Name : SHAHNAWAZ ALAM

@Bharat intern

Domain : Machine Learning Intern

Task#2 : Wine Quality Prediction

Importing Libraries/Packages

In [5]:

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

%matplotlib inline
```

Importing Dataset

In [6]:

```
print("Importing data...")
dataset = pd.read_csv(r"C:\Users\md naiyer azam\Desktop\winequality-red.csv")
print("Sucessfully imported.")
```

```
Importing data...
Sucessfully imported.
```

In [7]:

```
dataset.head()
```

Out[7]:

	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

In [8]:

```
dataset.tail()
```

Out[8]:

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

In [9]:

```
dataset.shape
```

Out[9]:

```
(1599, 12)
```

In [10]:

```
#info of data
dataset.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 [11]:

```
dataset.describe()
```

Out[11]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000

In [12]:

```
dataset.columns # to find the no of columns
```

Out[12]:

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

In [14]:

```
#shape of datasets
print("Shape of our datasets of Red-Wine:{s}".format(s = dataset.shape))
print("Column headers/names: {s}".format(s = list(dataset)))
```

```
Shape of our datasets of Red-Wine:(1599, 12)
Column headers/names: ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur diox
ide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol', 'quality']
```

In [15]:

```
dataset['quality'].unique()
```

Out[15]:

```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [16]:

```
dataset.quality.value_counts().sort_index()
```

Out[16]:

```
3     10
4     53
5    681
6    638
7    199
8     18
Name: quality, dtype: int64
```

Exploratory Data Analysis for Wine quality Prediction

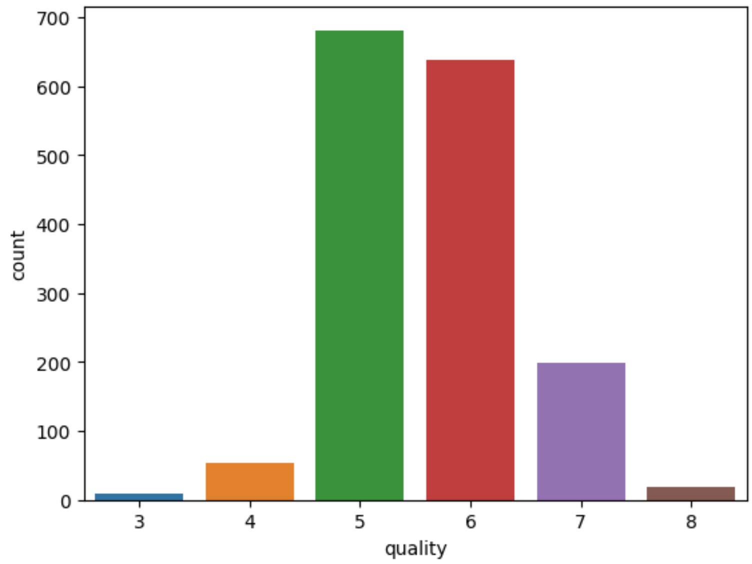
We will create some simple plot for visualizing the data.

In [17]:

```
sns.countplot(x='quality', data=dataset)
```

Out[17]:

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



In [18]:

```
dataset['alcohol'].describe()
```

Out[18]:

```
count    1599.000000
mean       10.422983
std         1.065668
min         8.400000
25%         9.500000
50%        10.200000
75%        11.100000
max        14.900000
Name: alcohol, dtype: float64
```

In [19]:

```
dataset['sulphates'].describe()
```

Out[19]:

```
count    1599.000000
mean        0.658149
std         0.169507
min         0.330000
25%         0.550000
50%         0.620000
75%         0.730000
max         2.000000
Name: sulphates, dtype: float64
```

In [20]:

```
dataset['citric acid'].describe()
```

Out[20]:

```
count    1599.000000
mean        0.270976
std         0.194801
min         0.000000
25%         0.090000
50%         0.260000
75%         0.420000
max         1.000000
Name: citric acid, dtype: float64
```

In [21]:

```
dataset['fixed acidity'].describe()
```

Out[21]:

```
count    1599.000000
mean        8.319637
std         1.741096
min         4.600000
25%         7.100000
50%         7.900000
75%         9.200000
max        15.900000
Name: fixed acidity, dtype: float64
```

In [22]:

```
dataset['residual sugar'].describe()
```

Out[22]:

```
count    1599.000000
mean        2.538806
std         1.409928
min         0.900000
25%         1.900000
50%         2.200000
75%         2.600000
max        15.500000
Name: residual sugar, dtype: float64
```

In [23]:

```
Q1 = dataset.quantile(0.25)
Q3 = dataset.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
fixed acidity            2.100000
volatile acidity         0.250000
citric acid              0.330000
residual sugar           0.700000
chlorides                0.020000
free sulfur dioxide     14.000000
total sulfur dioxide    40.000000
density                  0.002235
pH                       0.190000
sulphates                0.180000
alcohol                  1.600000
quality                  1.000000
dtype: float64
```

In [60]:

```
#The data point where we have False that means these values are valid whereas True indicates presence of an outlier.
print(dataset < (Q1 - 1.5 * IQR)) |(dataset > (Q3 + 1.5 * IQR))
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0             False             False        False           False      False
1             False             False        False           False      False
2             False             False        False           False      False
3             False             False        False           False      False
4             False             False        False           False      False
...             ...               ...          ...             ...        ...
1594          False             False        False           False      False
1595          False             False        False           False      False
1596          False             False        False           False      False
1597          False             False        False           False      False
1598          False             False        False           False      False

      free sulfur dioxide  total sulfur dioxide  density     pH  sulphates  \
0                   False                 False    False  False      False
1                   False                 False    False  False      False
2                   False                 False    False  False      False
3                   False                 False    False  False      False
4                   False                 False    False  False      False
```

In [29]:

```
dataset_out = dataset[~((dataset < (Q1 - 1.5 * IQR)) |(dataset > (Q3 + 1.5 * IQR))).any(axis=1)]

dataset_out.shape
```

Out[29]:

```
(1179, 12)
```

In [30]:

```
dataset_out
```

Out[30]:

	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

1179 rows × 12 columns

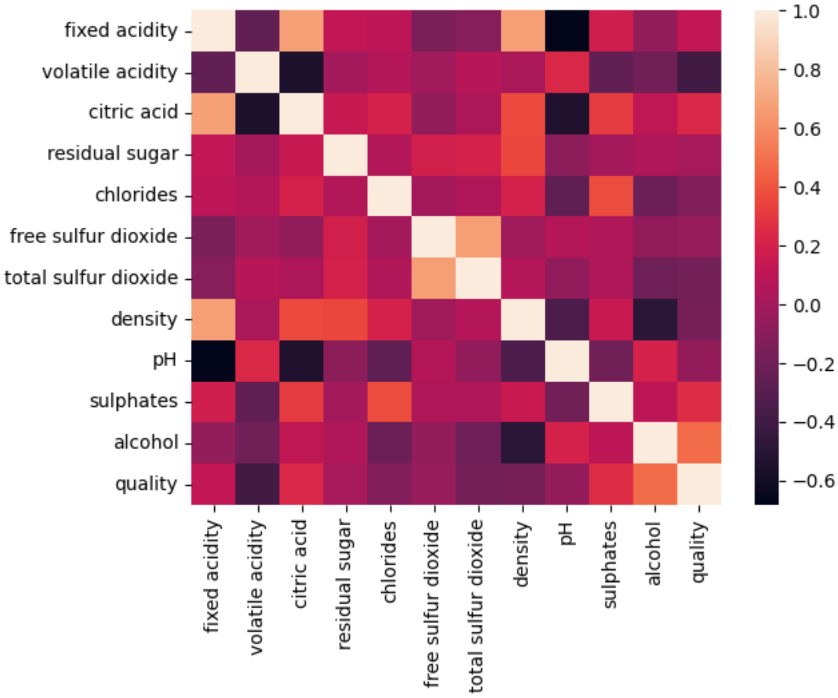
In [31]:

```
correlations = dataset_out.corr()['quality'].drop('quality')
print(correlations)
```

```
fixed acidity           0.113422
volatile acidity       -0.346962
citric acid             0.212133
residual sugar          0.007934
chlorides              -0.190869
free sulfur dioxide    -0.003609
total sulfur dioxide   -0.203374
density                -0.215375
pH                     -0.060288
sulphates               0.413533
alcohol                 0.492551
Name: quality, dtype: float64
```

In [32]:

```
sns.heatmap(dataset.corr())
plt.show()
```



In [33]:

```
#impact of various factor on quality
correlations.sort_values(ascending=False)
```

Out[33]:

```
alcohol                 0.492551
sulphates               0.413533
citric acid             0.212133
fixed acidity           0.113422
residual sugar          0.007934
free sulfur dioxide    -0.003609
pH                     -0.060288
chlorides              -0.190869
total sulfur dioxide   -0.203374
density                -0.215375
volatile acidity       -0.346962
Name: quality, dtype: float64
```

In [34]:

```
def get_features(correlation_threshold):
    abs_corrs = correlations.abs()
    high_correlations = abs_corrs[abs_corrs > correlation_threshold].index.values.tolist()
    return high_correlations
```

In [35]:

```
# taking features with correlation more than 0.05 as input x and quality as target variable y
features = get_features(0.05)
print(features)
x = dataset_out[features]
y = dataset_out['quality']
```

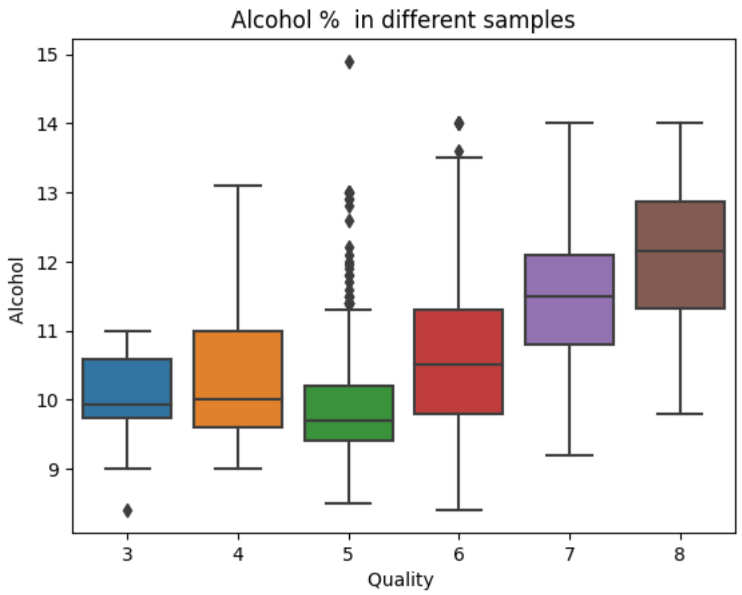
```
['fixed acidity', 'volatile acidity', 'citric acid', 'chlorides', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'al
cohol']
```

In [36]:

```
#to finding the no of outiers we have in our dataset with properties
bx = sns.boxplot(x='quality', y='alcohol', data = dataset)
bx.set(xlabel='Quality ', ylabel='Alcohol ', title='Alcohol %  in different samples')
```

Out[36]:

```
[Text(0.5, 0, 'Quality '),
 Text(0, 0.5, 'Alcohol '),
 Text(0.5, 1.0, 'Alcohol %  in different samples')]
```

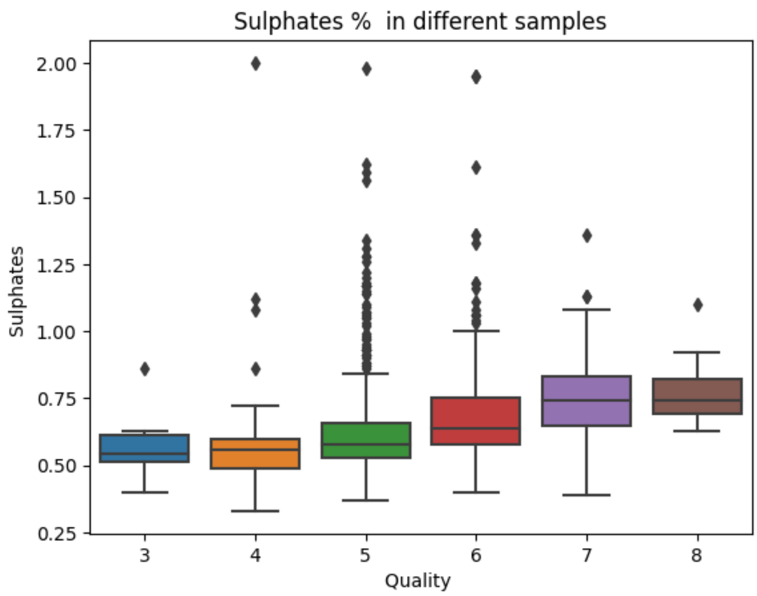


In [37]:

```
bx = sns.boxplot(x='quality', y='sulphates', data = dataset)
bx.set(xlabel='Quality ', ylabel='Sulphates ', title='Sulphates %  in different samples')
```

Out[37]:

```
[Text(0.5, 0, 'Quality '),
 Text(0, 0.5, 'Sulphates '),
 Text(0.5, 1.0, 'Sulphates %  in different samples')]
```

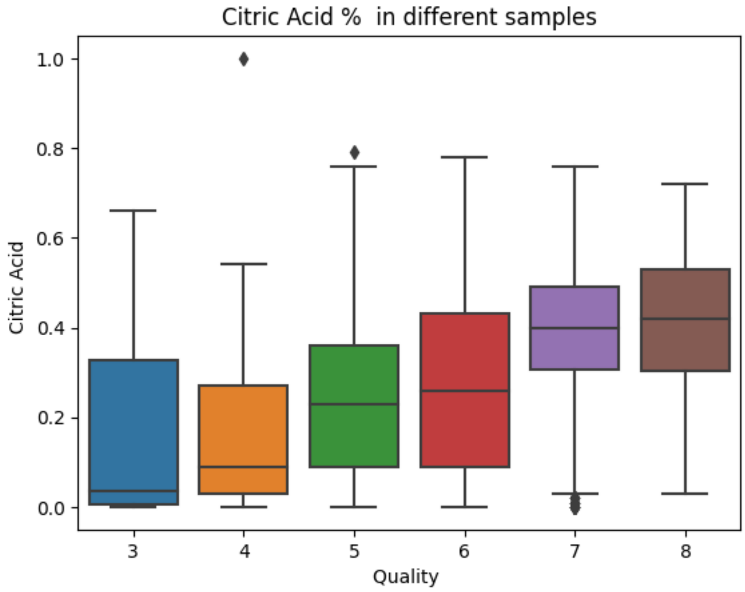


In [39]:

```
bx = sns.boxplot(x='quality', y='citric acid', data = dataset)
bx.set(xlabel='Quality ', ylabel='Citric Acid ', title='Citric Acid %  in different samples')
```

Out[39]:

```
[Text(0.5, 0, 'Quality '),
 Text(0, 0.5, 'Citric Acid '),
 Text(0.5, 1.0, 'Citric Acid %  in different samples')]
```

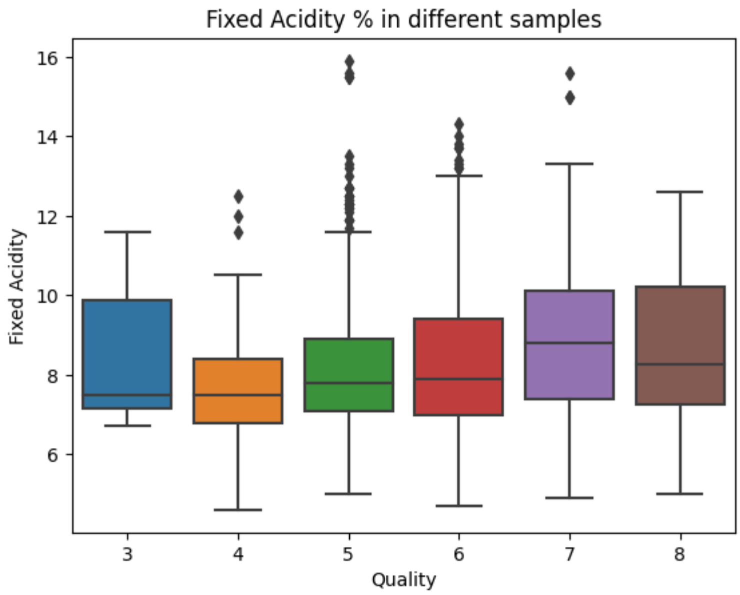


In [40]:

```
bx = sns.boxplot(x='quality', y='fixed acidity', data = dataset)
bx.set(xlabel='Quality', ylabel='Fixed Acidity', title='Fixed Acidity % in different samples')
```

Out[40]:

```
[Text(0.5, 0, 'Quality'),
 Text(0, 0.5, 'Fixed Acidity'),
 Text(0.5, 1.0, 'Fixed Acidity % in different samples')]
```



In [41]:

```
x
```

Out[41]:

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

1179 rows × 9 columns

In [42]:

```
y
```

Out[42]:

```
0       5
1       5
2       5
3       6
4       5
       ..
1594    5
1595    6
1596    6
1597    5
1598    6
Name: quality, Length: 1179, dtype: int64
```

In [43]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=3)
```

In [44]:

```
# x_train.shape
# x_test.shape
# y_train.shape
y_test.shape
```

Out[44]:

```
(354,)
```

In [45]:

```
# fitting linear regression to training data
regressor = LinearRegression()
regressor.fit(x_train,y_train)
```

Out[45]:

```
▾ LinearRegression
LinearRegression()
```

In [46]:

```
#To retrieve the intercept
regressor.intercept_
```

Out[46]:

```
27.670573873546417
```

In [48]:

```
# this gives the coefficients of the 10 features selected above.
regressor.coef_
```

Out[48]:

```
array([ 4.22974782e-02, -8.16827531e-01, -4.00865196e-01, -2.68428276e+00,
       -1.47339257e-03, -2.37486638e+01, -4.72842021e-01,  1.71236742e+00,
        2.47526682e-01])
```

In [49]:

```
train_pred = regressor.predict(x_train)
train_pred
```

Out[49]:

```
array([6.14356299, 5.11710037, 5.21197237, 5.13828062, 5.97949972,
       5.66562893, 5.4777587 , 5.75868703, 5.98907913, 5.40401462,
       5.52303708, 5.21113234, 5.38046811, 5.75877598, 5.35007708,
       5.08567146, 5.70418446, 6.15016457, 4.98217495, 6.37902248,
       5.34435775, 5.58388766, 5.56975986, 6.5429133 , 5.95905468,
       5.36649122, 5.24598625, 5.58550515, 5.18791293, 5.25072061,
       5.10187748, 5.00442024, 5.69182774, 5.89415555, 5.21543362,
       5.72691046, 5.08042222, 5.16537087, 6.26665775, 5.11379649,
       4.84031354, 5.32908031, 6.59578316, 5.9574155 , 5.17612261,
       5.52155991, 5.08413929, 6.1392644 , 5.48990749, 5.93825753,
       6.23616917, 5.92388793, 5.7786765 , 6.0650639 , 5.79356716,
       5.78930793, 6.0279377 , 4.86136512, 6.06957539, 5.1960625 ,
       5.82623979, 5.21010511, 5.18855806, 5.17190517, 5.06530766,
       5.2522647 , 5.64833165, 5.66231692, 5.54553416, 5.89096209,
       5.29556643, 5.10200981, 5.02472467, 5.47288678, 5.45596721,
       6.29709987, 5.76623284, 5.26529395, 5.64531976, 5.61024562,
       5.93475858, 5.87726527, 5.91779798, 5.45342902, 6.52865604,
       4.95957019, 5.76606249, 5.04086682, 5.79175917, 5.0713891 ,
```

In [50]:

```
test_pred = regressor.predict(x_test)
test_pred
```

Out[50]:

```
array([5.31808602, 5.58846727, 5.83179258, 5.23562426, 6.36492755,
       5.75166188, 5.61511554, 6.51307801, 6.033911  , 5.66126467,
       5.15680921, 5.48432811, 5.53204251, 5.17612261, 5.98484046,
       5.76958525, 6.09867422, 5.24902132, 5.45163284, 5.31035025,
       5.09350311, 5.87828479, 6.40866401, 5.412199  , 5.96442862,
       5.64014045, 5.51992784, 5.13588457, 6.28333602, 5.24519459,
       5.0320614 , 5.27962193, 5.59753018, 5.48395895, 5.58964467,
       6.0845468 , 5.19985585, 6.19604141, 5.34136276, 5.46949893,
       5.52658067, 5.96992765, 5.69237733, 6.52259415, 5.39271847,
       5.25392748, 5.99084808, 5.47407662, 5.49640697, 5.43513813,
       6.32835806, 6.16672701, 6.20060859, 5.78978599, 5.70708754,
       5.27350261, 5.36347142, 5.35513893, 6.26073939, 5.39379095,
       5.13365707, 5.39539395, 5.31604688, 5.55738131, 5.35711922,
       5.41823198, 5.04290802, 5.63751858, 5.05701887, 5.37699259,
       5.4916961 , 6.49050987, 5.67902012, 5.58946844, 5.624972  ,
       5.61792591, 5.86986021, 5.21037668, 5.78468929, 5.26398956,
       5.72691046, 5.40092383, 6.69197151, 5.64531976, 5.56034568,
       5.43467889, 5.55686222, 6.0233058 , 6.43578849, 5.52513157,
       5.67902012, 4.64898744, 6.14662984, 6.51674285, 5.97318723,
       5.49719429, 5.46490347, 6.16584155, 5.94581464, 5.07746641,
       5.2441662 , 5.78930793, 5.75304305, 5.10426467, 5.1827166 ,
       5.31511321, 5.99987354, 5.68526126, 5.07037239, 4.94121606,
       6.52865604, 5.74004727, 6.28794889, 5.43117363, 5.71550177,
       5.81784698, 4.9303826 , 5.24038113, 5.09211785, 5.14563991,
       5.26449292, 6.57023719, 6.70915507, 5.41197844, 6.34009483,
       6.4260368 , 6.30187789, 5.71188375, 5.91280815, 5.82027154,
       5.27651711, 5.74529965, 5.84539052, 5.22809651, 5.66670316,
       5.5902705 , 5.01881644, 6.21672901, 5.47214183, 5.47596665,
       5.94786716, 6.63668959, 6.27596776, 5.91709308, 5.1588646 ,
       5.47042012, 6.45247034, 5.0713891 , 6.0862025 , 6.03998607,
       5.25484004, 5.23470007, 5.55305246, 5.53467428, 5.04086682,
       5.16716583, 5.48101184, 6.22269596, 5.50728207, 5.10962883,
       5.58846727, 6.21149416, 6.11221593, 4.94675827, 5.84251347,
       5.64450235, 5.4419686 , 5.08727946, 6.01457484, 5.92755667,
       5.16288998, 5.88359672, 5.37380208, 5.05286101, 5.32431777,
       5.6439566 , 5.46490347, 6.56429012, 5.74004727, 6.24090367,
       5.74724044, 4.92431392, 5.53386144, 5.20778505, 4.86136512,
       5.43460576, 6.30967672, 5.31790065, 5.25163527, 6.05613027,
       5.84035633, 5.08784791, 6.15870704, 6.12754243, 5.52607036,
       4.97187315, 5.25359412, 6.70790937, 6.28794889, 5.13236485,
       5.32725711, 6.28448589, 5.45923718, 5.73101634, 5.69440787,
       5.65949152, 5.84565418, 5.01329309, 5.23624174, 6.23691057,
       5.55686222, 5.04799289, 5.29971201, 5.79794396, 5.30597556,
       5.40898341, 6.10600355, 5.77233753, 5.93825753, 5.10830136,
       5.2400021 , 5.35766108, 6.04499105, 5.76822909, 5.76958525,
       5.99354945, 5.61024562, 5.06244794, 5.47042012, 4.93392193,
       5.28715373, 5.22670922, 5.53684889, 5.53208489, 5.95021487,
       5.40558751, 5.19161171, 5.47596665, 5.11794805, 5.0981402 ,
       5.21451607, 5.70435419, 5.6116368 , 4.99467008, 6.46094847,
       6.65968339, 5.36928711, 6.16584155, 5.38024253, 5.38068717,
       6.09599831, 5.31294223, 5.43759048, 5.611229  , 6.29147929,
       5.41823198, 6.02194676, 5.37832235, 5.80309593, 5.47680605,
       5.38908021, 5.24902132, 5.04068696, 5.69516629, 5.86000764,
       5.40120967, 5.3834551 , 6.23387007, 6.0375956 , 6.67075119,
       5.10200981, 5.42506762, 5.80924931, 5.70588308, 6.05499602,
       4.97975135, 5.05000333, 4.95080592, 5.55137988, 5.60201573,
       5.51449882, 6.32041445, 4.98861147, 5.82834176, 5.37670742,
       5.20026968, 6.53073594, 4.84246122, 5.27651711, 5.10819397,
       5.92311671, 5.0143682 , 5.58802138, 6.20016963, 5.50501381,
       4.89930303, 5.18962732, 5.55263282, 5.58964467, 6.17709158,
       5.55700084, 5.39046152, 6.52219068, 5.67196235, 5.05000333,
       5.08079148, 5.5943058 , 6.0929811 , 5.83125641, 5.11413388,
       5.39621839, 5.35003757, 5.53339833, 5.17331786, 6.1812361 ,
       6.66364432, 6.08288344, 5.76263575, 6.46721371, 6.16370399,
       5.30301361, 5.80309593, 5.38956387, 5.31619663, 5.56898153,
       5.13831066, 5.45929077, 6.36179741, 5.78350828, 4.8024992 ,
       4.93392193, 6.15635513, 5.63443962, 5.78846641, 5.38404606,
       5.24038113, 5.72478184, 5.03161613, 6.06088742, 5.70624392,
       6.02392175, 6.31342769, 6.19084804, 5.18021277, 5.09496384,
       5.3259214 , 6.26077414, 5.39835093, 6.09093032, 5.27630308,
       5.08727946, 5.58886863, 6.12677789, 6.33604641])
```

In [51]:

```
train_rmse = metrics.mean_squared_error(train_pred, y_train) ** 0.5
train_rmse
```

Out[51]:

```
0.5716073011886497
```

In [52]:

```
test_rmse = metrics.mean_squared_error(test_pred, y_test) ** 0.5
test_rmse
```

Out[52]:

```
0.5670861234986311
```

In [53]:

```
# rounding off the predicted values for test set
predicted_data = np.round_(test_pred)
predicted_data
```

Out[53]:

```
array([5., 6., 6., 5., 6., 6., 6., 7., 6., 6., 5., 5., 6., 5., 6., 6., 6.,
       5., 5., 5., 5., 6., 6., 5., 6., 6., 6., 5., 6., 5., 5., 5., 6., 5.,
       6., 6., 5., 6., 5., 5., 6., 6., 6., 7., 5., 5., 6., 5., 5., 5., 6.,
       6., 6., 6., 6., 5., 5., 5., 6., 5., 5., 5., 5., 6., 5., 5., 5., 6.,
       5., 5., 5., 6., 6., 6., 6., 6., 6., 5., 6., 5., 6., 5., 7., 6., 6.,
       5., 6., 6., 6., 6., 6., 5., 6., 7., 6., 5., 5., 6., 6., 5., 5., 6.,
       6., 5., 5., 5., 6., 6., 5., 5., 7., 6., 6., 5., 6., 6., 5., 5., 5.,
       5., 5., 7., 7., 5., 6., 6., 6., 6., 6., 6., 5., 6., 6., 5., 6., 6.,
       5., 6., 5., 5., 6., 7., 6., 6., 5., 5., 6., 5., 6., 6., 5., 5., 6.,
       6., 5., 5., 5., 6., 6., 5., 6., 6., 6., 5., 6., 6., 5., 5., 6., 6.,
       5., 6., 5., 5., 5., 6., 5., 7., 6., 6., 6., 5., 6., 5., 5., 5., 6.,
       5., 5., 6., 6., 5., 6., 6., 6., 5., 5., 7., 6., 5., 5., 6., 5., 6.,
       6., 6., 6., 5., 5., 6., 6., 5., 5., 6., 5., 5., 6., 6., 6., 5., 5.,
       5., 6., 6., 6., 6., 6., 5., 5., 5., 5., 5., 6., 6., 6., 5., 5., 5.,
       5., 5., 5., 6., 6., 5., 6., 7., 5., 6., 5., 5., 6., 5., 5., 6., 6.,
       5., 6., 5., 6., 5., 5., 5., 5., 6., 6., 5., 5., 6., 6., 7., 5., 5.,
       6., 6., 6., 5., 5., 5., 6., 6., 6., 6., 5., 6., 5., 5., 7., 5., 5.,
       5., 6., 5., 6., 6., 6., 5., 5., 6., 6., 6., 6., 5., 7., 6., 5., 5.,
       6., 6., 6., 5., 5., 5., 6., 5., 6., 7., 6., 6., 6., 6., 5., 6., 5.,
       5., 6., 5., 5., 6., 6., 5., 5., 6., 6., 6., 5., 5., 6., 5., 6., 6.,
       6., 6., 6., 5., 5., 5., 6., 5., 6., 5., 5., 6., 6., 6.])
```

In [54]:

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, test_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, test_pred))
rmse = np.sqrt(metrics.mean_squared_error(y_test, test_pred))
print('Root Mean Squared Error:',rmse)
```

```
Mean Absolute Error: 0.4566775059037429
Mean Squared Error: 0.3215866714647046
Root Mean Squared Error: 0.5670861234986311
```

In [55]:

```
from sklearn.metrics import r2_score
r2_score(y_test,test_pred)
```

Out[55]:

```
0.4070484025414417
```

In [56]:

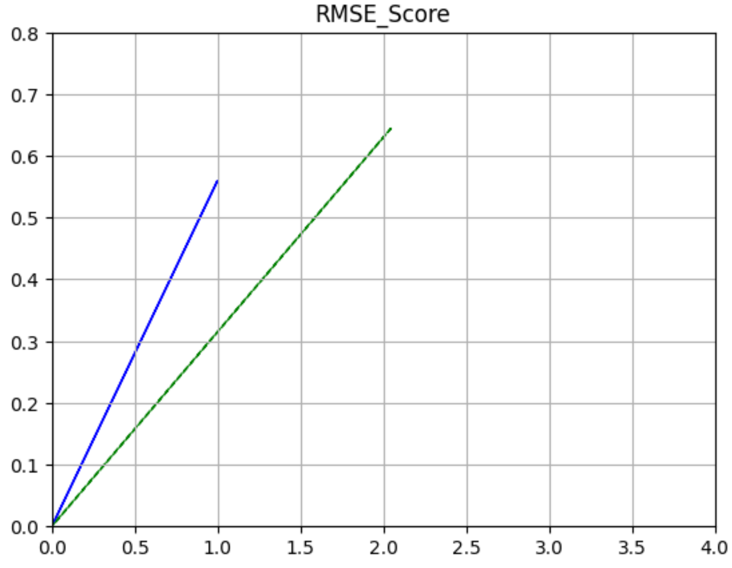
```
ame(regressor.coef_,features)
'Coeffecient']

 holding all other features fixed, a 1 unit increase in suplhates will lead to an increase of 0.8 in Quality of wine, and similarly for th
 holding all other features fixed, a 1 unit increase in volatile acidity will lead to a decrease of 0.99 in Quality of wine, and
```

Out[56]:

	Coeffecient
fixed acidity	0.042297
volatile acidity	-0.816828
citric acid	-0.400865
chlorides	-2.684283
total sulfur dioxide	-0.001473
density	-23.748664
pH	-0.472842
sulphates	1.712367
alcohol	0.247527

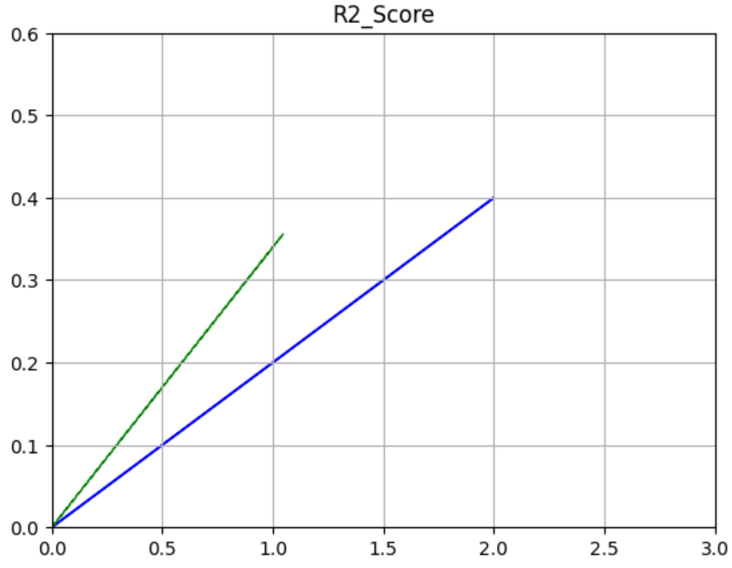
In [57]:

```
import matplotlib.pyplot as plt1
ax=plt1.axes()
color1= 'green'
color2= 'blue'
ax.arrow(0,0,1,0.56,head_width=0.00,head_length=0,fc=color2,ec=color2)
ax.arrow(0,0,2,0.63,head_width=0.00,head_length=0.05,fc=color1,ec=color1,linestyle='--')
ax.set_ylim([0,0.8])
ax.set_xlim([0,4])
plt.grid()
plt.title('RMSE_Score')

plt1.show()
```



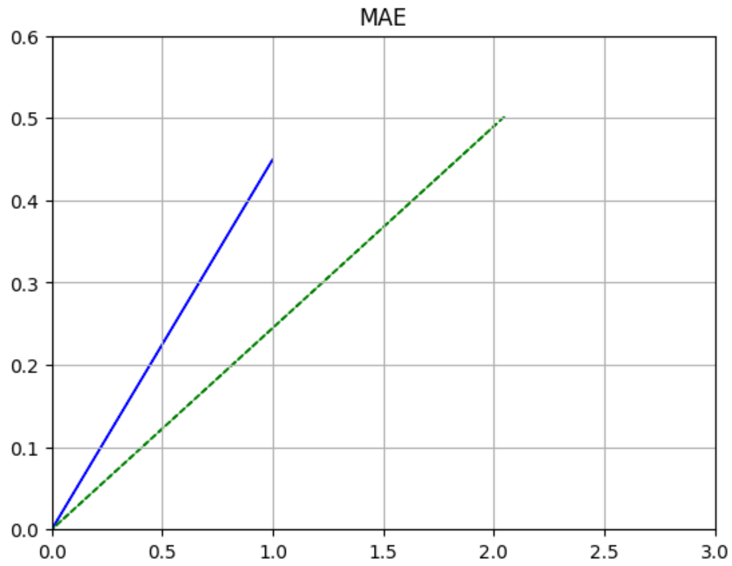
In [58]:

```
import matplotlib.pyplot as plt1
ax=plt1.axes()
color1= 'green'
color2= 'blue'
ax.arrow(0,0,2,0.40,head_width=0.00,head_length=0,fc=color2,ec=color2)
ax.arrow(0,0,1,0.34,head_width=0.00,head_length=0.05,fc=color1,ec=color1,linestyle='--')
ax.set_ylim([0,0.6])
ax.set_xlim([0,3])
plt.grid()
plt.title('R2_Score')

plt1.show()
```



In [59]:

```
import matplotlib.pyplot as plt1
ax=plt1.axes()
color1= 'green'
color2= 'blue'
ax.arrow(0,0,1,0.45,head_width=0.00,head_length=0,fc=color2,ec=color2)
ax.arrow(0,0,2,0.49,head_width=0.00,head_length=0.05,fc=color1,ec=color1,linestyle='--')
ax.set_ylim([0,0.6])
ax.set_xlim([0,3])
plt.grid()
plt.title('MAE')

plt1.show()
```



Conclusion

We have created a Linear Regression Model which we help to Analysis the Wine quality prediction.

End of the code

Thank You