Problem Statement

Linear Regression

Import Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
a=pd.read_csv("wine.csv")
a
```

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

To display top 10 rows

In [3]:

```
c=a.head(15)
c
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alc
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
5	7.4	0.660	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	
6	7.9	0.600	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	
7	7.3	0.650	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	
8	7.8	0.580	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	
9	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	
10	6.7	0.580	0.08	1.8	0.097	15.0	65.0	0.9959	3.28	0.54	
11	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	
12	5.6	0.615	0.00	1.6	0.089	16.0	59.0	0.9943	3.58	0.52	
13	7.8	0.610	0.29	1.6	0.114	9.0	29.0	0.9974	3.26	1.56	
14	8.9	0.620	0.18	3.8	0.176	52.0	145.0	0.9986	3.16	0.88	

To find Missing values

In [4]:

```
c.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         15 non-null     float64
 1   volatile acidity      15 non-null     float64
 2   citric acid           15 non-null     float64
 3   residual sugar        15 non-null     float64
 4   chlorides             15 non-null     float64
 5   free sulfur dioxide   15 non-null     float64
 6   total sulfur dioxide  15 non-null     float64
 7   density               15 non-null     float64
 8   pH                    15 non-null     float64
 9   sulphates             15 non-null     float64
 10  alcohol               15 non-null     float64
 11  quality               15 non-null     int64
dtypes: float64(11), int64(1)
memory usage: 1.5 KB
```

To display summary of statistics

In [5]:

```
a.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

To display column heading

In [6]:

```
a.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')
```

Pairplot

In [7]:

```
s=a.dropna(axis=1)
s
```

Out[7]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	ε
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 [8]:

```
s.columns
```

Out[8]:

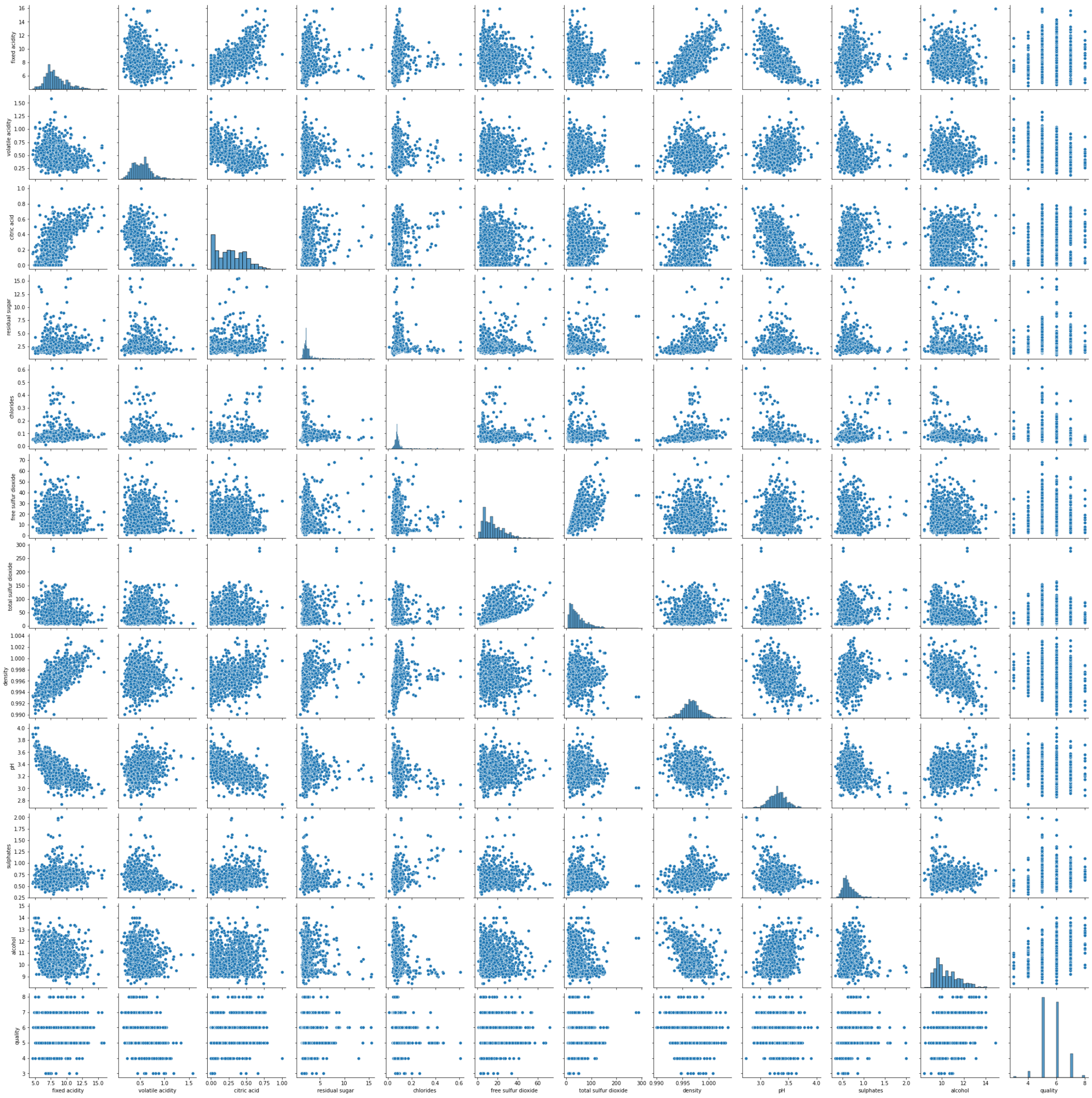
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual suga
r',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'densit
y',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [9]:

```
sns.pairplot(a)
```

Out[9]:

```
<seaborn.axisgrid.PairGrid at 0x13f98ad35b0>
```



Distribution Plot

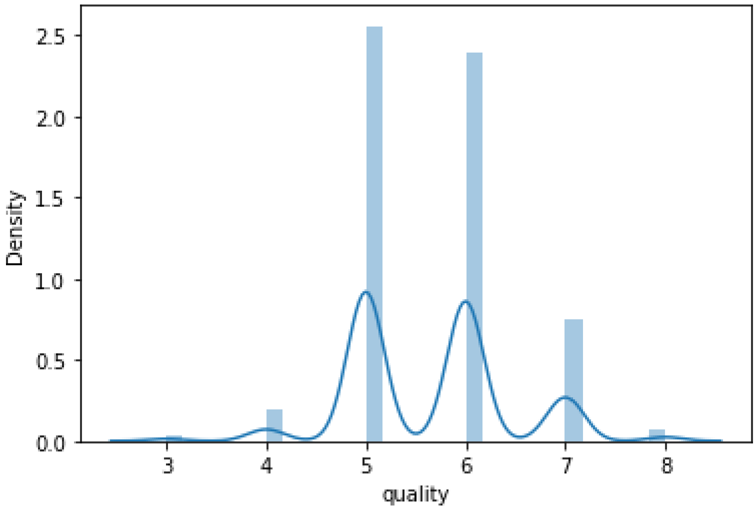
In [10]:

```
sns.distplot(a['quality'])
```

```
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[10]:

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



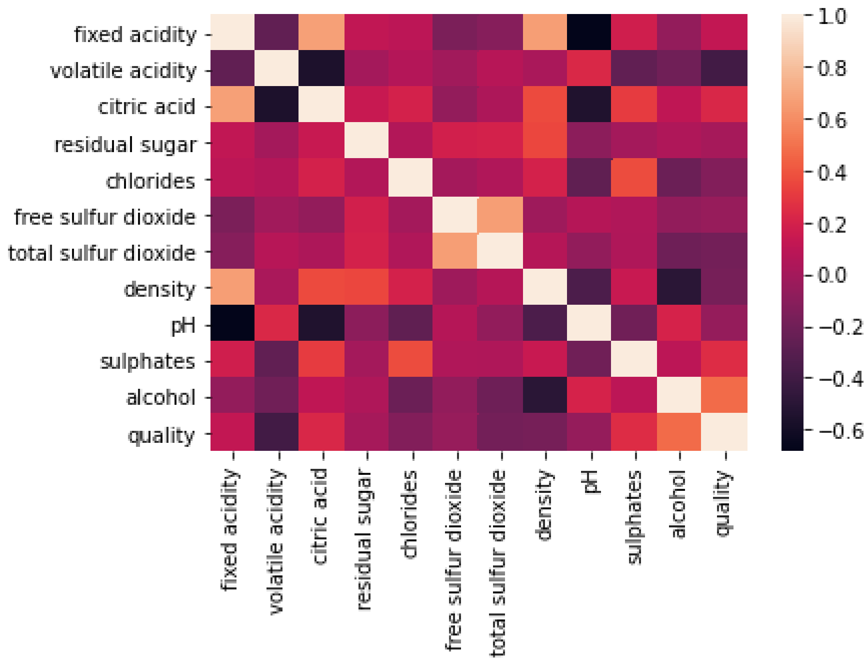
Correlation

In [11]:

```
b=s[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality']]
sns.heatmap(b.corr())
```

Out[11]:

<AxesSubplot:>



Train the model - Model Building

In [12]:

```
g=s[['quality']]
h=s['quality']
```

To split dataset into training end test

In [13]:

```
from sklearn.model_selection import train_test_split
g_train,g_test,h_train,h_test=train_test_split(g,h,test_size=0.6)
```

To run the model

In [14]:

```
from sklearn.linear_model import LinearRegression
```

In [15]:

```
lr=LinearRegression()
lr.fit(g_train,h_train)
```

Out[15]:

```
LinearRegression()
```

In [16]:

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

```
-2.6645352591003757e-15
```

Coeffecient

In [17]:

```
coeff=pd.DataFrame(lr.coef_,g.columns,columns=['Co-effecient'])
coeff
```

Out[17]:

	Co-effecient
quality	1.0

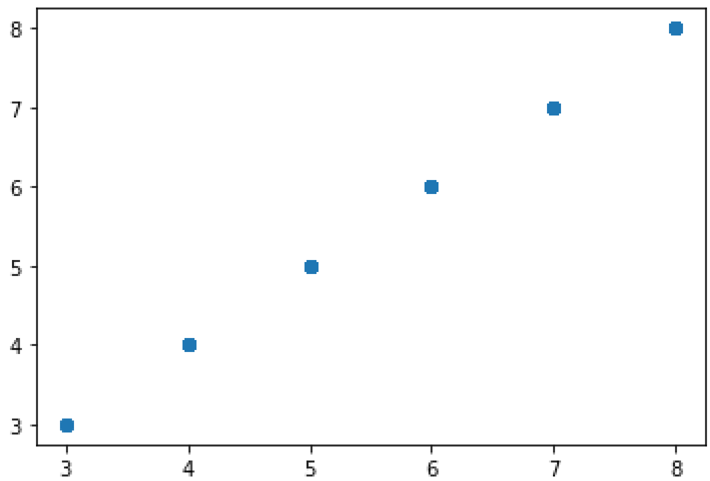
Best Fit line

In [18]:

```
prediction=lr.predict(g_test)
plt.scatter(h_test,prediction)
```

Out[18]:

```
<matplotlib.collections.PathCollection at 0x13fa13a0dc0>
```



To find score

In [19]:

```
print(lr.score(g_test,h_test))
```

```
1.0
```

Import Lasso and ridge

In [20]:

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

Ridge

In [21]:

```
ri=Ridge(alpha=5)
ri.fit(g_train,h_train)
```

Out[21]:

```
Ridge(alpha=5)
```

In [22]:

```
ri.score(g_test,h_test)
```

Out[22]:

```
0.9998503809459203
```

In [23]:

```
ri.score(g_train,h_train)
```

Out[23]:

```
0.9998503814630366
```

Lasso

In [24]:

```
l=Lasso(alpha=6)
l.fit(g_train,h_train)
```

Out[24]:

```
Lasso(alpha=6)
```

In [25]:

```
l.score(g_test,h_test)
```

Out[25]:

```
-3.456232135379267e-06
```

In [27]:

```
ri.score(g_train,h_train)
```

Out[27]:

```
0.9998503814630366
```

ElasticNet

In [28]:

```
from sklearn.linear_model import ElasticNet
e=ElasticNet()
e.fit(g_train,h_train)
```

Out[28]:

```
ElasticNet()
```

Coeffecient,intercept

In [29]:

```
print(e.coef_)
```

```
[0.11651054]
```

In [30]:

```
print(e.intercept_)
```

```
4.980170652045737
```

Prediction

In [31]:

```
d=e.predict(g_test)
d
```

Out[31]:

```
array([5.67923387, 5.79574441, 5.56272334, 5.56272334, 5.67923387,
       5.79574441, 5.32970226, 5.56272334, 5.56272334, 5.56272334,
       5.79574441, 5.79574441, 5.67923387, 5.67923387, 5.79574441,
       5.56272334, 5.67923387, 5.67923387, 5.56272334, 5.67923387,
       5.56272334, 5.79574441, 5.56272334, 5.56272334, 5.56272334,
       5.79574441, 5.67923387, 5.67923387, 5.56272334, 5.79574441,
       5.56272334, 5.79574441, 5.67923387, 5.79574441, 5.67923387,
       5.56272334, 5.56272334, 5.67923387, 5.67923387, 5.56272334,
       5.67923387, 5.56272334, 5.32970226, 5.56272334, 5.56272334,
       5.56272334, 5.67923387, 5.56272334, 5.79574441, 5.79574441,
       5.67923387, 5.79574441, 5.56272334, 5.67923387, 5.67923387,
       5.67923387, 5.67923387, 5.91225495, 5.91225495, 5.56272334,
       5.79574441, 5.56272334, 5.67923387, 5.67923387, 5.56272334,
       5.56272334, 5.67923387, 5.67923387, 5.67923387, 5.67923387,
       5.56272334, 5.56272334, 5.56272334, 5.67923387, 5.56272334,
       5.56272334, 5.4462128 , 5.67923387, 5.56272334, 5.56272334,
       5.56272334, 5.67923387, 5.67923387, 5.67923387, 5.79574441,
       5.56272334, 5.4462128 , 5.56272334, 5.56272334, 5.56272334,
```

In [32]:

```
print(e.score(g_test,h_test))
```

```
0.21944367053660874
```

Evaluation

In [33]:

```
from sklearn import metrics
print("Mean Absolute error:",metrics.mean_absolute_error(h_test,d))
```

```
Mean Absolute error: 0.6098848282606883
```

In [34]:

```
print("Mean Squared error:",metrics.mean_squared_error(h_test,d))
```

```
Mean Squared error: 0.5190665712618644
```

In [35]:

```
print("Mean Squared error:",np.sqrt(metrics.mean_squared_error(h_test,d)))
```

```
Mean Squared error: 0.7204627480042701
```

In [ ]:

```

```