400000	
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	23.000000	

In [450…

```
#check for duplicate
file_automob.index[file_automob.duplicated()]
```

Out[450]:

```
Int64Index([], dtype='int64')
```

In [451…

```
#checking for type of data in columns
file_automob.dtypes
```

Out[451]:
```
symboling             int64
normalized-losses    object
make                 object
fuel-type            object
aspiration           object
num-of-doors         object
body-style           object
drive-wheels         object
engine-location      object
wheel-base          float64
length              float64
width               float64
height              float64
curb-weight           int64
engine-type          object
num-of-cylinders     object
engine-size           int64
fuel-system          object
bore                 object
stroke               object
compression-ratio   float64
horsepower           object
peak-rpm             object
city-mpg              int64
highway-mpg           int64
price                object
dtype: object
```

In [452…
```
#Conversion of some columns’ datatype from object type to numeric value (numeric value r
file_automob['normalized-losses'] = pd.to_numeric(file_automob['normalized-losses'], err

file_automob['wheel-base'] = pd.to_numeric(file_automob['wheel-base'], errors='coerce')

file_automob['bore'] = pd.to_numeric(file_automob['bore'], errors='coerce')

file_automob['stroke'] = pd.to_numeric(file_automob['stroke'], errors='coerce')

file_automob['horsepower'] = pd.to_numeric(file_automob['horsepower'], errors='coerce')

file_automob['peak-rpm'] = pd.to_numeric(file_automob['peak-rpm'], errors='coerce')

file_automob['price'] = pd.to_numeric(file_automob['price'], errors='coerce')
```

In [453…
```
#checking for columns with NaN
file_automob.isna().sum()
```

Out[453]:
```
symboling            0
normalized-losses   41
make                 0
fuel-type            0
aspiration           0
num-of-doors         0
body-style           0
drive-wheels         0
engine-location      0
wheel-base           0
length               0
width                0
height               0
curb-weight          0
engine-type          0
num-of-cylinders     0
engine-size          0
fuel-system          0
bore                 4
stroke               4
compression-ratio    0
horsepower           2
peak-rpm             2
city-mpg             0
highway-mpg          0
price                4
dtype: int64
```

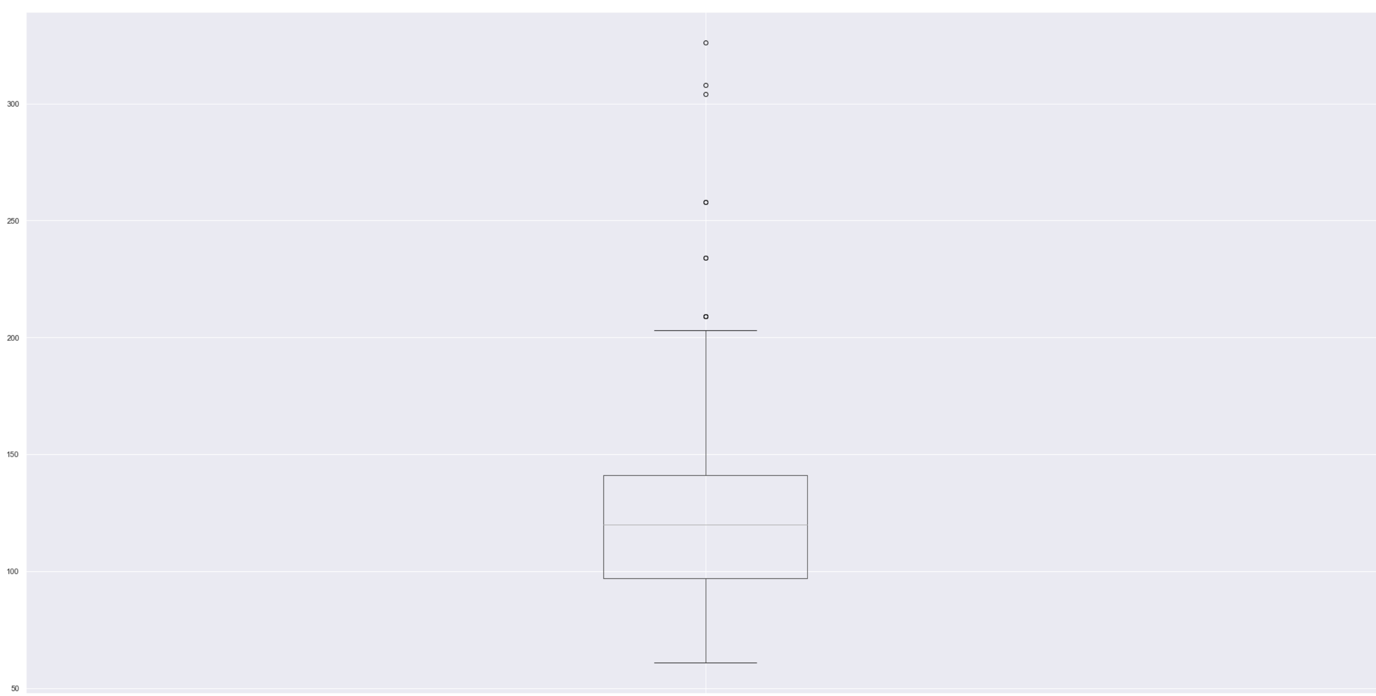
In [454…
```
# fill -1 to all ,mean value
numeric_columns = file_automob.select_dtypes(include=['number']).columns
file_automob[numeric_columns] = file_automob[numeric_columns].fillna(file_automob[numeri
```

In [455…
```
file_automob.shape
```

Out[455]: (205, 26)

In [456…
```
# file_automob.boxplot(column='price')
file_automob.boxplot(column='engine-size')
```

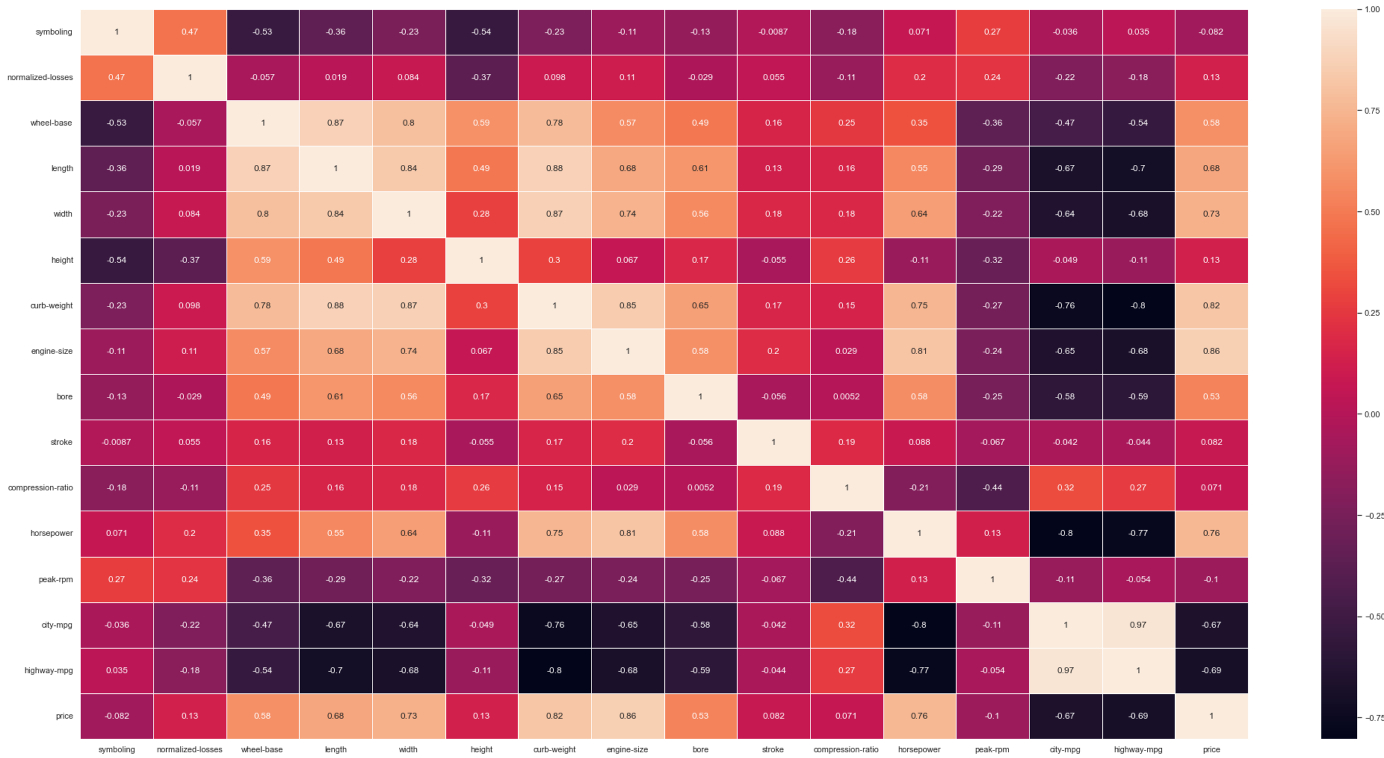
Out[456]: <AxesSubplot:>



Loading [MathJax]/extensions/Safe.js

In [457…

```python
#plotting heatmap to see the correlation
import seaborn as sn
import matplotlib.pyplot as plt

sn.set(rc = {'figure.figsize':(35,18)})
hm = sn.heatmap(data=file_automob.corr(),linewidths=.75,annot=True)
plt.show()
```



In [ ]:

Encoding the data to numeric value

In [458…

```python
#label encoding
file_automob["make"].value_counts()
file_automob["make"] = file_automob["make"].astype('category')
file_automob["make_code"] = file_automob["make"].cat.codes
file_automob[["make", "make_code"]].head(10)
```

Out[458]:

	make	make_code
0	alfa-romero	0
1	alfa-romero	0
2	alfa-romero	0
3	audi	1
4	audi	1
5	audi	1
6	audi	1
7	audi	1
8	audi	1
9	audi	1


```python
…oding
```

```
file_automob["fuel-type"].value_counts()
file_automob=pd.get_dummies(file_automob, columns=["fuel-type"])
```

In [460…

```
file_automob["aspiration"].value_counts()
file_automob=pd.get_dummies(file_automob, columns=["aspiration"])
```

In [461…

```
file_automob["num-of-doors"].value_counts()
file_automob["num-of-doors"] = file_automob["num-of-doors"].replace('?', 'four')#replace
#find and replace


file_automob.head()
```

Out[461]:

	symboling	normalized-losses	make	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	width	...	horsepowe
0	3	122.0	alfa-romero	two	convertible	rwd	front	88.6	168.8	64.1	...	111.(
1	3	122.0	alfa-romero	two	convertible	rwd	front	88.6	168.8	64.1	...	111.(
2	1	122.0	alfa-romero	two	hatchback	rwd	front	94.5	171.2	65.5	...	154.(
3	2	164.0	audi	four	sedan	fwd	front	99.8	176.6	66.2	...	102.(
4	2	164.0	audi	four	sedan	4wd	front	99.4	176.6	66.4	...	115.(

5 rows × 29 columns

In [462…

```
file_automob["num-of-cylinders"].value_counts()
file_automob=pd.get_dummies(file_automob, columns=["num-of-cylinders"])
```

In [463…

```
#label encoding
file_automob["body-style"].value_counts()
file_automob["body-style"] = file_automob["body-style"].astype('category')
file_automob["body-style-code"] = file_automob["body-style"].cat.codes
```

In [464…

```
#one-hot encoding
file_automob["drive-wheels"].value_counts()
file_automob=pd.get_dummies(file_automob, columns=["drive-wheels"])
```

In [465…

```
file_automob["engine-type"].value_counts()
file_automob=pd.get_dummies(file_automob, columns=["engine-type"])
```

In [466…

```
file_automob["engine-location"].value_counts()
file_automob=pd.get_dummies(file_automob, columns=["engine-location"])
```

In [467…

```
file_automob["fuel-system"].value_counts()
file_automob=pd.get_dummies(file_automob, columns=["fuel-system"])
```

In [468…

```
file_automob_back_up=file_automob
file_automob.drop("make", axis=1, inplace=True)
file_automob.drop("body-style", axis=1, inplace=True)
file_automob
```

Out[468]:

	symboling	normalized-losses	num-of-doors	wheel-base	length	width	height	curb-weight	engine-size	bore	...	engine-location_front
0	3	122.0	two	88.6	168.8	64.1	48.8	2548	130	3.47	...	1
1	3	122.0	two	88.6	168.8	64.1	48.8	2548	130	3.47	...	1
2	1	122.0	two	94.5	171.2	65.5	52.4	2823	152	2.68	...	1
3	2	164.0	four	99.8	176.6	66.2	54.3	2337	109	3.19	...	1
4	2	164.0	four	99.4	176.6	66.4	54.3	2824	136	3.19	...	1
...	...	...	...	...	...	...	...	...	...	...	...	...
200	-1	95.0	four	109.1	188.8	68.9	55.5	2952	141	3.78	...	1
201	-1	95.0	four	109.1	188.8	68.8	55.5	3049	141	3.78	...	1
202	-1	95.0	four	109.1	188.8	68.9	55.5	3012	173	3.58	...	1
203	-1	95.0	four	109.1	188.8	68.9	55.5	3217	145	3.01	...	1
204	-1	95.0	four	109.1	188.8	68.9	55.5	3062	141	3.78	...	1

205 rows × 50 columns

In [469…

```
file_automob
```

Out[469]:

	symboling	normalized-losses	num-of-doors	wheel-base	length	width	height	curb-weight	engine-size	bore	...	engine-location_front
0	3	122.0	two	88.6	168.8	64.1	48.8	2548	130	3.47	...	1
1	3	122.0	two	88.6	168.8	64.1	48.8	2548	130	3.47	...	1
2	1	122.0	two	94.5	171.2	65.5	52.4	2823	152	2.68	...	1
3	2	164.0	four	99.8	176.6	66.2	54.3	2337	109	3.19	...	1
4	2	164.0	four	99.4	176.6	66.4	54.3	2824	136	3.19	...	1
...	...	...	...	...	...	...	...	...	...	...	...	...
200	-1	95.0	four	109.1	188.8	68.9	55.5	2952	141	3.78	...	1
201	-1	95.0	four	109.1	188.8	68.8	55.5	3049	141	3.78	...	1
202	-1	95.0	four	109.1	188.8	68.9	55.5	3012	173	3.58	...	1
203	-1	95.0	four	109.1	188.8	68.9	55.5	3217	145	3.01	...	1
204	-1	95.0	four	109.1	188.8	68.9	55.5	3062	141	3.78	...	1

205 rows × 50 columns

In [470…

```
file_automob["num-of-doors"].value_counts()
set_nums = {"num-of-doors":  {"four": 4,  "two": 2}}
file_automob = file_automob.replace(set_nums)
```

In [471…

```
np.corrcoef(file_automob['price'],file_automob['engine-size'])
file_automob
```

Out[471]:

	symboling	normalized-losses	num-of-doors	wheel-base	length	width	height	curb-weight	engine-size	bore	...	engine-location_front
0	3	122.0	2	88.6	168.8	64.1	48.8	2548	130	3.47	...	1
1	3	122.0	2	88.6	168.8	64.1	48.8	2548	130	3.47	...	1
2	1	122.0	2	94.5	171.2	65.5	52.4	2823	152	2.68	...	1
3	2	164.0	4	99.8	176.6	66.2	54.3	2337	109	3.19	...	1
4	2	164.0	4	99.4	176.6	66.4	54.3	2824	136	3.19	...	1
...	...	...	...	...	...	...	...	...	...	...	...	...
200	-1	95.0	4	109.1	188.8	68.9	55.5	2952	141	3.78	...	1
201	-1	95.0	4	109.1	188.8	68.8	55.5	3049	141	3.78	...	1
202	-1	95.0	4	109.1	188.8	68.9	55.5	3012	173	3.58	...	1
203	-1	95.0	4	109.1	188.8	68.9	55.5	3217	145	3.01	...	1
204	-1	95.0	4	109.1	188.8	68.9	55.5	3062	141	3.78	...	1

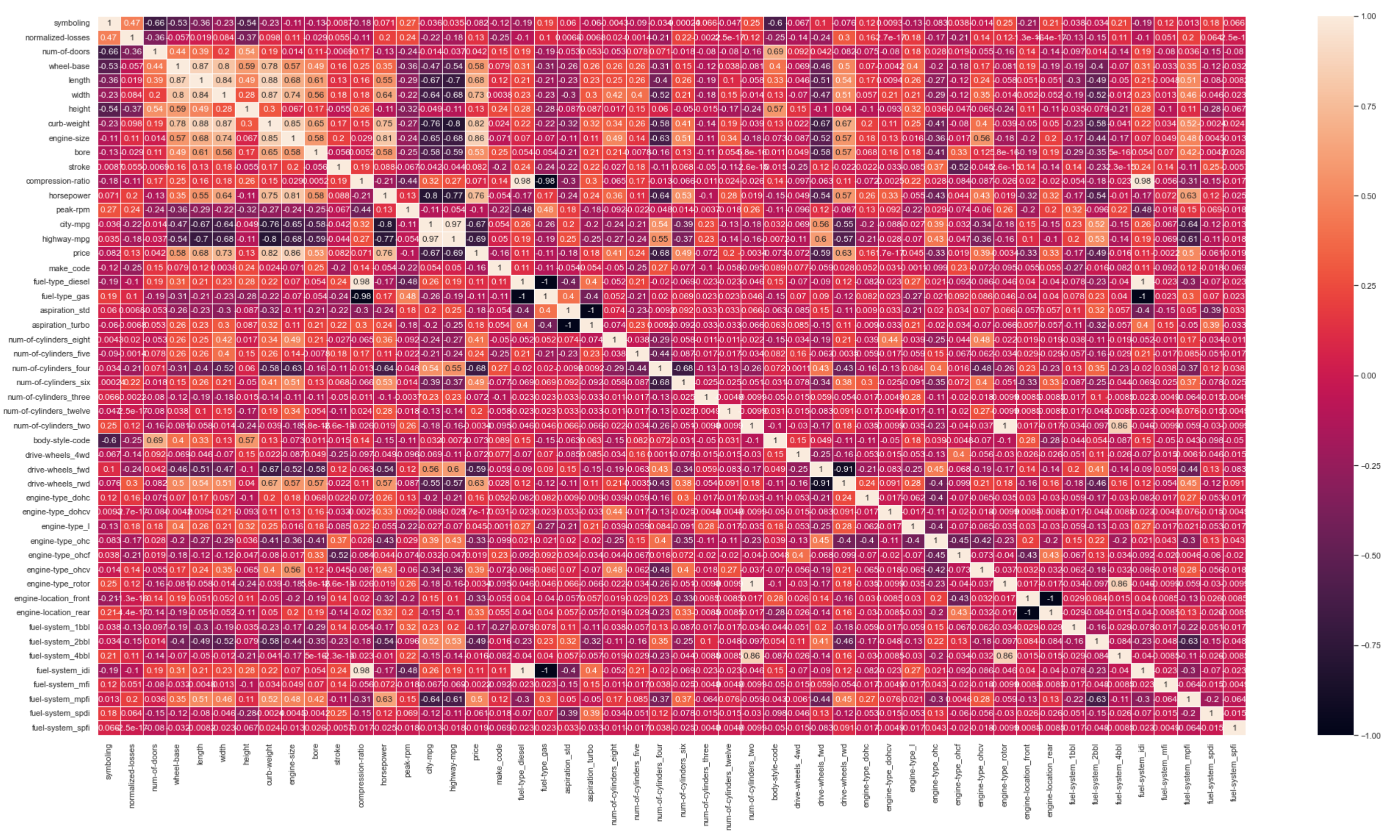
205 rows × 50 columns

In [472…

```
#min max normalization is used when features are of different scales.

for i,col in enumerate(file_automob,1):
    file_automob[col]=(file_automob[col]-file_automob[col].min())/(file_automob[col].max
```

In [473…

```
sn.set(rc = {'figure.figsize':(35,18)})
hm = sn.heatmap(data=file_automob.corr(),linewidths=.75,annot=True)
plt.show()
```



Creation of training and testing data for univariate

In [474…
```
#split the data into 80% training and 20% testing

file_automob_data = file_automob.sample(frac=1,random_state=42)

# Define a size for your train set
train_size = int(0.80 * len(file_automob))

# Split your dataset
train_set = file_automob_data[:train_size]
test_set = file_automob_data[train_size:]
```

In [475…
```
train_set
```

Out[475]:

	symboling	normalized-losses	num-of-doors	wheel-base	length	width	height	curb-weight	engine-size	bore	..
15	0.4	0.298429	1.0	0.492711	0.714925	0.550000	0.658333	0.675718	0.558491	0.771429	..
9	0.4	0.298429	0.0	0.376093	0.553731	0.633333	0.350000	0.607060	0.264151	0.421429	..
100	0.4	0.214660	1.0	0.309038	0.482090	0.408333	0.575000	0.315749	0.222642	0.564286	..
132	1.0	0.445026	0.0	0.364431	0.679104	0.516667	0.691667	0.453840	0.226415	0.714286	..
68	0.2	0.146597	1.0	0.682216	0.743284	0.833333	0.908333	0.877424	0.460377	0.742857	..
...	...	...	...	...	...	...	...	...	...	...	..
59	0.6	0.335079	0.0	0.355685	0.547761	0.516667	0.491667	0.347944	0.230189	0.607143	..
176	0.2	0.000000	1.0	0.460641	0.514925	0.516667	0.591667	0.359193	0.230189	0.550000	..
131	0.8	0.298429	0.0	0.276968	0.532836	0.525000	0.225000	0.377036	0.267925	0.657143	..
17	0.4	0.298429	1.0	0.682216	0.834328	0.883333	0.708333	0.782389	0.558491	0.771429	..
72	1.0	0.403141	0.0	0.291545	0.585075	0.850000	0.250000	0.852211	0.652830	0.657143	..

164 rows × 50 columns

In [476…
```
test_set
```

Out[476]:

	symboling	normalized-losses	num-of-doors	wheel-base	length	width	height	curb-weight	engine-size	bore	..
180	0.2	0.130890	1.0	0.521866	0.697015	0.516667	0.525000	0.637316	0.415094	0.521429	..
134	1.0	0.445026	0.0	0.364431	0.679104	0.516667	0.691667	0.472847	0.226415	0.000000	..
171	0.8	0.361257	0.0	0.344023	0.523881	0.441667	0.350000	0.475562	0.320755	0.771429	..
198	0.0	0.198953	1.0	0.516035	0.711940	0.575000	0.700000	0.603957	0.260377	0.771429	..
63	0.4	0.298429	1.0	0.355685	0.547761	0.516667	0.641667	0.370442	0.230189	0.607143	..
54	0.6	0.251309	1.0	0.189504	0.383582	0.325000	0.525000	0.179209	0.113208	0.385714	..
107	0.4	0.502618	1.0	0.620991	0.680597	0.675000	0.741667	0.594259	0.222642	0.657143	..
50	0.6	0.204188	0.0	0.189504	0.268657	0.325000	0.525000	0.155935	0.113208	0.350000	..
201	0.2	0.157068	1.0	0.655977	0.711940	0.708333	0.641667	0.605508	0.301887	0.885714	..
169	0.8	0.361257	0.0	0.344023	0.523881	0.441667	0.350000	0.412335	0.320755	0.771429	..
58	1.0	0.445026	0.0	0.253644	0.416418	0.450000	0.150000	0.392552	0.071698	0.564108	..
48	0.4	0.298429	1.0	0.769679	0.873134	0.775000	0.416667	1.000000	0.743396	0.778571	..
88	0.2	0.376963	1.0	0.282799	0.467164	0.425000	0.316667	0.354926	0.184906	0.450000	..
21	0.6	0.277487	0.0	0.206997	0.241791	0.291667	0.250000	0.150504	0.109434	0.307143	..
57	1.0	0.445026	0.0	0.253644	0.416418	0.450000	0.150000	0.347944	0.033962	0.564108	..
160	0.4	0.136126	1.0	0.265306	0.376119	0.341667	0.433333	0.235066	0.139623	0.464286	..
197	0.2	0.047120	1.0	0.516035	0.711940	0.575000	0.808333	0.602793	0.301887	0.885714	..
129	0.6	0.298429	0.0	0.344023	0.516418	1.000000	0.225000	0.728472	0.535849	1.000000	..
37	0.4	0.214660	0.0	0.288630	0.394030	0.408333	0.458333	0.290147	0.184906	0.435714	..
157	0.4	0.136126	1.0	0.265306	0.376119	0.341667	0.416667	0.240884	0.139623	0.464286	..
193	0.4	0.298429	1.0	0.402332	0.626866	0.550000	0.608333	0.416990	0.181132	0.464286	..
1	1.0	0.298429	0.0	0.058309	0.413433	0.316667	0.083333	0.411171	0.260377	0.664286	..
52	0.6	0.204188	0.0	0.189504	0.268657	0.325000	0.525000	0.161753	0.113208	0.350000	..
149	0.4	0.104712	1.0	0.300292	0.485075	0.425000	0.591667	0.450737	0.177358	0.771429	..
130	0.4	0.298429	1.0	0.276968	0.602985	0.516667	0.616667	0.423196	0.267925	0.657143	..
151	0.6	0.115183	0.0	0.265306	0.262687	0.275000	0.558333	0.214119	0.116981	0.364286	..
103	0.4	0.225131	1.0	0.402332	0.649254	0.516667	0.608333	0.609775	0.452830	0.635714	..
99	0.4	0.214660	1.0	0.309038	0.482090	0.408333	0.575000	0.324282	0.222642	0.564286	..
116	0.4	0.502618	1.0	0.620991	0.680597	0.675000	0.741667	0.684251	0.343396	0.828571	..
87	0.6	0.314136	1.0	0.282799	0.467164	0.425000	0.316667	0.354926	0.184906	0.450000	..
74	0.6	0.298429	0.0	0.740525	0.867164	0.975000	0.633333	0.863848	0.916981	0.900000	..
121	0.6	0.465969	1.0	0.206997	0.391045	0.291667	0.250000	0.194337	0.109434	0.307143	..
204	0.2	0.157068	1.0	0.655977	0.711940	0.716667	0.641667	0.610551	0.301887	0.885714	..
20	0.4	0.083770	1.0	0.230321	0.264179	0.275000	0.350000	0.163305	0.109434	0.350000	..
188	0.8	0.151832	1.0	0.311953	0.456716	0.433333	0.658333	0.314973	0.181132	0.464286	..
71	0.2	0.298429	1.0	0.845481	0.917910	0.950000	0.725000	0.873545	0.652830	0.657143	..
106	0.6	0.869110	0.0	0.367347	0.558209	0.633333	0.158333	0.640419	0.452830	0.635714	..
14	0.6	0.298429	1.0	0.492711	0.714925	0.550000	0.658333	0.607836	0.388679	0.550000	..

Loading [MathJax]/extensions/Safe.js

	symboling	normalized-losses	num-of-doors	wheel-base	length	width	height	curb-weight	engine-size	bore	..
92	0.6	0.298429	1.0	0.230321	0.361194	0.291667	0.558333	0.174554	0.135849	0.435714	..
179	1.0	0.691099	0.0	0.475219	0.632836	0.616667	0.350000	0.592708	0.415094	0.521429	..
102	0.4	0.225131	1.0	0.402332	0.649254	0.516667	0.691667	0.701319	0.452830	0.635714	..

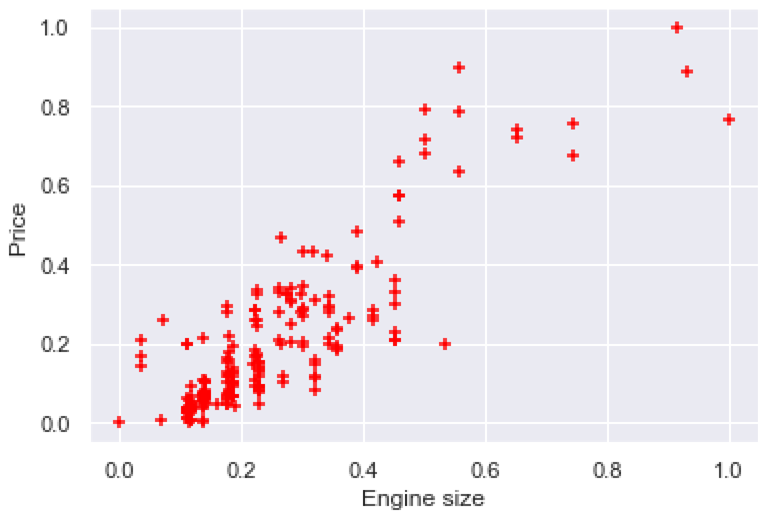
41 rows × 50 columns

In [477…

```
%matplotlib inline
plt.xlabel("Engine size")
plt.ylabel("Price ")
plt.scatter(file_automob["engine-size"],file_automob["price"],color='red',marker='+')
```

Out[477]:

```
<matplotlib.collections.PathCollection at 0x293c13d47c0>
```

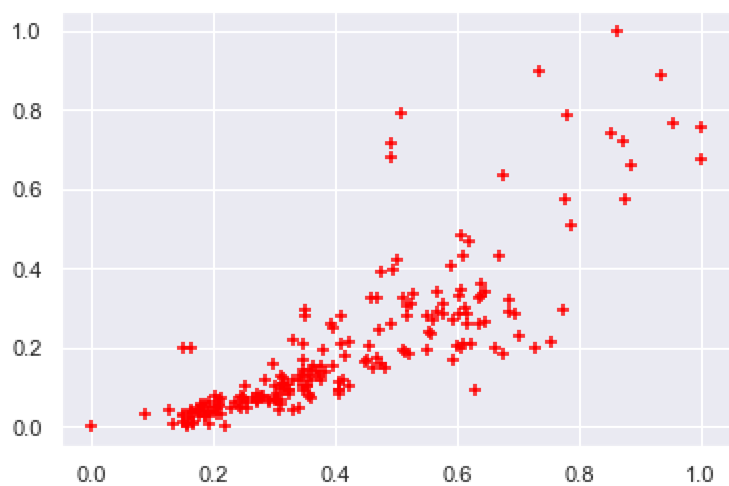


In [478…

```
plt.scatter(file_automob["curb-weight"],file_automob["price"],color='red',marker='+')
```

Out[478]:

```
<matplotlib.collections.PathCollection at 0x293c15e5eb0>
```



Using Inbuilt Linear regression model for comparision

In [479…

```
#inbuilt model
```

Loading [MathJax]/extensions/Safe.js

```
from sklearn import linear_model as lm
model = lm.LinearRegression()
model.fit(np.array(train_set["engine-size"].values).reshape(-1,1), train_set.price.value
```

Out[479]: LinearRegression()

In [480…
```
model.coef_,model.intercept_
```

Out[480]: (array([1.11877542]), -0.07432904428071588)

In [481…
```
import sklearn
import math
predicted_val=[]
for i in test_set["engine-size"]:
    predicted_val.append(model.predict([[i]]))
actual_val=[]
for j in test_set["price"]:
    actual_val.append(j)
mse=sklearn.metrics.mean_squared_error(actual_val,predicted_val)
print('MSE is: ', mse)
rmse=math.sqrt(mse)
print('RMSE is: ', rmse)
```

```
MSE is:  0.012841620378245079
RMSE is:  0.1133208735328363
```

In [482…
```
model.score(np.array(test_set["engine-size"].values).reshape(-1,1), test_set.price.value
#sklearn.metrics.mean_squared_error(test_set)
```

Out[482]: 0.7040297043422985

Univariate Linear regression using Closed form

In [483…
```
#linear regression model for closed form
m=0
c=0

def determine_para(A,B):
    result = [[0],[0]]
    C=np.linalg.inv(A)# find inverse
    for i in range(len(C)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                result[i][j] += C[i][k] * B[k][j]
    return result

def predict_target_value(val):
    return m * val + c

n=len(train_set)
#print(n)
sum_of_x=sum(train_set["engine-size"])
sum_of_y=sum(train_set["price"])
sum_of_xy=sum(train_set["price"]*train_set["engine-size"])
sum_of_x_square=sum(train_set["engine-size"]**2)

A = [[n, sum_of_x],
    [sum_of_x, sum_of_x_square]]

B= [[sum_of_y],
    [sum_of_xy]]
```

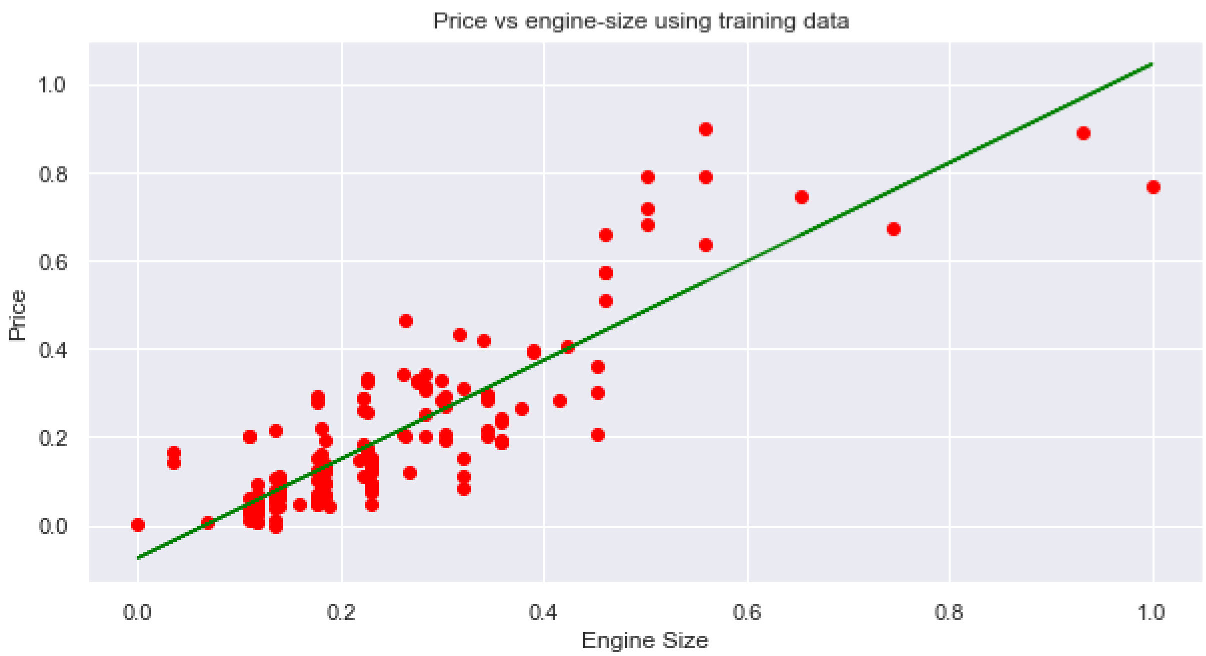
```
parameters=determine_para(A,B)
c=parameters[0][0]
m=parameters[1][0]
m,c
```

Out[483]: (1.1187754249398885, -0.07432904428071707)

In [484…
```
def find_pred_val(x):
    y_pred=[]
    for i in x :
        y_pred.append(predict_target_value(i))
    return y_pred

def rmse_val():
    X_test = test_set["engine-size"]
    Y_test = test_set["price"]
    Y_predicted = find_pred_val(X_test)
    mse = np.square(np.subtract(Y_test,Y_predicted)).mean()
    rmse=math.sqrt(mse)
    return rmse
rmse=rmse_val()
print('MSE is: ', mse)
rmse=math.sqrt(mse)
print('RMSE is: ', rmse)
```

```
MSE is:  0.012841620378245079
RMSE is:  0.1133208735328363
```

In [485…
```
plt.figure(figsize=(10,5))
plt.scatter(train_set["engine-size"], train_set["price"],color = "red")

plt.plot(train_set["engine-size"],predict_target_value(train_set["engine-size"]), color
plt.title("Price vs engine-size using training data ")
plt.ylabel("Price")
plt.xlabel("Engine Size")
plt.show()
```

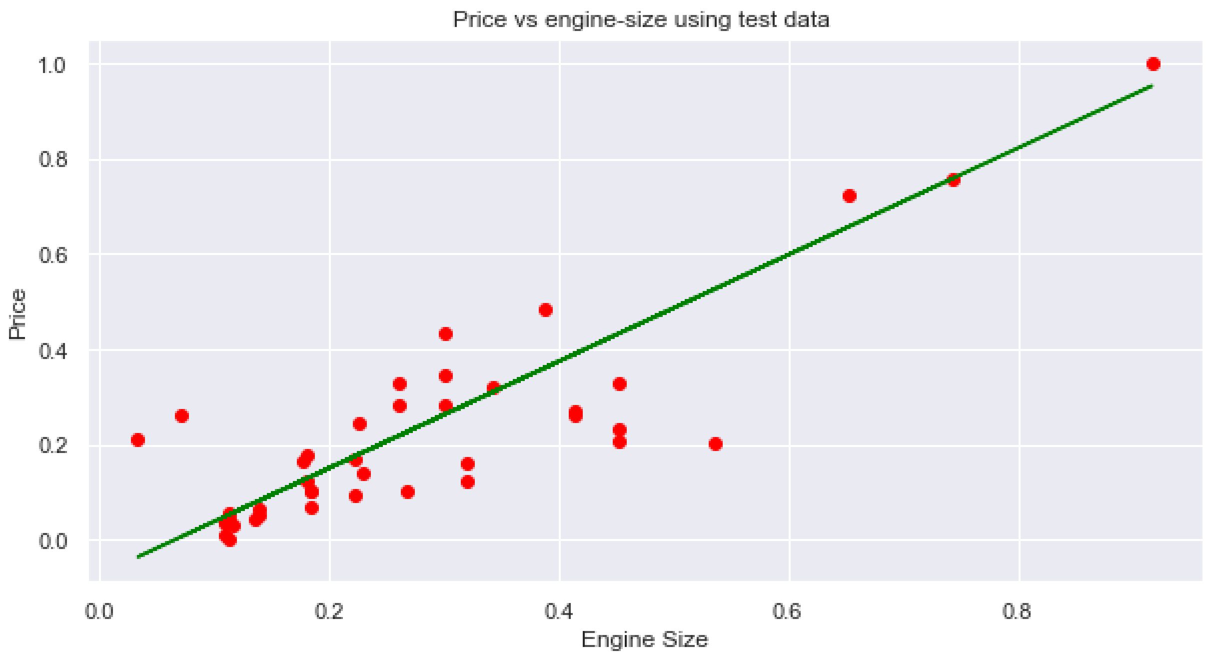


In [486…
```
plt.figure(figsize=(10,5))
plt.scatter(test_set["engine-size"], test_set["price"],color = "red")
plt.plot(test_set["engine-size"],predict_target_value(test_set["engine-size"]), color =
plt.title("Price vs engine-size using test data ")
```

Loading [MathJax]/extensions/Safe.js

```
rice")
```

```
plt.xlabel("Engine Size")
plt.show()
```



Univariate Linear regresssion using gradient descent

In [487…

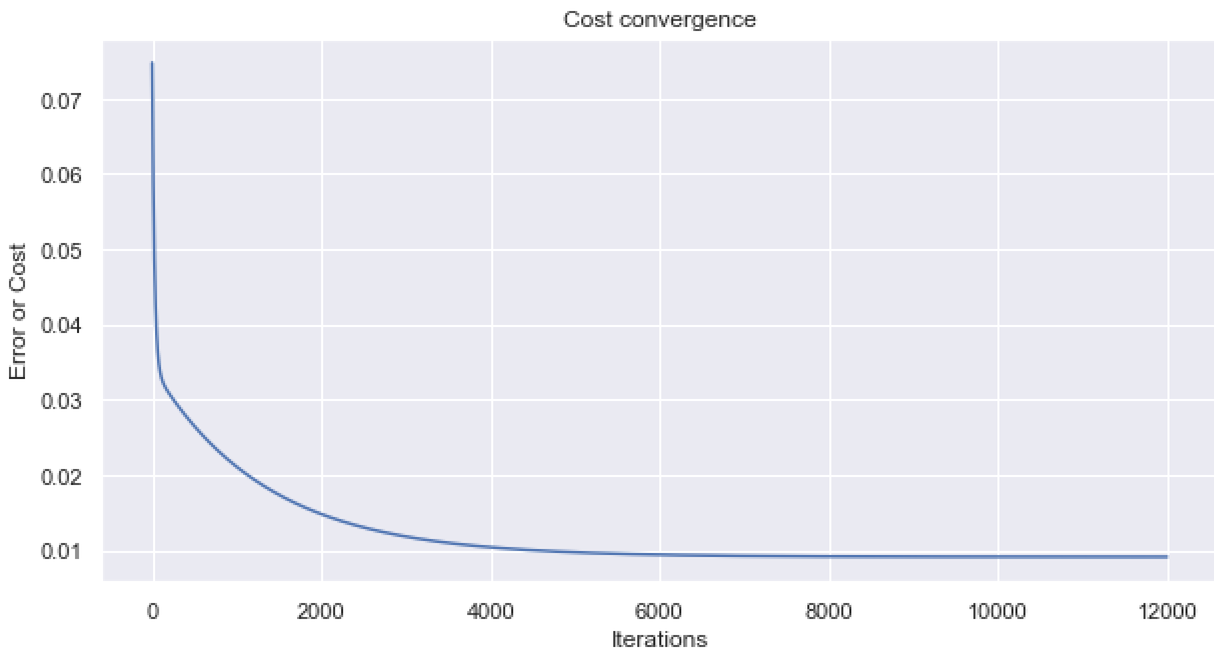
```
w0=0
w1=0
def predict(val):
    return w0 * val + w1

def find_pred(x):
    y_pred=[]
    for i in x :
        y_pred.append(predict(i))
    return y_pred

def rmse_():
    Y_test = test_set["price"]
    X_test = test_set["engine-size"]
    Y_predicted = find_pred(X_test)
    mse = np.square(np.subtract(Y_test,Y_predicted)).mean()
    rmse=math.sqrt(mse)
    print('The Mean Square Error(RMSE) is: ',mse)
    return rmse


def gradient(X,Y):
    m=0
    c=0
    iterations=12000#386600
    a=0.009 #0.000056  #learning rate
    losses=[]
    n=float(len(X))
    for i in range(iterations):#for iterations
        y_pred=m*X+c
        mse=1/n*np.sum((Y-y_pred)**2)
        losses.append(mse)
        part_der_wrt_m=-2/n*(np.sum(X*(Y-y_pred)))
        part_der_wrt_c=-2/n*(np.sum((Y-y_pred)))
```

Loading [MathJax]/extensions/Safe.js

```
            m = m-(a*part_der_wrt_m)
            c = c-(a*part_der_wrt_c)
        plt.figure(figsize=(10,5))
        plt.title("Cost convergence")
        plt.xlabel("Iterations")
        plt.ylabel("Error or Cost")
        plt.plot(losses)
        return m,c


X_train=train_set["engine-size"]
Y_train=train_set["price"]

# plt.scatter(X,Y)
# plt.show()

parameters=gradient(X_train,Y_train)
w0=parameters[0]
w1=parameters[1]
parameters[0],parameters[1]
```

Out[487]:

```
(1.1064240760502142, -0.07128546857536082)
```



In [488…

```
rmse=rmse_()
print('The Root Mean Square Error(RMSE) is: ',rmse)
```

```
The Mean Square Error(RMSE) is:  0.012701828970142576
The Root Mean Square Error(RMSE) is:  0.11270239114651727
```

Multivariate Linear regresssion using inbuilt function

In [489…

```
# importing module
from sklearn.linear_model import LinearRegression
# creating an object of LinearRegression class
LR = LinearRegression()

X = file_automob[[ 'width', 'height','length', 'curb-weight', 'engine-size',
                  'horsepower','normalized-losses','bore','wheel-base','fuel-system_mpfi
                  'drive-wheels_rwd','num-of-cylinders_twelve','num-of-cylinders_six',
                  'num-of-cylinders_eight','engine-type_ohcv']]
Y = file_automob['price']
```

Loading [MathJax]/extensions/Safe.js

```
X_sample=X.sample(frac=1,random_state=11)
Y_sample=Y.sample(frac=1,random_state=11)

train_size = int(0.8 * len(X))

X_train = X_sample[:train_size]
Y_train = Y_sample[:train_size]

X_test = X_sample[train_size:]
Y_test = Y_sample[train_size:]



LR.fit(X_train,Y_train)
```

Out[489]:
```
LinearRegression()
```

In [490…
```
y_prediction =  LR.predict(X_test)
y_prediction
```

Out[490]:
```
array([0.04384636, 0.27215356, 0.0370507 , 0.14580452, 0.1303776 ,
       0.08989466, 0.042458  , 0.08659459, 0.09760683, 0.2938108 ,
       0.24566261, 0.43619594, 0.43320337, 0.16138151, 0.03620116,
       0.36556709, 0.30130425, 0.10913507, 0.88212057, 1.17830258,
       0.10579358, 0.07149404, 0.19241201, 0.371574  , 0.0235186 ,
       0.04610698, 0.15587699, 0.04028869, 0.25347847, 0.01693921,
       0.26053726, 0.7506575 , 0.04593311, 0.03385995, 0.40576928,
       0.13103607, 0.08335017, 0.03895706, 0.11177206, 0.22435964,
       0.07585564])
```

In [491…
```
# MSE
mse = np.square(np.subtract(Y_test,y_prediction)).mean()
rmse=math.sqrt(mse)
print('MSE is: ', mse)
```

```
MSE is:  0.026984294648760725
```

In [492…
```
print('RMSE is: ', rmse)
```

```
RMSE is:  0.16426897043800062
```

Multivariate Linear regresssion using closed form for some features as including all features resulted in non invertible matrix

In [493…
```
file_automob.columns
```

Out[493]:
```
Index(['symboling', 'normalized-losses', 'num-of-doors', 'wheel-base',
       'length', 'width', 'height', 'curb-weight', 'engine-size', 'bore',
       'stroke', 'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price', 'make_code', 'fuel-type_diesel',
       'fuel-type_gas', 'aspiration_std', 'aspiration_turbo',
       'num-of-cylinders_eight', 'num-of-cylinders_five',
       'num-of-cylinders_four', 'num-of-cylinders_six',
       'num-of-cylinders_three', 'num-of-cylinders_twelve',
       'num-of-cylinders_two', 'body-style-code', 'drive-wheels_4wd',
       'drive-wheels_fwd', 'drive-wheels_rwd', 'engine-type_dohc',
       'engine-type_dohcv', 'engine-type_l', 'engine-type_ohc',
       'engine-type_ohcf', 'engine-type_ohcv', 'engine-type_rotor',
       'engine-location_front', 'engine-location_rear', 'fuel-system_1bbl',
       'fuel-system_2bbl', 'fuel-system_4bbl', 'fuel-system_idi',
       'fuel-system_mfi', 'fuel-system_mpfi', 'fuel-system_spdi',
       'fuel-system_spfi'],
      dtype='object')
```

In [494…
```
# X = file_automob.drop('price',axis="columns")
X = file_automob[[ 'width', 'height','length', 'curb-weight', 'engine-size',
                  'horsepower','normalized-losses','bore','wheel-base','fuel-system_mpfi
                  'drive-wheels_rwd','num-of-cylinders_twelve','num-of-cylinders_six',
                  'num-of-cylinders_eight','engine-type_ohcv']]
Y = file_automob['price']


X_sample=X.sample(frac=1,random_state=11)
Y_sample=Y.sample(frac=1,random_state=11)

train_size = int(0.8 * len(X))

X_train = X_sample[:train_size]
Y_train = Y_sample[:train_size]

X_test = X_sample[train_size:]
Y_test = Y_sample[train_size:]
```

In [495…
```
# add x0 =1 to dataset so that to multiply with w0 of the equation
#np.ones: Return a new array of given shape and type, filled with ones.

X_train_new = np.c_[np.ones((X_train.shape[0],1)),X_train]
X_test_new = np.c_[np.ones((X_test.shape[0],1)),X_test]

# find the parameter (W=inv(Xt.X).(Xt.Y))
W = np.matmul(np.linalg.inv(np.matmul(X_train_new.T , X_train_new)) ,np.matmul(X_train_n


#Y~~X.W
Y_pred = np.matmul(X_test_new , W)
Y_pred
```

Out[495]:
```
array([0.04384636, 0.27215356, 0.0370507 , 0.14580452, 0.1303776 ,
       0.08989466, 0.042458  , 0.08659459, 0.09760683, 0.2938108 ,
       0.24566261, 0.43619594, 0.43320337, 0.16138151, 0.03620116,
       0.36556709, 0.30130425, 0.10913507, 0.88212057, 1.17830258,
       0.10579358, 0.07149404, 0.19241201, 0.371574  , 0.0235186 ,
       0.04610698, 0.15587699, 0.04028869, 0.25347847, 0.01693921,
       0.26053726, 0.7506575 , 0.04593311, 0.03385995, 0.40576928,
       0.13103607, 0.08335017, 0.03895706, 0.11177206, 0.22435964,
       0.07585564])
```

In [496…
```
# MSE
```

```
mse = np.square(np.subtract(Y_test,Y_pred)).mean()
rmse=math.sqrt(mse)
print('MSE is: ', mse)
```

```
MSE is:  0.026984294648760485
```

In [497…

```
# RMSE
rmse=math.sqrt(mse)
print('RMSE is: ', rmse)
```

```
RMSE is:  0.1642689704379999
```

Multivariate Linear regresssion using gradient descent

In [498…

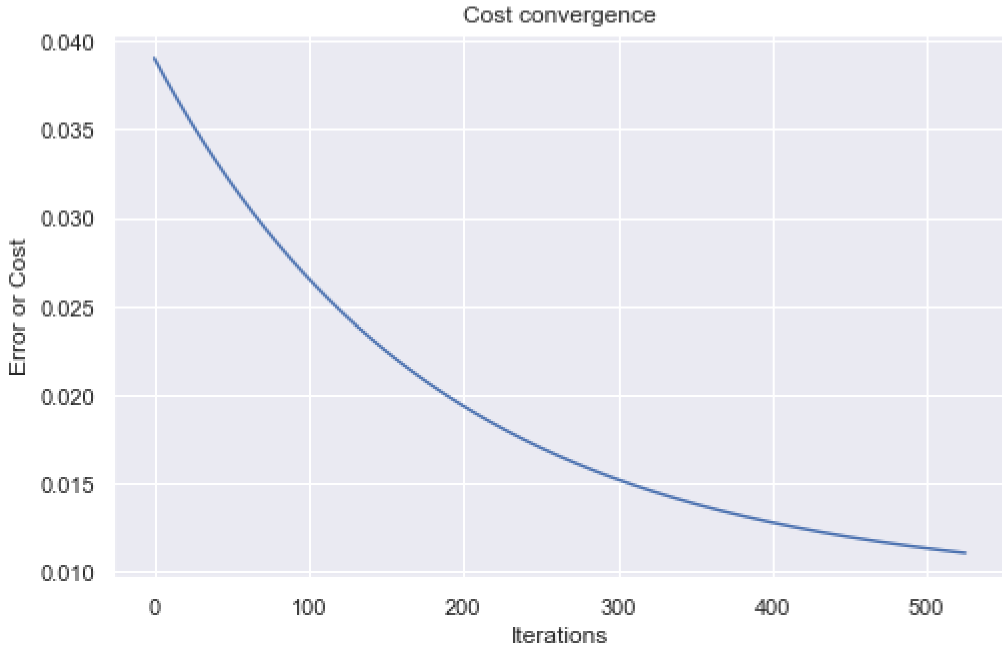
```
a=0.0009
iteration=526
n=len(Y_train)


def cost_fun(X,Y,W):
    #h=(X.wT)-Y
    h=(np.matmul(X,W.T))-Y
    # Equation of cost : J=1/2n*(hT.h)=1/2n*((X.wT)-Y)T.((X.wT)-Y)
    cost = 1/(2*n) * np.matmul(h.T, h)
    return cost

def gradient_descent(X,Y,W):
    losses = np.zeros(iteration)
    for i in range(iteration):
        cost= cost_fun(X, Y, W)
        h =(np.matmul(X,W.T))-Y
        #W=W-a*(1/n)*(XT.((X.wT)-Y))
        W = W - (a * (1/n) * np.matmul(X.T, h))
        losses[i] = cost
    return W, losses


#initialize the parameter list to 0's as number of columns
W = np.zeros(X_train_new.shape[1])

initial_cost=cost_fun(X_train_new,Y_train,W)

W, errors = gradient_descent(X_train_new, Y_train, W)
```

In [499…

```
plt.figure(figsize=(8,5))
plt.title("Cost convergence")
plt.xlabel("Iterations")
plt.ylabel("Error or Cost")
plt.plot(errors)
```

Out[499]:

```
[<matplotlib.lines.Line2D at 0x293bf4bd490>]
```



In [500…

```
#Y~~XW
Y_pred = np.matmul(X_test_new , W)
Y_pred
```

Out[500]:

```
array([0.11359184, 0.24877547, 0.1113323 , 0.17404583, 0.13059704,
       0.14096133, 0.11181719, 0.10536457, 0.12154128, 0.2336385 ,
       0.22360861, 0.24277081, 0.22783729, 0.17876933, 0.10247644,
       0.2585757 , 0.25379041, 0.1195151 , 0.3104399 , 0.2737284 ,
       0.1218028 , 0.09399162, 0.1976006 , 0.2359169 , 0.0982692 ,
       0.1116657 , 0.17597444, 0.11142883, 0.26154042, 0.09740014,
       0.14527938, 0.30013581, 0.1116095 , 0.09987955, 0.24797645,
       0.16781049, 0.12542451, 0.11167359, 0.12319246, 0.18549648,
       0.11970865])
```

In [501…

```
mse = np.square(np.subtract(Y_test,Y_pred)).mean()
print('MSE is: ', mse)
```

```
MSE is:  0.02690475433695938
```

In [502…

```
# RMSE
rmse=math.sqrt(mse)
print('RMSE is: ', rmse)
```

```
RMSE is:  0.164026687880233
```

In [ ]: