Problem Statement

A real estate agent want help to predict the house price for regions in USA.He gave us the dataset to work on to use linear regression model.Create a model that helps him to estimate of what the house would sell for

Import libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# To import dataset
df=pd.read_csv('wine csv')
df
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

In [3]:
```
# To display top 10 rows
df.head(10)
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	5
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	7
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	7
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	5

Data Cleaning and Pre-Processing

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [5]:
```
df.describe()
```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	der
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.00(
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.99(
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003

In [6]:
```
df.columns
```

Out[6]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

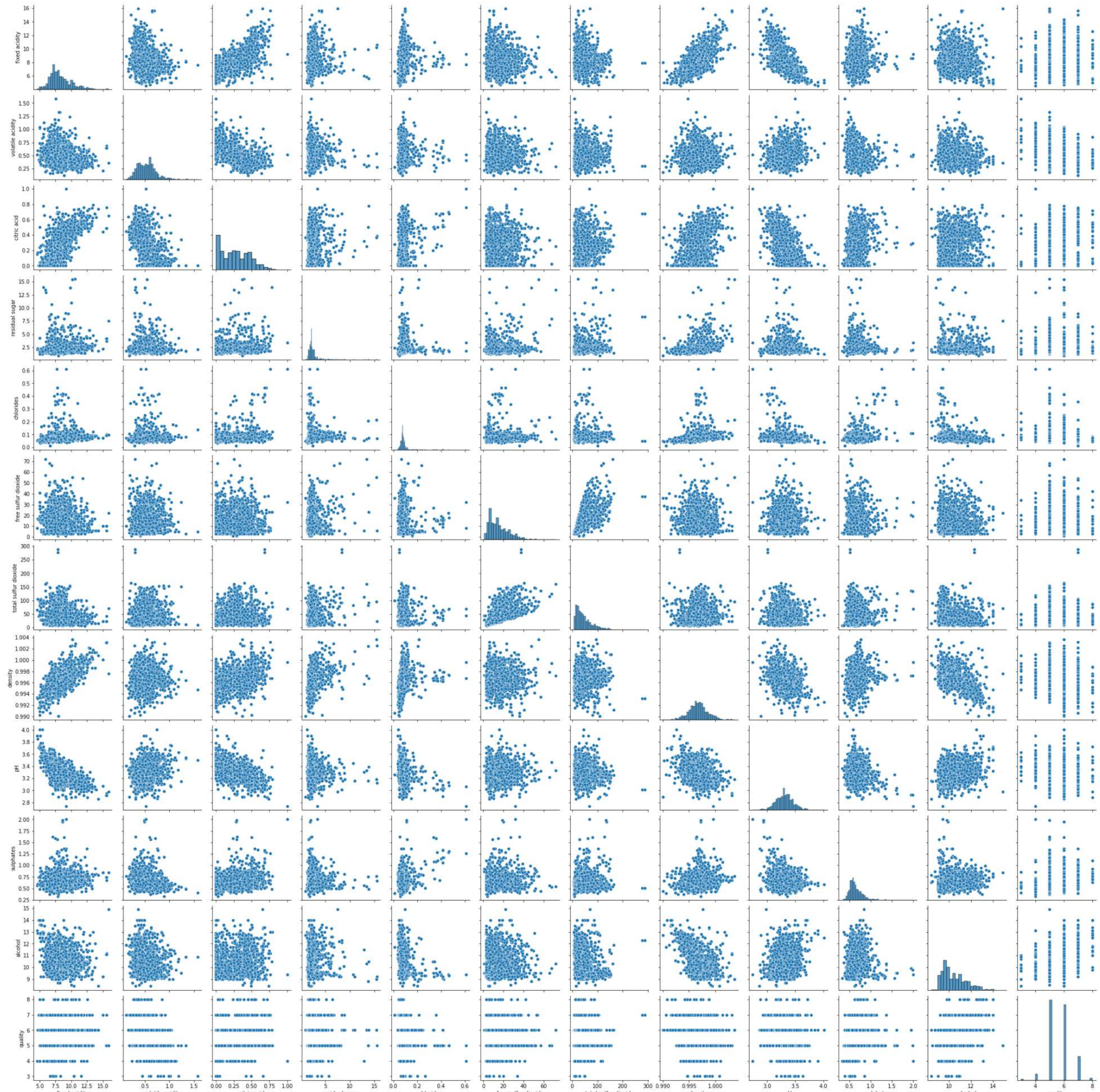
In [7]:
```
a = df.dropna(axis='columns')
a.columns
```

Out[7]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

EDA and Visualization

In [8]:
```
sns.pairplot(a)
```

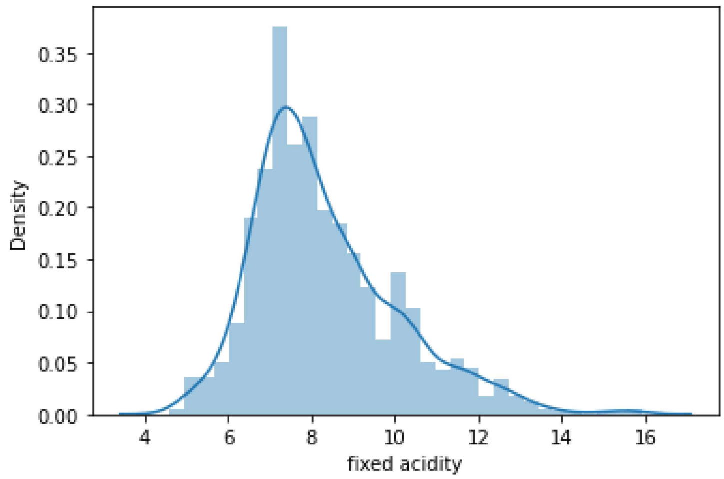
Out[8]: <seaborn.axisgrid.PairGrid at 0x271bd925970>



In [9]:
```
sns.distplot(a['fixed acidity'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Please
adapt your code to use either `displot` (a figure-level function with similar flexibil
ity) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

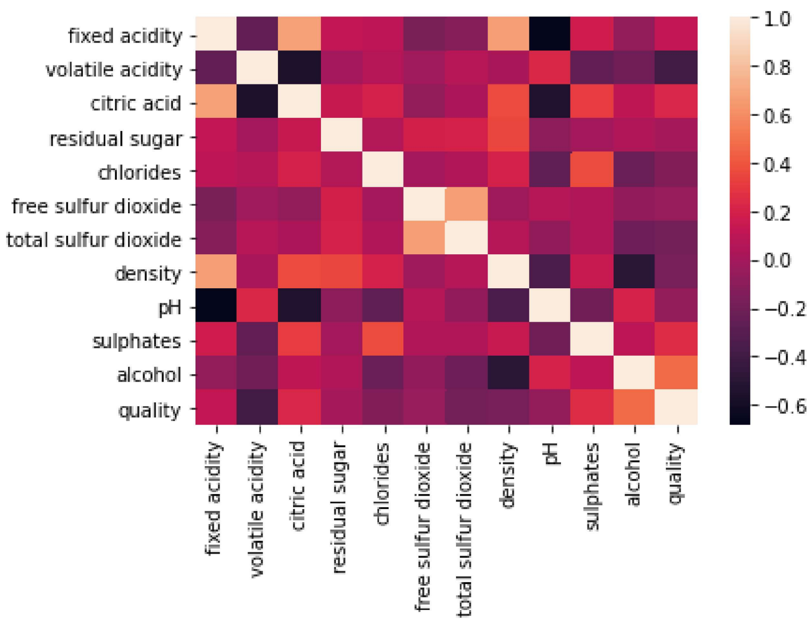
Out[9]:
```
<AxesSubplot:xlabel='fixed acidity', ylabel='Density'>
```



In [10]:
```
a1=a[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality']]
```

In [11]:
```
sns.heatmap(a1.corr())
```

Out[11]:
```
<AxesSubplot:>
```



To Train the Model - Model Building

We are going to train Linear Regression model;We need to split out data into two variables x and y where x

In [12]:
```
x=a1[['volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality']]
y=a1['fixed acidity']
```

To split my dataset into training and test data

In [13]:
```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

In [14]:
```
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[14]:
```
LinearRegression()
```

In [15]:
```
print(lr.intercept_)
```

```
-619.0216119611669
```

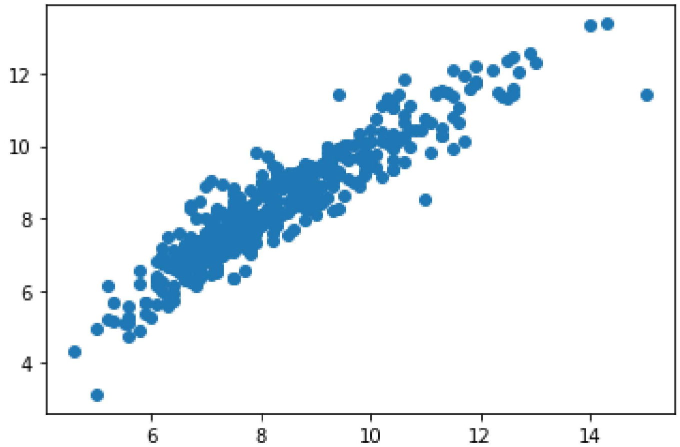
In [16]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[16]:

	Co-efficient
volatile acidity	0.479182
citric acid	2.069099
residual sugar	-0.243553
chlorides	-4.274731
free sulfur dioxide	0.009713
total sulfur dioxide	-0.006133
density	642.160654
pH	-5.375318
sulphates	-0.532472
alcohol	0.544640
quality	0.011271

In [17]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[17]: <matplotlib.collections.PathCollection at 0x271c72040d0>



In [18]:
```
print(lr.score(x_test,y_test))
```

```
0.869487541741365
```

In [19]:
```
from sklearn.linear_model import ElasticNet
en = ElasticNet()
en.fit(x_train,y_train)
```

Out[19]: ElasticNet()

In [20]:
```
print(en.coef_)
```

```
[-0.          0.          0.          0.         -0.01677704 -0.0019159
  0.         -0.          0.         -0.          0.        ]
```

In [21]:
```
print(en.intercept_)
```

```
8.71694776887708
```

In [22]:
```
print(en.predict(x_test))
```

```
[8.36478453 7.90704039 8.32299764 8.18111768 8.29710716 8.34982028
 8.44903272 8.62684876 8.36323161 8.60815581 8.65512122 8.35080401
 8.2338308  8.48067091 8.35463582 8.34645456 8.48455427 8.27173433
 8.57605154 8.35991744 8.42505816 7.8696545  8.09153631 8.54586317
 8.18836676 7.90414076 8.24915803 8.59137877 8.46394542 8.46534368
 8.09914837 8.40833267 8.63068056 8.49651577 8.56072431 8.57988335
 8.5573586  8.60577382 7.98326199 8.52380451 8.42221008 8.57030383
 8.02059633 8.57988335 8.37141286 8.45912988 8.5099271  7.92666551
 7.80736186 8.4356214  8.22854918 8.37379485 8.09153631 8.36235099
 8.24439405 8.18111768 8.6024081  8.50702747 8.12887065 8.55352679
 8.45384826 8.50894337 8.16915617 8.604324   8.43758886 8.49553205
 7.50200937 8.51230909 8.49791404 8.07382709 8.27556614 8.53146812
 8.58563106 8.59712648 8.08050698 8.3944037  8.15621094 8.57843353
 8.59712648 8.56693811 8.43178959 8.29814245 8.61390352 8.27799968
 8.58946286 8.44566701 8.2731326  8.58754696 8.61007171 8.48450271
 8.29135945 8.43515531 8.46006206 8.3512701  8.59666039 8.46389386
 8.01194899 8.43225568 8.15859293 8.58179925 8.11214516 8.13663737
 8.58754696 8.36670043 8.25205766 8.18158377 8.22999899 8.58133316
 8.52623805 8.51230909 8.17065754 8.36468142 8.3574839  7.7046806
 8.17438624 8.35846763 8.49791404 8.34645456 8.15522721 8.52188861
 8.57796744 8.58179925 8.28514566 8.14279961 8.1144756  8.40973093
 8.39393761 8.45291609 8.59329467 7.912685   8.18831521 8.2669188
 8.58563106 8.40015141 8.53866565 8.33112734 8.50702747 8.43515531
 8.04498543 8.56983774 8.6417099  8.5161409  8.18691694 8.47922109
 8.65320532 8.194529   8.33687505 8.37281112 8.57076992 7.91993408
 8.49553205 8.53483384 8.07475927 8.28224603 8.35893371 8.63021448
 8.1926131  8.45431435 8.54058155 7.99180623 8.63451237 8.65320532
 8.58133316 8.40926484 8.60623991 8.55399288 8.10784727 8.44136911
 8.21322195 8.3493542  8.32967752 8.58179925 8.44473483 8.63451237
 7.95110617 8.6455417  8.54777908 8.36618279 8.28607783 8.03840866
 8.19168093 8.57221973 8.49936385 8.50319566 8.37954256 8.63068056
 8.44810055 8.45959597 8.57460173 8.34024077 8.41931045 8.4706253
 8.34267431 8.41118074 8.59329467 8.4687094  7.89078099 8.61198762
 8.12265685 8.10111583 8.485538   8.35991744 8.37524467 8.5611904
 8.40206732 8.54156528 8.40973093 8.30720432 8.6455417  8.37042913
 8.54632926 8.47782283 8.31776757 8.24439405 8.31963192 8.51950662
 8.61581942 8.37954256 8.43608749 8.30244034 8.22098867 7.88684608
 8.58563106 8.3944037  8.6004922  8.46964157 7.92324824 8.58946286
 7.66734626 8.49025042 8.41553019 8.52188861 8.50702747 8.41164683
 8.6455417  7.94012839 8.61198762 8.11929113 8.56647202 8.44711682
 8.28706156 8.50894337 8.47730519 8.56409003 8.50847728 8.44520092
 8.22047103 8.49599813 8.30238879 8.4274917  8.02059633 8.53146812
 8.10784727 8.59329467 8.04700445 8.60623991 8.59282858 8.39486979
 8.36038353 8.32537963 8.54777908 8.41408038 8.50511156 8.60623991
 8.06896    8.25298983 8.40206732 8.04892035 8.48113699 8.31108768
 8.42365989 8.56693811 8.21467177 8.49408223 8.40496695 8.31098458
 8.63259647 8.32729553 8.57460173 8.26412227 8.4356214  8.44810055
 8.50127976 8.62829857 8.18831521 8.64745761 8.63068056 8.57030383
 8.39631961 8.20027671 8.14611378 8.40589912 8.09340066 8.41309665
 8.12410667 8.08811904 8.26743644 8.21042543 8.28752765 8.38440965
 8.34982028 8.3613157  8.58754696 8.1432657  8.50894337 8.38720618
 8.54586317 8.49553205 8.0670441  8.65128941 7.99465431 8.35510191
 8.40786658 8.04560618 8.55689251 8.57843353 8.60815581 8.2506594
 8.58563106 8.50847728 8.57796744 8.43417159 8.05037016 8.06849392
 8.25159157 8.42267617 8.32398137 8.06031266 8.46725958 8.46151187
 8.12943984 8.13177028 8.16724027 8.59521057 8.11597697 8.28276367
 8.55399288 8.26023891 8.59329467 8.42122635 8.54011546 8.54058155
 8.56838793 8.31771601 8.28374739 8.60194201 8.58563106 7.71283607
 8.14140135 8.45623025 8.37286268 8.61007171 8.37042913 8.00138574
 8.39631961 8.59137877 8.57221973 8.5592745  8.35183929 8.46296169
 8.37762666 8.6455417  8.41547864 8.16625654 8.23393391 8.58946286
```

```
 8.45819771 8.11789287 8.27841422 8.18929894 8.49459987 8.37286268
 8.30720432 8.12887065 7.98326199 8.1782696  8.43800339 8.58754696
 8.34598848 8.59329467 8.63787809 8.6024081  8.60815581 7.92371433
 7.85722691 8.2463615  7.33853684 8.63979399 8.61390352 8.59137877
 8.31823366 8.18453495 7.90414076 8.14569925 7.93049732 8.52862004
 8.46244405 8.53866565 8.64937351 8.57605154 8.42221008 8.48641862
 8.04022145 8.55544269 8.43039133 8.64745761 8.26505445 8.55544269
 8.58179925 8.22999899 8.28374739 8.60623991 8.49791404 7.68023994
 8.38969127 8.57030383 8.12084405 8.37234504 8.42267617 8.43608749
 8.54441336 8.4025334  8.36276552 8.48978434 8.06657801 8.55016107
 8.12503884 8.49123415 8.61587098 8.44520092 8.45912988 8.24206361
 8.52717023 8.17350562 8.10681198 8.33547679 8.57651763 8.39823551
 8.46725958 8.23434844 8.59329467 8.54058155 8.29809089 8.40926484
 8.56885402 8.60815581 8.52188861 8.65128941 8.19934454 8.4006175
 8.62684876 8.04405326 8.63787809 8.22093712 7.92526725 8.00138574
 7.99563803 8.32781317 8.62301695 8.60815581 8.53146812 8.25019331
 8.63979399 8.04793662 8.27411632 8.38575636 8.41164683 8.46197796
 8.28752765 8.19789472 8.59712648 8.32827926 8.42842388 8.24630995
 8.61007171 8.38575636 8.50894337 8.61390352 8.39295389 8.57796744]
```

In [23]:

```
print(en.score(x_test,y_test))
```

```
0.01809375188238571
```

Evaluation Metrics

In [25]:

```
from sklearn import metrics
print("Mean Absolytre Error:",metrics.mean_absolute_error(y_test,prediction))
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Mean Absolytre Error: 0.45459459134728825
Mean Squared Error: 0.36848178256419484
Root Mean Squared Error: 0.6070270031589986
```