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(r'C:\Users\user\Downloads\11_winequality-red.csv')
df
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
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
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66

1599 rows × 12 columns

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	alco
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	1
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	1

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 di
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.4
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.8
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.0
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.0
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.0
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.0
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.0

In [6]:

```
df.columns
```

Out[6]:

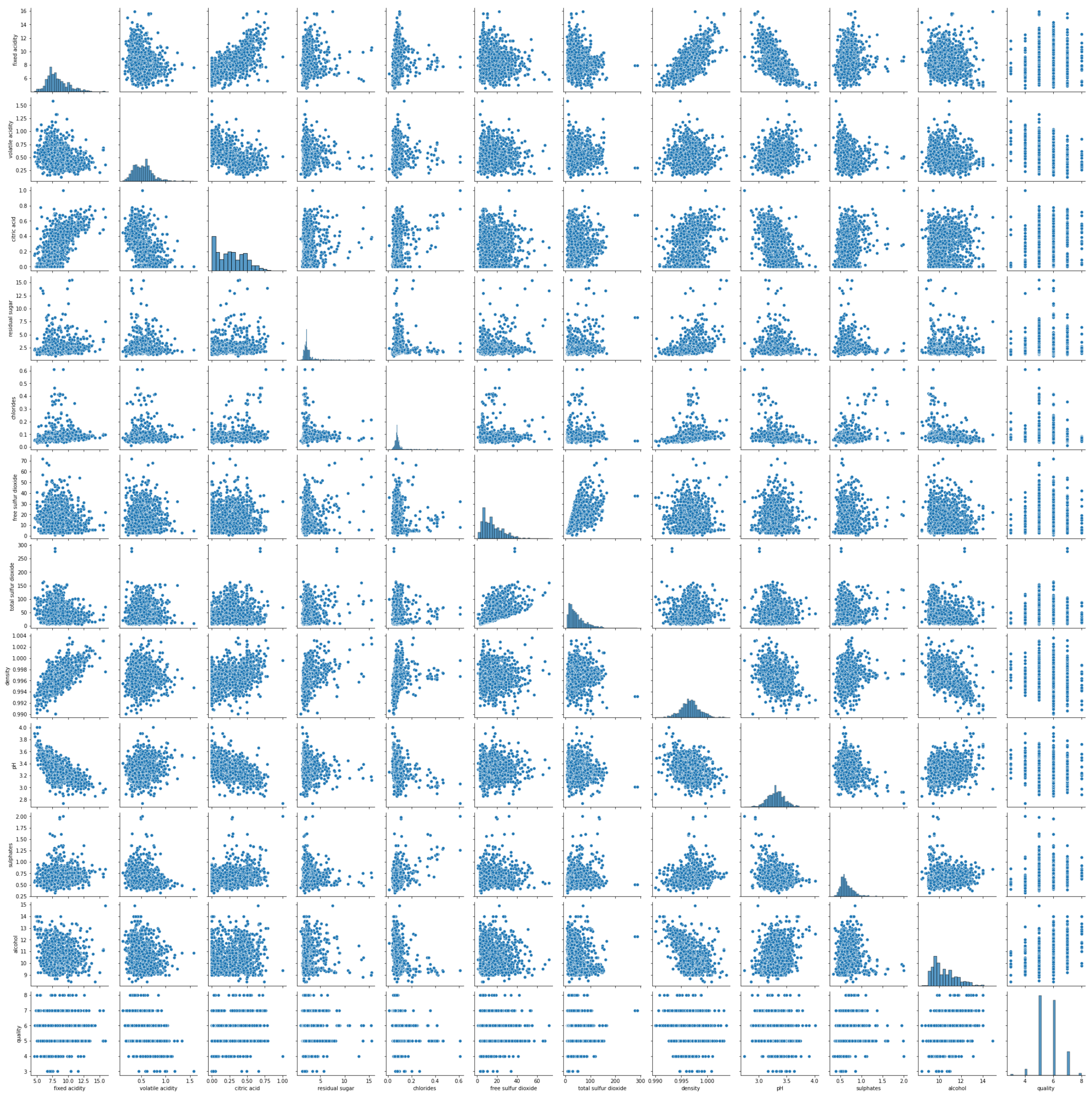
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual suga
r',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'densit
y',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [7]:

```
sns.pairplot(df)
```

Out[7]:

```
<seaborn.axisgrid.PairGrid at 0x1dc3c24f490>
```



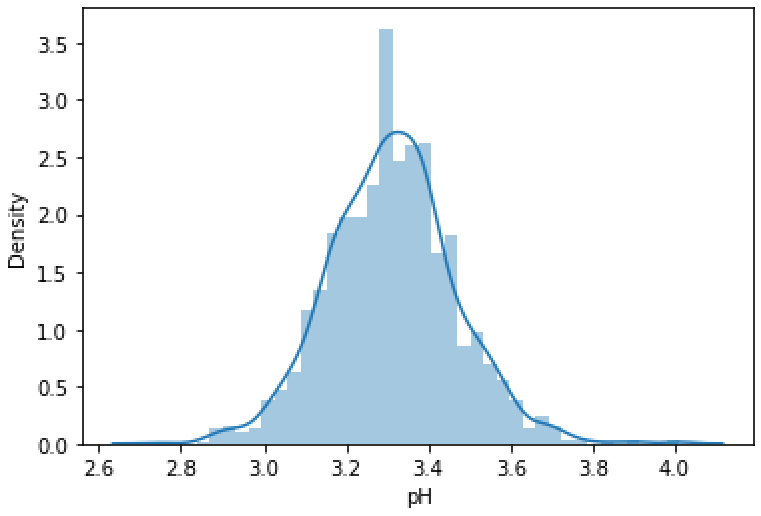
In [8]:

```
sns.distplot(df['pH'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `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 flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[8]:

```
<AxesSubplot:xlabel='pH', ylabel='Density'>
```

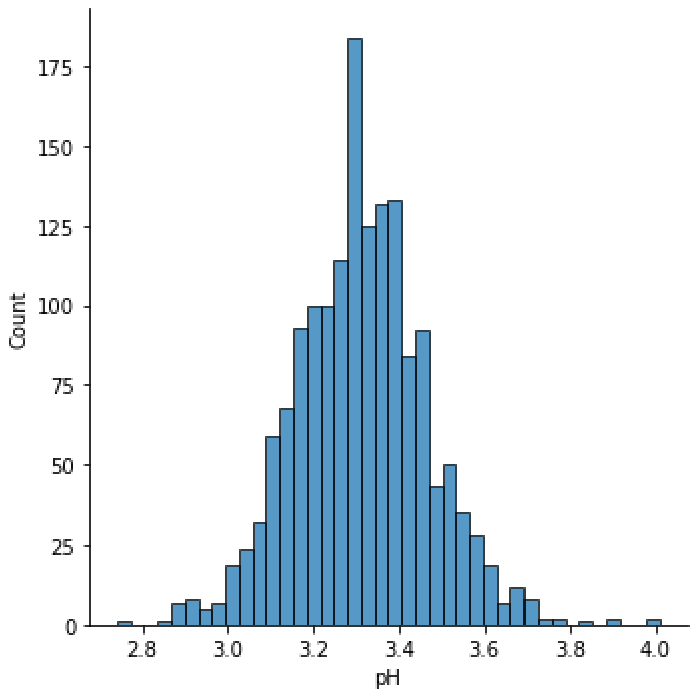


In [9]:

```
sns.displot(df["pH"])
```

Out[9]:

```
<seaborn.axisgrid.FacetGrid at 0x1dc37480340>
```



In [10]:

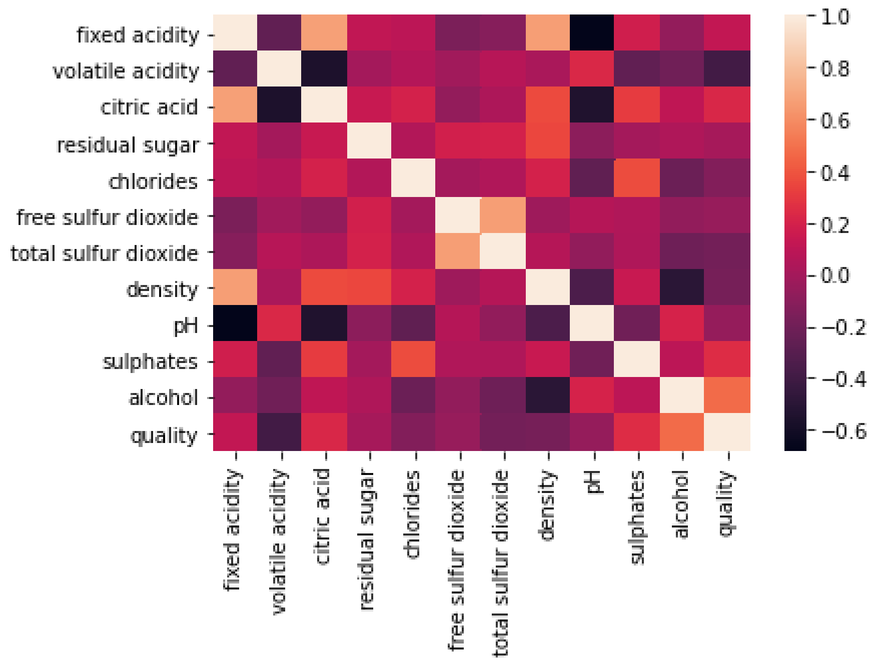
```
df1=df[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality']]
```

In [11]:

```
sns.heatmap(df1.corr())
```

Out[11]:

```
<AxesSubplot:>
```



In [12]:

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

In [13]:

```
from sklearn.model_selection import train_test_split
```

In [14]:

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

In [15]:

```
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(x_train,y_train)#ValueError: Input contains NaN, infinity or a value too large for
```

Out[15]:

```
LinearRegression()
```

In [16]:

```
print(lr.intercept_)
```

```
[-60.44143358]
```

In [17]:

```
coef= pd.DataFrame(lr.coef_)
coef
```

Out[17]:

	0	1	2	3	4	5	6	7	
0	-0.095235	-0.02507	-0.054099	-0.025459	-0.456016	0.001445	-0.000719	64.227282	-0.092

In [18]:

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

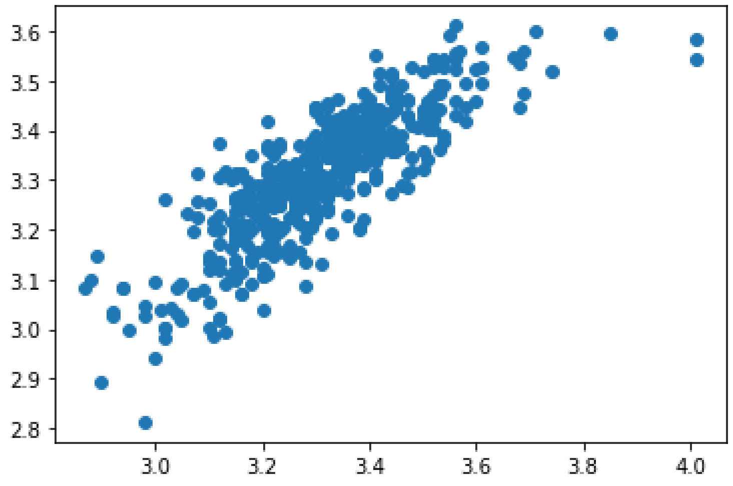
```
0.6906683679483773
```

In [19]:

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

Out[19]:

```
<matplotlib.collections.PathCollection at 0x1dc44c1c9d0>
```



In [20]:

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

Out[20]:

```
0.6906683679483773
```

In [21]:

```
lr.score(x_train,y_train)
```

Out[21]:

```
0.703243766350664
```

In [22]:

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

In [23]:

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[23]:

```
Ridge(alpha=10)
```

In [24]:

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

Out[24]:

```
0.5200743893210129
```

In [25]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[25]:

```
Lasso(alpha=10)
```

In [26]:

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

Out[26]:

```
-2.672632216116355e-05
```

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

In [29]:

```
print(en.intercept_)
```

```
[3.31135836]
```

In [30]:

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

```
[3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836
 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836 3.31135836]
```

In [31]:

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

```
-2.672632216116355e-05
```

Evaluation Metrics

In [32]:

```
from sklearn import metrics
```

In [33]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 0.12136729222520108
```

In [34]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 0.024956623578158078
```

In [35]:

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

```
Root Mean Squared Error: 0.15797665516828135
```