In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv('C:/Users/Raghavendra K/Downloads/winequality-red.csv')
```

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

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

In [4]:
```
df.shape
```

Out[4]:
```
(1599, 12)
```

In [5]:
```
df.tail()
```

Out[5]:

	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 [6]:
```
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 [7]:
```
df.isnull().sum()
```

Out[7]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

In [8]:
```
df['quality'].unique()
```

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

In [9]:
```
df.describe()
```

Out[9]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	

In [10]:
```
df.columns
```

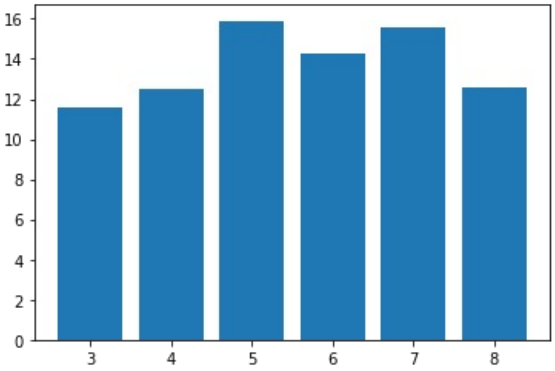
Out[10]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [11]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [12]:
```
plt.bar(df['quality'],df['fixed acidity'])
plt.xlable('quality')
plt.ylable('fixed acidity')
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [12], in <cell line: 2>()
      1 plt.bar(df['quality'],df['fixed acidity'])
----> 2 plt.xlable('quality')
      3 plt.ylable('fixed acidity')
      4 plt.show()

AttributeError: module 'matplotlib.pyplot' has no attribute 'xlable'
```



In [ ]:
```
plt.bar(df['quality'],df['volatile acidity'])
plt.xlable('quality')
plt.ylable('volatile acidity')
plt.show()
```

In [ ]:
```
plt.bar(df['quality'],df[ 'residual sugar'])
plt.xlable('quality')
plt.ylable( 'residual sugar')
plt.show()
```

In [ ]:
```
plt.bar(df['quality'],df['chlorides'])
plt.xlable('quality')
plt.ylable('chlorides')
plt.show()
```

In [ ]:
```
plt.bar(df['quality'],df['total sulfur dioxide'])
plt.xlable('quality')
plt.ylable('total sulfur dioxide')
plt.show()
```

In [ ]:
```
plt.bar(df['quality'],df['alcohol'])
plt.xlable('quality')
plt.ylable('alcohol')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(10,5))
sns.heatmap(df.corr(),annot=True,fmt='0.1f')
```

In [ ]:
```
#Binarization of target variable
df['quality'].unique()
```

In [ ]:
```
df['quality']=[1 if x>=7 else 0 for x in df['quality']]
```

In [ ]:
```
df['quality'].unique()
```

In [ ]:
```
df['quality'].value_counts()
```

In [ ]:
```
import seaborn as sns
```

In [ ]:
```
sns.countplot(df['quality'])
```

In [ ]:
```
#Handling Imbalanced dataset-smote techniques
from imblearn.over_sampling import SMOTE
```

In [ ]:
```
from imblearn.over_sampling import SMOTE
```

In [ ]:
```

```

In [ ]:
```
x_res,y_res = SMOTE().fit_resample(x,y)
```

In [ ]:
```
y_res.value_counts()
```

In [ ]:
```
x = df.drop('quality',axis=1)
y = df['quality']
```

In [ ]:
```

```

In [ ]:
```
x
```

In [ ]:
```
y
```

In [ ]:
```
#Split the dataset into train and test
from sklearn.model_selection import train_test_split
```

In [ ]:
```
x_train,x_test,y_train,y_test=train_test_split(x_res,y_res,test_size=0.20,random_state=42)
```

In [ ]:
```
#FEATURE SCALING
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
st = StandardScaler()
x_train = st.fit_transform(x_train)
x_test = st.transform(x_test)
```

In [ ]:
```
x_train
```

APPLYING - PRINCIPAL COMPONENT ANALYSIS-PCA

In [ ]:
```
from sklearn.decomposition import PCA
```

In [ ]:
```
pca = PCA(n_components=0.90)
```

In [ ]:
```
x_train = pca.fit_transform(x_train)
x_test = pca.transform(x_test)
```

In [ ]:
```
pca.explained_variance_ratio_
```

In [ ]:
```
sum(pca.explained_variance_ratio_)
```

In [ ]:
```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
```

In [ ]:
```
log = LogisticRegression()
log.fit(x_train,y_train)
```

In [ ]:
```
y_pred1 = log.predict(x_test)
```

In [ ]:
```
from sklearn.metrics import accuracy_score
```

In [ ]:
```
accuracy_score(y_test,y_pred1)
```

In [ ]:
```
from sklearn.metrics import precision_score,recall_score,f1_score
```

In [ ]:
```
precision_score(y_test,y_pred1)
```

In [ ]:
```
recall_score(y_test,y_pred1)
```

In [ ]:
```
f1_score(y_test,y_pred1)
```

In [ ]:
```
#SUPPORT VECTOR CLASSIFIER-SVC
from sklearn import svm
```

In [ ]:
```
svm=svm.SVC()
```

In [ ]:
```
svm.fit(x_train,y_train)
```

In [ ]:
```
y_pred2=svm.predict(x_test)
```

In [ ]:
```
accuracy_score(y_test,y_pred2)
```

In [ ]:
```
precision_score(y_test,y_pred2)
```

In [ ]:
```
f1_score(y_test,y_pred2)
```

In [ ]:
```
#KNeighbors Classifier
from sklearn.neighbors import KNeighborsClassifier
```

In [ ]:
```
knn = KNeighborsClassifier()
```

In [ ]:
```
knn.fit(x_train,y_train)
```

In [ ]:
```
y_pred3 = knn.predict(x_test)
```

In [ ]:
```
accuracy_score(y_test,y_pred3)
```

In [ ]:
```
precision_score(y_test,y_pred3)
```

In [ ]:
```
f1_score(y_test,y_pred3)
```

In [ ]:
```
recall_score(y_test,y_pred3)
```

In [ ]:
```
#Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:
```
dt = DecisionTreeClassifier()
```

In [ ]:
```
dt.fit(x_train,y_train)
```

In [ ]:
```
y_pred4=dt.predict(x_test)
```

In [ ]:
```
accuracy_score(y_test,y_pred4)
```

In [ ]:
```
precision_score(y_test,y_pred4)
```

In [ ]:
```
f1_score(y_test,y_pred4)
```

In [ ]:
```
#Random Forest Classifier

form sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
rf = RandomForestClassifier()
rf.fit(x_train,y_train)
```

In [ ]:
```
y_pred5=rf.predict(x_test)
```

In [ ]:
```
accuracy_score(y_test,y_pred5)
```

In [ ]:
```
precision_score(y_test,y_pred5)
```

In [ ]:
```
f1_score(y_test,y_pred5)
```

In [ ]:
```
recall_score(y_test,y_pred5)
```

In [ ]:
```
#Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier
```

In [ ]:
```
gbc = GradientBoostingClassifier()
gbc.fit(x_train,y_train)
```

In [ ]:
```
y_pred6=gbc.predict(x_test)
```

In [ ]:
```
accuracy_score(y_test,y_pred6)
```

In [ ]:
```
precision_score(y_test,y_pred6)
```

In [ ]:
```
f1_score(y_test,y_pred6)
```

In [ ]:
```
import pandas as pd
```

In [ ]:
```
final_data = pd.DataFrame({'Models':['LR','SVC','KNN','DT','RF','GBC'],'ACC':[accuracy_score(y_test,y_pred1)*10
                                                                              accuracy_score(y_test,y_pred2)*100,
                                                                              accuracy_score(y_test,y_pred3)*100,
                                                                              accuracy_score(y_test,y_pred4)*100,
                                                                              accuracy_score(y_test,y_pred5)*100,
                                                                              accuracy_score(y_test,y_pred6)*100]})
```

final_data

In [ ]:
```
import seaborn as sns
```

In [ ]:
```
final_data
```

In [ ]:
```

```

In [ ]:
```
sns.barplot(final_data['Models'],final_data['ACC'])
```

SAVE THE MODEL

In [ ]:
```
x=df.drop('quality',axis=1)
y=df['quality']
```

In [ ]:
```
from imblearn.over_sampling import SMOTE
x_res,y_res=SMOTE().fit_resample(x,y)
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
st = StandardScaler()
x=st.fit_transform(x_res)
```

In [ ]:
```
x=pca.fit_transform(x)
```

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(x,y_res)
```

In [ ]:
```
import joblib
```

In [ ]:
```
 joblib.dump(rf,'quality_prediction_model')
```

In [ ]:
```
 model = joblib.load('quality_prediction_model')
```

model = joblib.dump(rf,'quality_prediction_model')

PREDICTION ON NEW DATA

In [ ]:
```
import pandas as pd
new_data = pd.DataFrame({
    'fixed acidity':7.3,
    'volatile acidity':0.65,
    'citric acid':0.00,
    'residual sugar':1.2,
    'chlorides':0.065,
    'free sulfur dioxide':15.0,
    'total sulfur dioxide':21.0,
    'density':0.9946,
    'PH':3.39,
    'sulphate':0.47,
    'alcohol':10.0,

},index=[0])
```

In [ ]:
```
new_data
```

In [ ]:
```
test = pca.transform(st.transform(new_data))
```

In [ ]:
```
model.predict(test)
```

In [ ]:
```
p = model.predict(test)
```

In [ ]:
```
if p[0]==1:
    print("Good Quality Wine")
else:
    print("Bad Quality Wine")
```

GUI

In [ ]:
```
from tkinter import *

from sklearn.preprocessing import StandardScaler
import joblib
```

In [ ]:
```
def show_entry_fields():
    p1=float(e1.get())
    p2=float(e2.get())
    p3=float(e3.get())
    p4=float(e4.get())
    p5=float(e5.get())
    p6=float(e6.get())
    p7=float(e7.get())
    p8=float(e8.get())
    p9=float(e9.get())
    p10=float(e10.get())
    p11=float(e11.get())
    model = joblib.load('quality_prediction_model')
    result=model.predict(pca.transform(st.transform([[p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11]])))

    if p[0]==1:
        Label(master,text="quality_prediction_model").grid(row=31)
        Label(master, text="quality_prediction_model").grid(row=31)

master =Tk()
master.title("quality_prdiction_model")
label = Label(master,text = "quality_prediction_model"
              ,bg  = "black", fg = "white" ).\
               grid(row=0,columnspan=2)

Label(master, text='fixed acidity').grid(row=1)
Label(master, text= 'volatile acidity').grid(row=2)
Label(master, text='citric acid').grid(row=3)
Label(master, text='residual sugar').grid(row=4)
Label(master, text='chlorides').grid(row=5)
Label(master, text= 'free sulfur dioxide').grid(row=6)
Label(master, text='total sulfur dioxide').grid(row=7)
Label(master, text='density').grid(row=8)
Label(master, text='PH').grid(row=9)
Label(master, text='sulphate').grid(row=10)
Label(master, text= 'alcohol').grid(row=11)

e1=Entry(master)
e2=Entry(master)
e3=Entry(master)
e4=Entry(master)
e5=Entry(master)
e6=Entry(master)
e7=Entry(master)
e8=Entry(master)
e9=Entry(master)
e10=Entry(master)
e11=Entry(master)

e1.grid(row=1, column=1)
e2.grid(row=2, column=1)
e3.grid(row=3, column=1)
e4.grid(row=4, column=1)
e5.grid(row=5, column=1)
e6.grid(row=6, column=1)
e7.grid(row=7, column=1)
e8.grid(row=8, column=1)
e9.grid(row=9, column=1)
e10.grid(row=10, column=1)
e11.grid(row=11, column=1)

Button(master, text='predict', command=show_entry_fields).grid()

mainloop()
```

In [ ]:
```

```

In [ ]:

In [ ]: