Problem statement

predicting the house price in USA.To create a model to help him 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("wine")
```

To display top 10 rows

In [3]:
```
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]:

```
# Display the statistical summary
df.describe()
```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphat
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.6581
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.1695
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.3300
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.5500
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.6200
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.7300
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.0000

In [6]:

```
# To display the col headings
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]:

```
cols=df.dropna(axis=1)
```

In [8]:

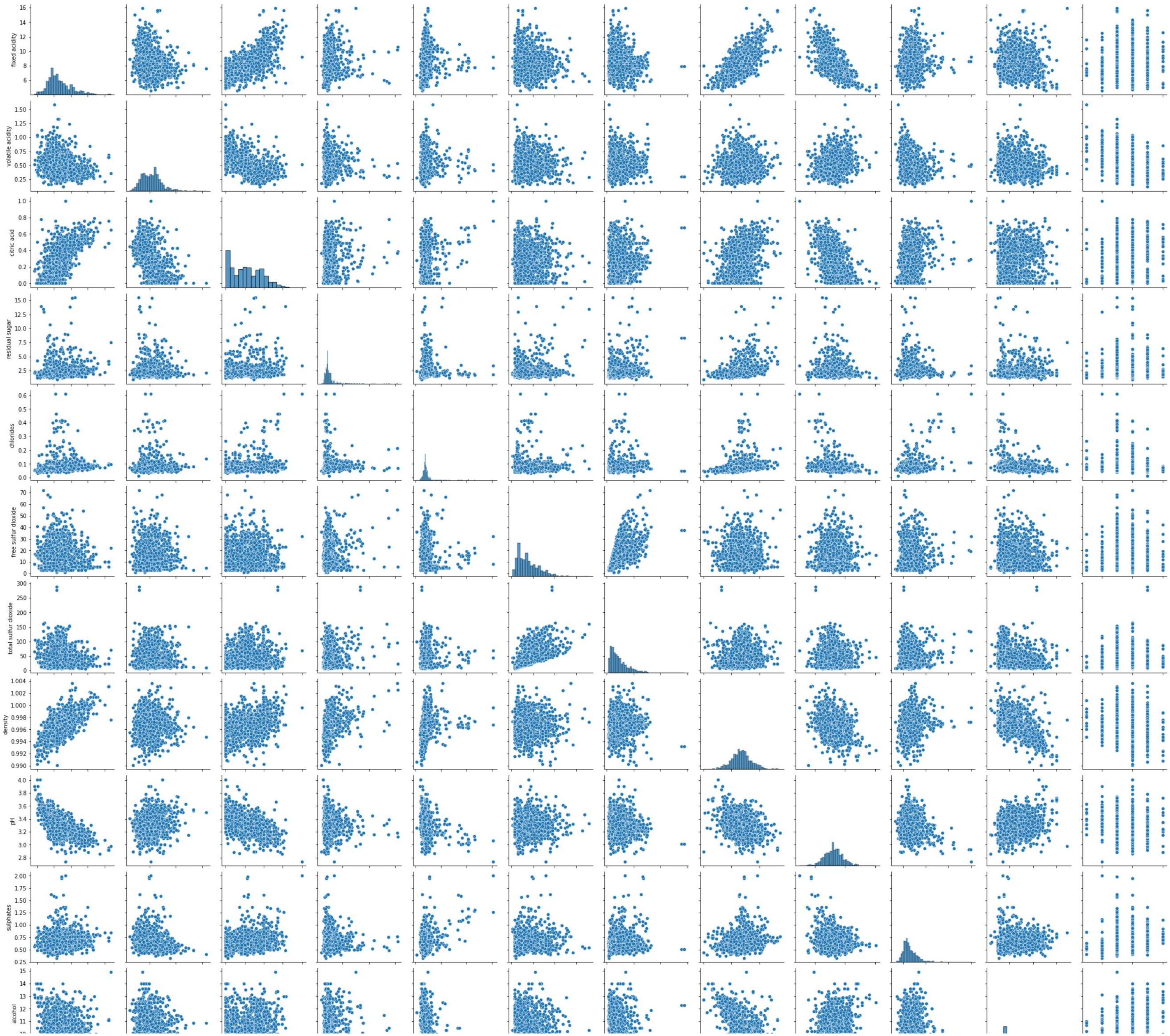
```
cols.columns
```

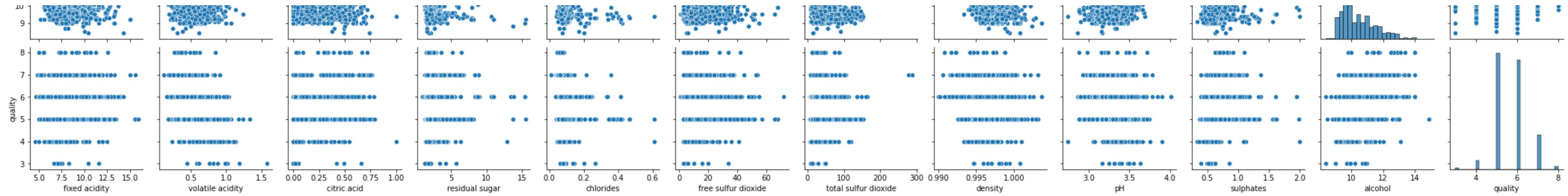
Out[8]:

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

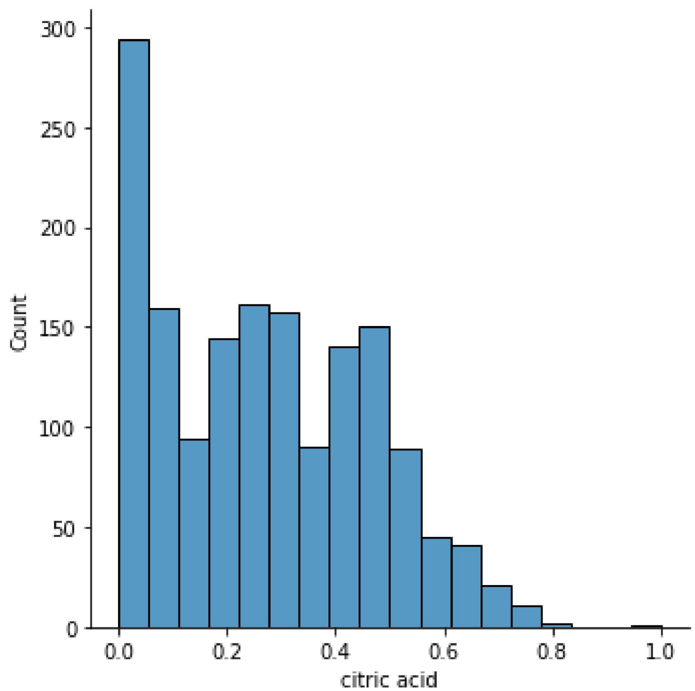
EDA and Visualization

In [9]:
```
sns.pairplot(cols)
```
Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x19141c3b9d0>
```


fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol



In [10]:
```
sns.displot(df['citric acid'])
```
Out[10]: <seaborn.axisgrid.FacetGrid at 0x19146aa7550>

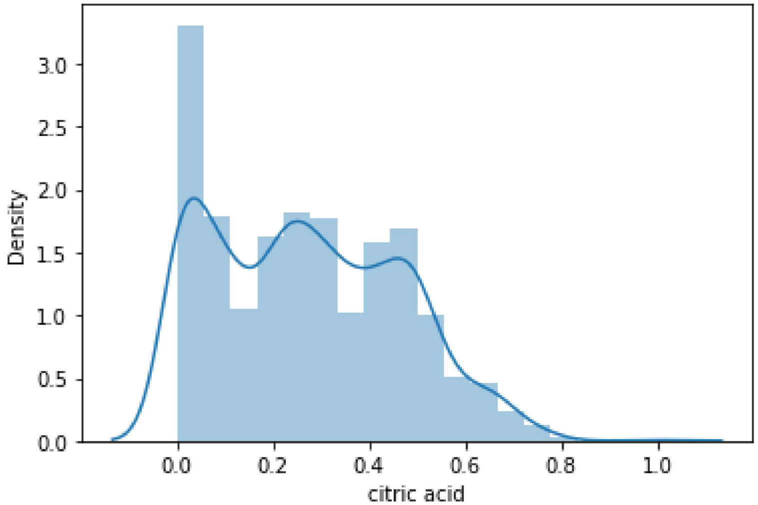


In [11]:

```
# We use displot in older version we get distplot use displot
sns.distplot(df['citric acid'])
```

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

Out[11]: <AxesSubplot:xlabel='citric acid', ylabel='Density'>



In [12]:

```
df1=cols[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar']]
df1
```

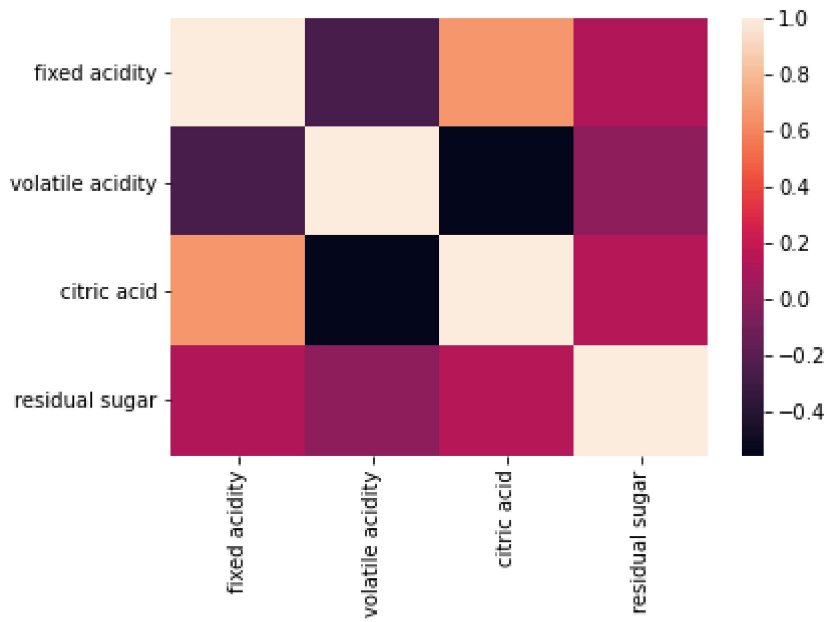
Out[12]:

	fixed acidity	volatile acidity	citric acid	residual sugar
0	7.4	0.700	0.00	1.9
1	7.8	0.880	0.00	2.6
2	7.8	0.760	0.04	2.3
3	11.2	0.280	0.56	1.9
4	7.4	0.700	0.00	1.9
...	...	...	...	...
1594	6.2	0.600	0.08	2.0
1595	5.9	0.550	0.10	2.2
1596	6.3	0.510	0.13	2.3
1597	5.9	0.645	0.12	2.0
1598	6.0	0.310	0.47	3.6

1599 rows × 4 columns

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

Out[13]: <AxesSubplot:>



To train the model - MODEL BUILD

Going to train linear regression model;We split our data into 2 variables x and y where x is independent var(input) and y is dependent on x(output), we could ignore address col as it is not required for our model

In [14]:
```
x=df1[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar']]
y=df1[['volatile acidity']]
```

To split the dataset into test data

In [15]:
```python
# importing lib for splitting test data
from sklearn.model_selection import train_test_split
```

In [16]:
```python
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

In [17]:
```python
from sklearn.linear_model import LinearRegression

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

Out[17]:
```
LinearRegression()
```

In [18]:
```python
print(lr.intercept_)
```

```
[0.]
```

In [19]:
```python
print(lr.score(x_test,y_test))
```

```
1.0
```

In [20]:
```python
coeff=pd.DataFrame(lr.coef_)
coeff
```

Out[20]:

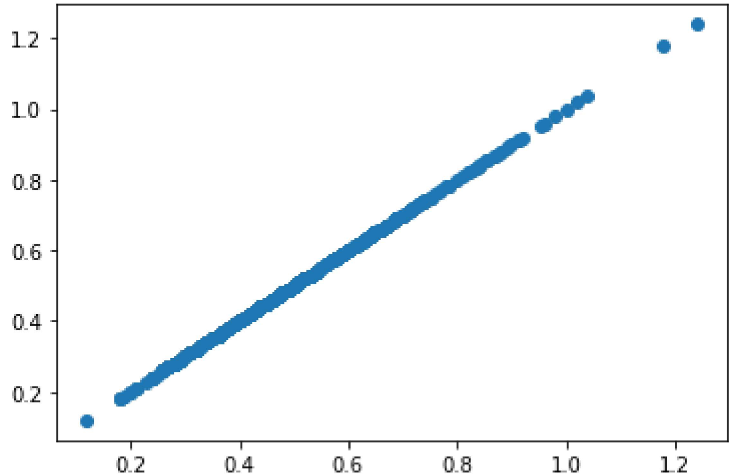
	0	1	2	3
0	-3.785235e-17	1.0	6.085788e-16	1.074156e-17

In [21]:

```
pred = lr.predict(x_test)
plt.scatter(y_test,pred)
```

Out[21]:

```
<matplotlib.collections.PathCollection at 0x1914a551d30>
```



In [22]:

```
from sklearn.linear_model import Ridge,Lasso
```

In [23]:

```
rr=Ridge(alpha=20)
rr.fit(x_train,y_train)
```

Out[23]:

```
Ridge(alpha=20)
```

In [24]:

```
rr.score(x_test,y_test)
```

Out[24]:

```
0.8810966096026683
```

In [25]:

```
la=Lasso(alpha=20)
la.fit(x_train,y_train)
```

Out[25]:

```
Lasso(alpha=20)
```

In [26]:

```
la.score(x_test,y_test)
```

Out[26]:

```
-3.097551325437209e-06
```

ELASTIC NET

In [27]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[27]:
```
ElasticNet()
```

In [28]:
```
print(en.coef_)
```

```
[-0.  0. -0.  0.]
```

In [29]:
```
print(en.intercept_)
```

```
[0.52791332]
```

In [30]:

```
prediction=en.predict(x_test)
prediction
```

```
Out[30]: array([0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
                0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
```

```
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
```

```
        0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
        0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
        0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
        0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
        0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
        0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
        0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
        0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
        0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332,
        0.52791332, 0.52791332, 0.52791332, 0.52791332, 0.52791332])
```

In [31]:
```
print(en.score(x_test,y_test))
```

```
-3.097551325437209e-06
```

EVALUATION METRICS

In [32]:
```
from sklearn import metrics
```

In [33]:
```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 0.141663054810843
```

In [34]:
```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 0.030854459722282l2
```

In [35]:
```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 0.17565437575614826
```