Problem Statement:

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

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

In [2]:
```
df=pd.read_csv("winequality.csv")
df
```

Out[2]:

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

1599 rows × 12 columns

In [3]:
```
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 [4]:
```
df.head()
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

Data cleaning and Pre-Processing

In [5]:
```
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 [6]:
```
df.describe()
```

Out[6]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfu dioxid
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.00000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.46779
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.89532
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.00000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.00000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.00000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.00000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.00000

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

Out[7]:

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

1599 rows × 12 columns

In [8]:
```
a = df.dropna(axis='columns')
a.columns
```

Out[8]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

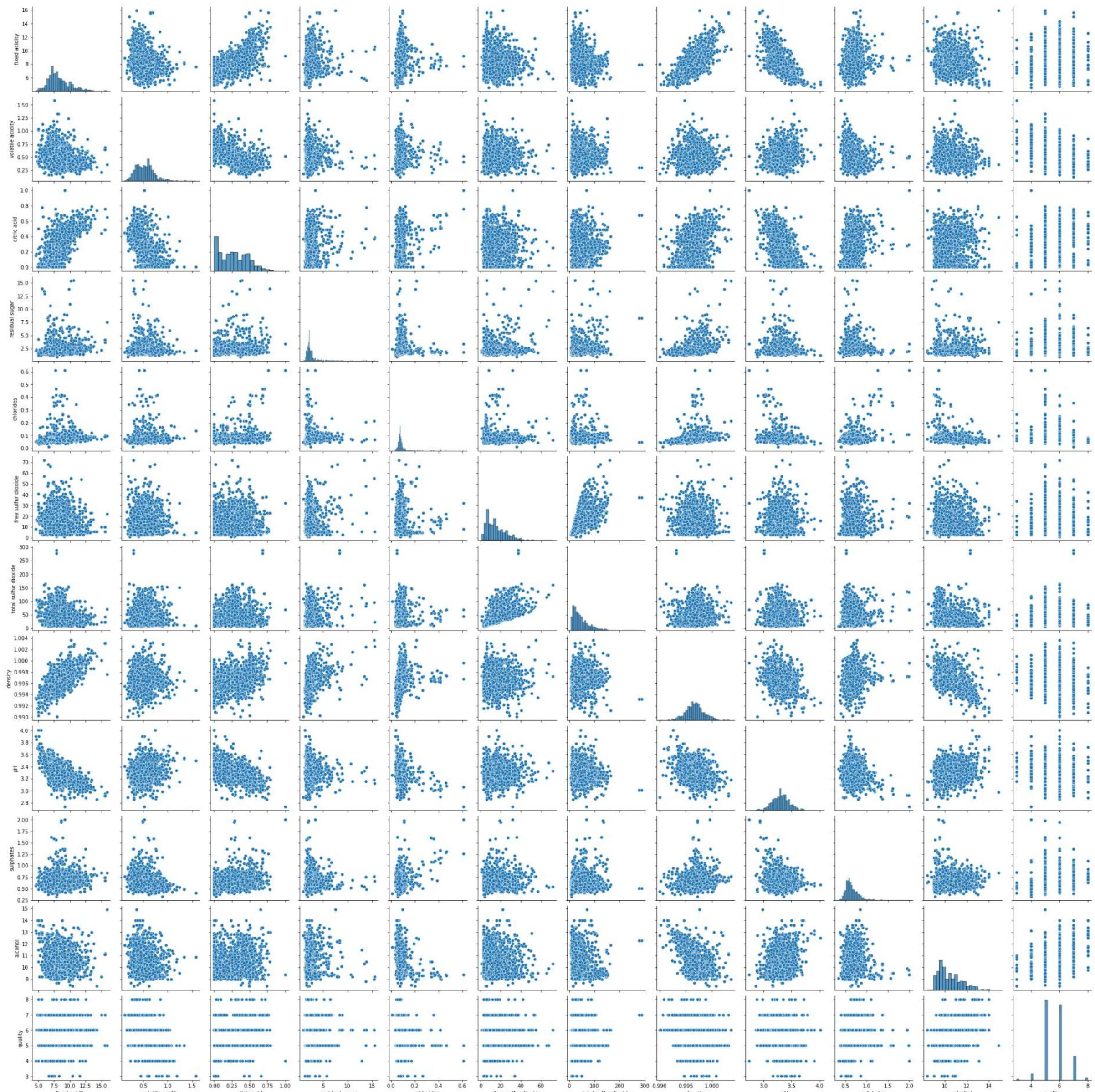
In [9]:
```
df.columns
```

Out[9]:
```
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 [10]:
```
sns.pairplot(df)
```

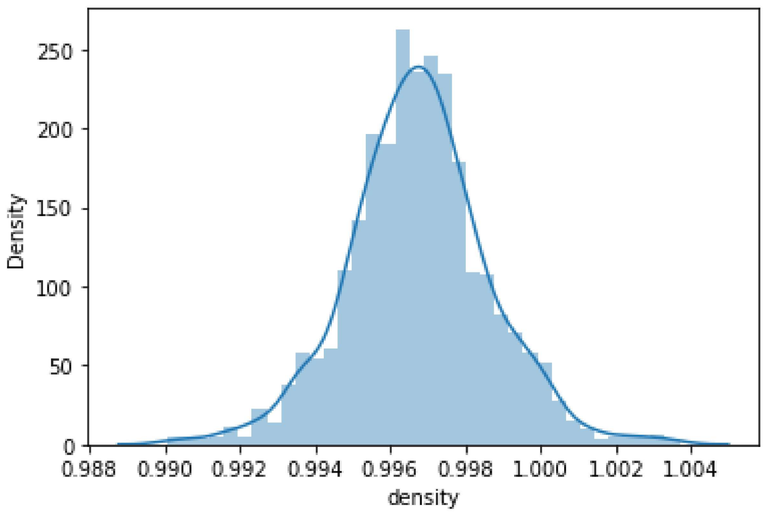
Out[10]: <seaborn.axisgrid.PairGrid at 0x1d3d2b57b20>



In [11]:
```
sns.distplot(df['density'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function for hi
stograms).
  warnings.warn(msg, FutureWarning)
```

Out[11]:
```
<AxesSubplot:xlabel='density', ylabel='Density'>
```

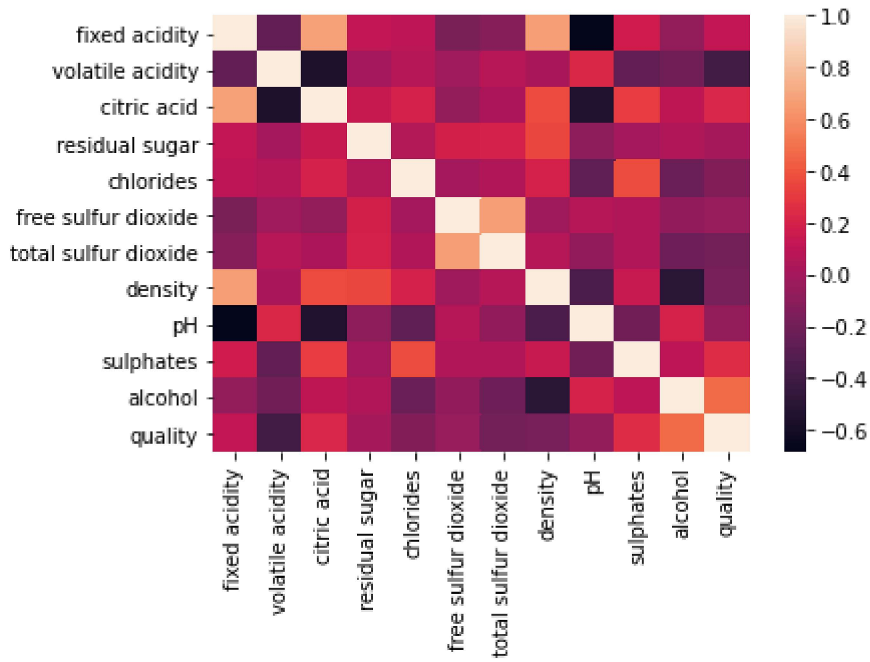


In [12]:
```
df1=df[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality']]
```

Plot Using Heat Map

In [13]:
```
sns.heatmap(df1.corr())
```

Out[13]: <AxesSubplot:>



To Train The Model-Model Building

we are going to train Linera Regression Model;We need to split out data into two variables x and y where x is independent variable(input) and y is dependent on x(output) we could ignore address column as it required for our model

In [17]:
```
x=df1[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol' ]]
y=df1['quality']
```

To Split my dataset into training and test data

In [20]:
```
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 [21]:
```
from sklearn.linear_model import LinearRegression
lr= LinearRegression()
lr.fit(x_train,y_train)
```

Out[21]: LinearRegression()

In [22]:
```
lr.intercept_
```

Out[22]: 78.07581406695047

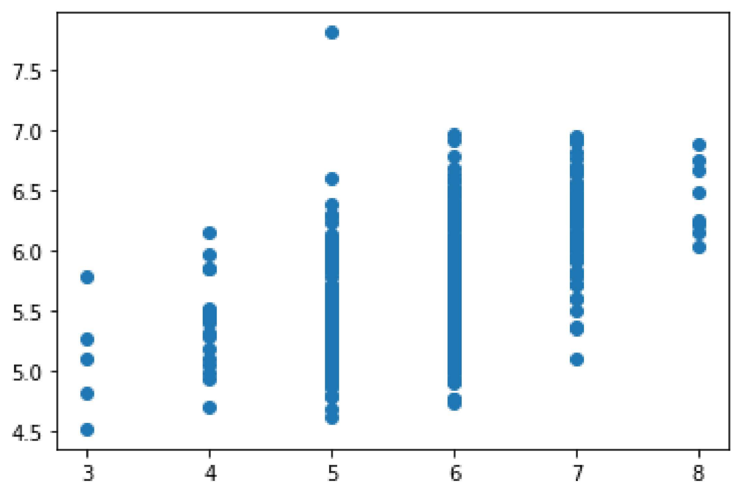
In [23]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[23]:

	Co-efficient
fixed acidity	0.095666
volatile acidity	-1.002742
citric acid	-0.261561
residual sugar	0.051312
chlorides	-1.804997
free sulfur dioxide	0.004852
total sulfur dioxide	-0.003235
density	-75.724863
pH	-0.050751
sulphates	1.050029
alcohol	0.232803

In [24]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[24]: <matplotlib.collections.PathCollection at 0x1d3db538e50>



In [25]:
```
lr.score(x_test,y_test)
```

Out[25]: 0.28802105194791683

ACURACY

In [26]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [27]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
rr.score(x_test,y_test)
rr.score(x_train,y_train)
```

Out[27]:
```
0.37049292434774606
```

In [28]:
```
rr.score(x_test,y_test)
```

Out[28]:
```
0.29628724440116716
```

In [29]:
```
la = Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[29]:
```
Lasso(alpha=10)
```

In [30]:
```
la.score(x_test,y_test)
```

Out[30]:
```
-0.00023154654113533013
```

In [31]:
```
from sklearn.linear_model import ElasticNet
en = ElasticNet()
en.fit(x_train,y_train)
```

Out[31]:
```
ElasticNet()
```

In [32]:
```
print(en.coef_)
```

```
[ 0.         -0.          0.          0.         -0.          0.00156272
 -0.00489567 -0.         -0.          0.          0.        ]
```

In [33]:
```

print(en.intercept_)

```

```
5.844068197708214
```

In [34]:
```
print(en.predict(x_test))
```

```
[5.75553111 5.77511379 5.81938234 5.37481655 5.67825071 5.71480303
 5.43200187 5.75553111 5.77001061 5.67991081 5.80115486 5.32075666
 5.79625919 5.61314166 5.6918874  5.67032698 5.25784562 5.37127608
 5.77001061 5.39450936 5.20951146 5.79469647 5.45210971 5.32700754
 5.37752696 5.75688632 5.74417705 5.72240912 5.60626824 5.62699862
 5.38607323 5.6877244  5.80938348 5.31429827 5.71335045 5.63022144
 5.78334241 5.70574436 5.74022157 5.44137818 5.78334241 5.67866573
 5.77292854 5.7698031  5.3939842  5.59033616 5.7419918  5.55491878
 5.65407725 5.77001061 5.67991081 5.59178875 5.59064105 5.62699862
 5.76667766 5.6785556  5.62991655 5.7964667  5.69282759 5.76824038
 5.76375973 5.22263575 5.48054354 5.76532245 5.74240682 5.76490743
 5.72907502 5.55314855 5.61960005 5.77136582 5.63699747 5.6332495
 5.70928483 5.23649271 5.73865885 5.53887656 5.77844674 5.54023177
 5.54981559 5.81448667 5.71886866 5.69480533 5.56043698 5.3728388
 5.77355107 5.78823808 5.73105276 5.6785556  5.79625919 5.5976246
 5.74730249 5.50971005 5.69595302 5.60231276 5.64262581 5.75063544
 5.67991081 5.26639189 5.71261778 5.74886521 5.63272434 5.74042908
 5.44565132 5.74532475 5.65897292 5.63272434 5.42512845 5.45137704
 5.25117972 5.75334586 5.7964667  5.64251568 5.61408184 5.79313375
 5.5286702  5.41075909 5.79959214 5.65908305 5.73105276 5.74553226
 5.73574092 5.76375973 5.74886521 5.35732174 5.7698031  5.75396839
 5.77532131 5.79490399 5.59960234 5.44211085 5.7341782  5.39784231
 5.75573863 5.17327678 5.60512055 5.67751804 5.79469647 5.44137818
 5.80625804 5.7686554  5.72532704 5.77313605 5.70949234 5.7964667
 5.55825173 5.77844674 5.74396954 5.76042678 5.74355452 5.74824268
 5.79313375 5.62991655 5.38961369 5.6440784  5.80292509 5.74886521
 5.57783441 5.76042678 5.58638068 5.54846039 5.65533508 5.77844674
 5.63387203 5.3225269  5.44804408 5.53221066 5.64387089 5.7553236
 5.69793077 5.74157677 5.73220046 5.7686554  5.7631372  5.75313835
 5.3939842  5.73220046 5.35596654 5.76844789 5.75490858 5.69126487
 5.7898008  5.65064692 5.64522609 5.67480763 5.63897522 5.6543949
 5.5250196  5.7075146  5.79136352 5.73261548 5.26117858 5.3063873
 5.75042793 5.72261663 5.74334701 5.76469992 5.56804307 5.72709728
 5.70772211 5.77823923 5.49200774 5.73886636 5.79469647 5.7208464
 5.7063669  5.78823808 5.77021812 5.70282644 5.60523068 5.77355107
 5.71126257 5.64564112 5.79469647 5.77313605 5.76490743 5.55085316
 5.5976246  5.66012062 5.57376879 5.56085201 5.81604938 5.80115486
 5.46856695 5.56960579 5.43200187 5.81448667 5.40710849 5.68928712
 5.71751345 5.72553456 5.54397974 5.6096012  5.78334241 5.54648264
 5.7341782  5.41981776 5.66439375 5.63887784 5.62054023 5.77157333
 5.69928598 5.70595188 5.81115371 5.73907387 5.6652238  5.70792962
 5.49346032 5.5606445  5.79313375 5.46721174 5.61647461 5.81115371
 5.45387994 5.77490628 5.67053449 5.74001405 5.67397758 5.57523413
 5.7141805  5.53085545 5.53680144 5.62814632 5.70261893 5.75709383
 5.41200416 5.63543475 5.80469533 5.50888    5.77844674 5.32877777
 5.66637149 5.74532475 5.78823808 5.63251683 5.25159475 5.2429511
 5.66876426 5.74396954 5.43335707 5.7831349  5.809591   5.64584863
 5.58387777 5.54002425 5.48346147 5.36669806 5.54887541 5.77334356
 5.7130328  5.3002338  5.6532472  5.61377695 5.77178084 5.73594843
 5.64012291 5.66324605 5.28804969 5.70011602 5.69480533 5.68595417
 5.64938909 5.67657786 5.27867338 5.79313375 5.72282414 5.48877216
 5.76490743 5.74573977 5.63939024 5.66543131 5.73771866 5.42648366
 5.76042678 5.43294205 5.76490743 5.74313949 5.79469647 5.72907502
 5.38242263 5.81938234 5.59856478 5.67386744 5.4507545  5.80448781
 5.80469533 5.79313375 5.7408441  5.78000946 5.64522609 5.70438916
 5.60346045 5.35419631 5.62970903 5.75136811 5.44555394 5.79959214
 5.62897636 5.72105391 5.7341782  5.78021698 5.73907387 5.72001635
```

```
 5.63929286 5.70792962 5.71126257 5.77532131 5.62345816 5.74573977
 5.56429509 5.6718897  5.7408441  5.67324491 5.70376662 5.66012062
 5.66533393 5.72886751 5.76490743 5.55616385 5.68970215 5.70397413
 5.57356127 5.75886406 5.64324835 5.71792847 5.74907272 5.68834694
 5.79979966 5.64230817 5.63230931 5.27867338 5.6585579  5.70167874
 5.37867466 5.57376879 5.2972185  5.64991425 5.61116391 5.63658245
 5.69126487 5.72240912 5.72574207 5.54435246 5.29232283 5.4905424
 5.28430172 5.76355222 5.67772555 5.25221728 5.5645026  5.44471113
 5.67616283 5.41586227 5.56928814 5.4749152  5.34898299 5.68480648
 5.74157677 5.7196987  5.59252142 5.70574436 5.36961599 5.78490513
 5.80292509 5.77803172 5.78177969 5.75219816 5.42138048 5.76532245
 5.75199065 5.71105506 5.72730479 5.4599233  5.79469647 5.64481107
 5.77688402 5.76355222 5.75199065 5.79469647 5.18264034 5.5909587
 5.74553226 5.79490399 5.61897751 5.72553456 5.75001291 5.65189199
 5.66595647 5.77844674 5.75688632 5.80782076 5.65897292 5.55043813
 5.63543475 5.5606445  5.77511379 5.78823808 5.72771981 5.78823808
 5.81115371 5.67365993 5.60001737 5.73594843 5.61939254 5.77355107
 5.58095985 5.80625804 5.67501514 5.58689308 5.67574781 5.65522495
 5.71324031 5.75063544 5.65949808 5.37773447 5.67699288 5.65032928
 5.74219931 5.73594843 5.80469533 5.30367689 5.48148373 5.70501169
 5.66220849 5.58356013 5.589811   5.67386744 5.7196987  5.72261663
 5.74553226 5.70595188 5.65251453 5.6507443  5.81115371 5.57075348
 5.73105276 5.7964667  5.63366452 5.76709268 5.80448781 4.48782153]
```

In [35]:
```python
print(en.score(x_test,y_test))
```

```
0.01857441769029211
```

In [36]:
```python
# Evaluation Metrics
from sklearn import metrics
```

In [37]:
```python
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 0.5440139286687178
```

In [38]:
```python
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 0.5017195747690457
```

In [39]:
```python
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prec
```

```
Root Mean Squared Error: 0.7083216605251076
```