Assignment No.3

Title:Create a multiclass classification model to predict wine quality based on chemical properties.

Dataset: Wine Quality Dataset (UCI)

Name: Vaishnav Kalidas Temgire

Roll No:23107127

Class : TY-B

In [1]:

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

In [3]:

```
df = pd.read_csv("C:/Users/prath/Downloads/WineQT.csv")
```

In [4]:

```
df
```

Out[4]:

	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	0.99780	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	
...	...	...	...	...	...	...	...	...	...	...	
1138	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	
1139	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	
1140	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	
1141	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	
1142	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	

1143 rows × 13 columns

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
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
Id                      0
dtype: int64
```

In [6]:

```
df.isna().sum()
```

Out[6]:

```
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
Id                      0
dtype: int64
```

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64  
 12  Id                    1143 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
```

In [8]:

```
df.describe
```

Out[8]:

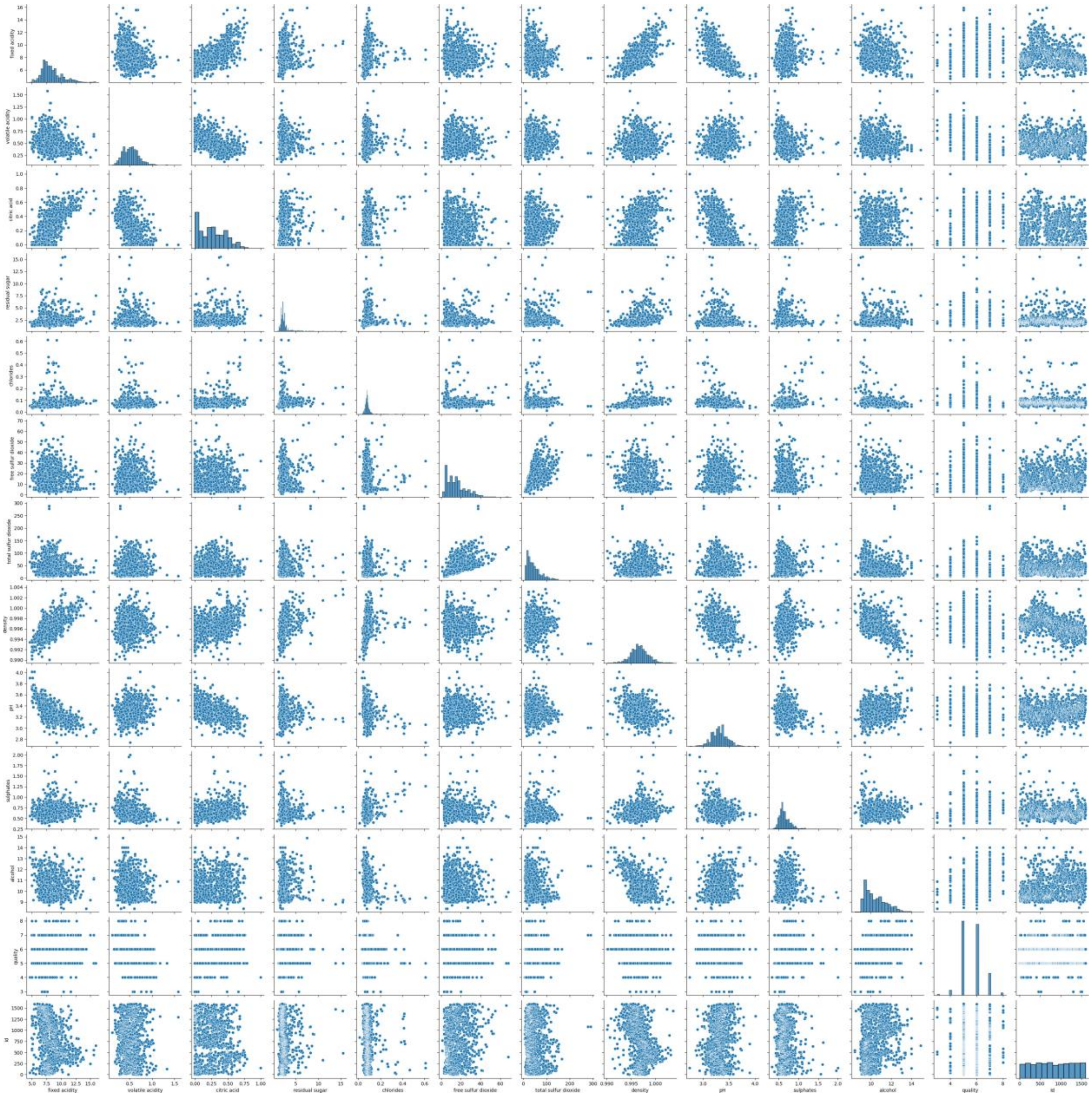
```
<bound method NDFrame.describe of       fixed acidity  volatile acidity  ci
tric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.0
76
1               7.8             0.880         0.00             2.6      0.0
98
2               7.8             0.760         0.04             2.3      0.0
92
3              11.2             0.280         0.56             1.9      0.0
75
4               7.4             0.700         0.00             1.9      0.0
76
...             ...               ...          ...             ...
...
1138            6.3             0.510         0.13             2.3      0.0
76
1139            6.8             0.620         0.08             1.9      0.0
68
1140            6.2             0.600         0.08             2.0      0.0
90
1141            5.9             0.550         0.10             2.2      0.0
62
1142            5.9             0.645         0.12             2.0      0.0
75

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates
\
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
...                   ...                   ...      ...   ...        ...
1138                 29.0                  40.0  0.99574  3.42       0.75
1139                 28.0                  38.0  0.99651  3.42       0.82
1140                 32.0                  44.0  0.99490  3.45       0.58
1141                 39.0                  51.0  0.99512  3.52       0.76
1142                 32.0                  44.0  0.99547  3.57       0.71

      alcohol  quality    Id
0         9.4        5     0
1         9.8        5     1
2         9.8        5     2
3         9.8        6     3
4         9.4        5     4
...       ...      ...   ...
1138     11.0        6  1592
1139      9.5        6  1593
1140     10.5        5  1594
1141     11.2        6  1595
1142     10.2        5  1597

[1143 rows x 13 columns]>
```

In [9]:

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

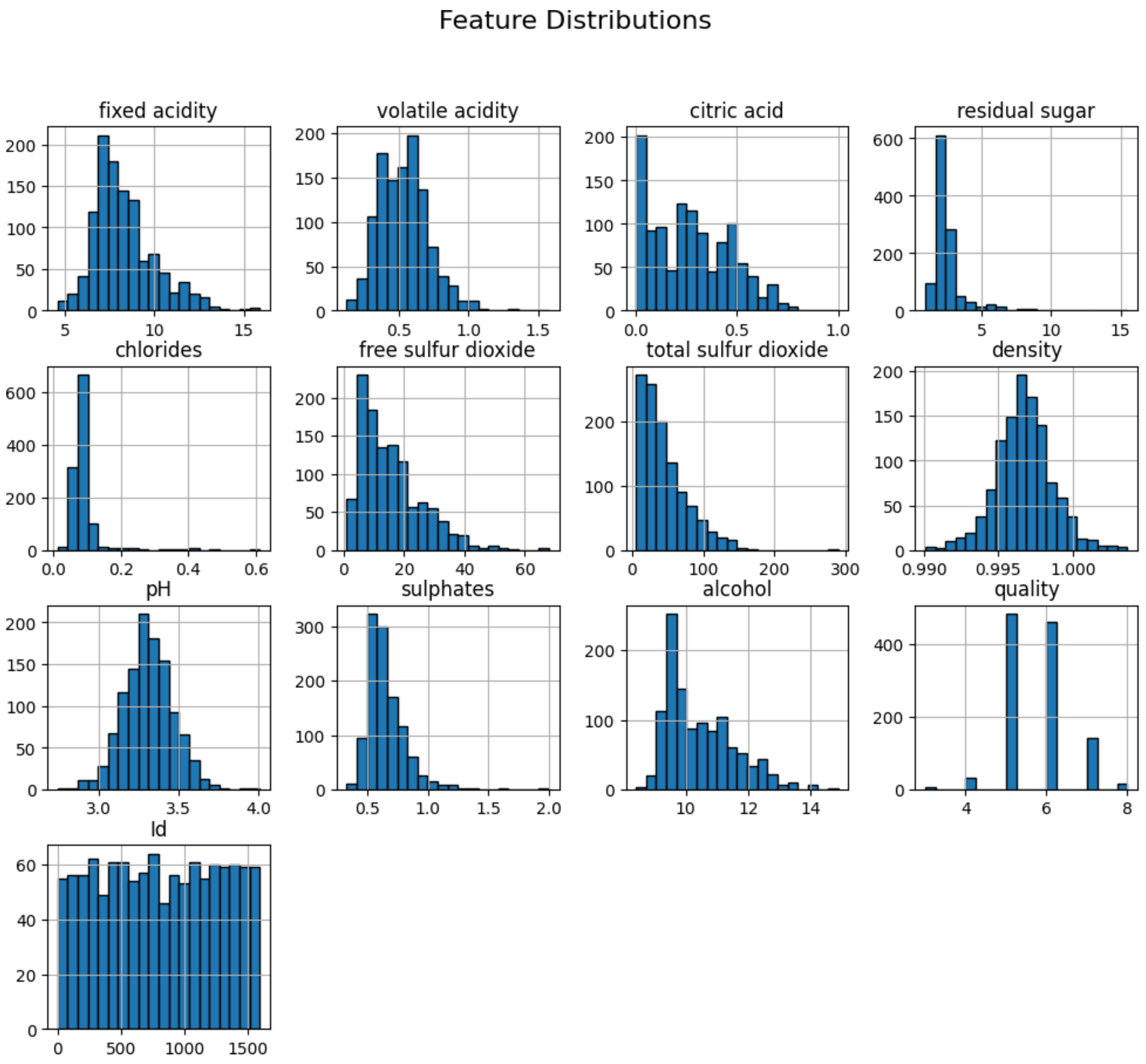
Out[9]:

```
<seaborn.axisgrid.PairGrid at 0x1b3d44435d0>
```



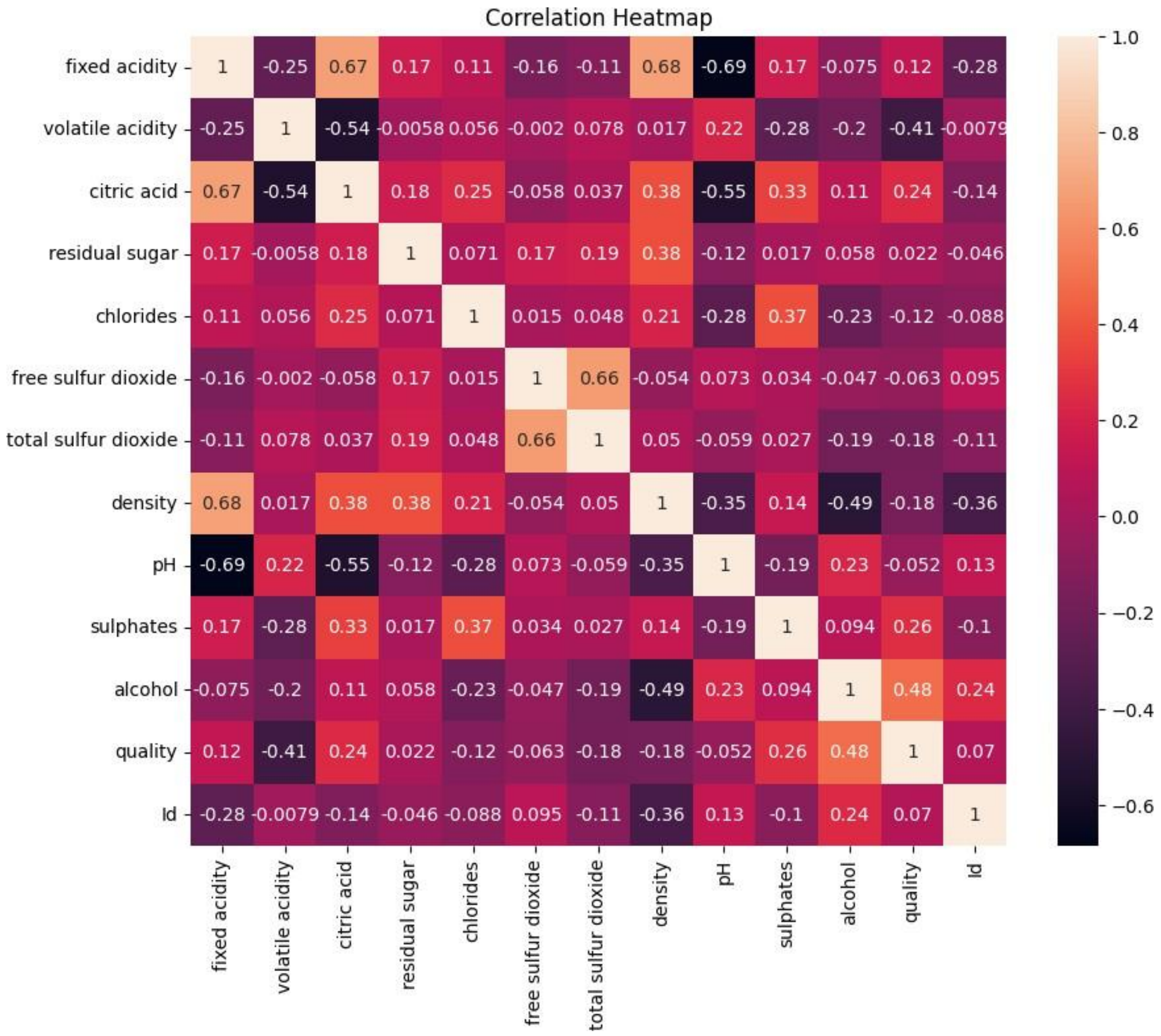
In [100…

```
df.hist(figsize=(12,10), bins=20, edgecolor="black")
plt.suptitle("Feature Distributions", fontsize=16)
plt.show()
```



In [106...

```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True)
plt.title("Correlation Heatmap")
plt.show()
```



In [10]:

```
## Dependent and Independent Variables
x = df.drop(columns=['quality','Id'],axis=1)
y= df['quality']
```

In [ ]:

```
## SMOTE For the Imbalance dataset

from imblearn.over_sampling import SMOTE
oversample = SMOTE()
x,y = oversample.fit_resample(x,df['quality'])
```

In [103...

```
y.value_counts()
```

Out[103]:

```
quality
5    483
6    483
7    483
4    483
8    483
3    483
Name: count, dtype: int64
```

In [105...

```
x.shape,y.shape
```

Out[105]:

```
((2898, 11), (2898,))
```

In [60]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test  =  train_test_split(x,y,train_size=0.8,random_
```

In [61]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [62]:
```
x_train_scaled = sc.fit_transform(x_train)
```

In [63]:
```
x_test_scaled = sc.transform(x_test)
```

In [119...
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
```

In [120...
```
model
```

Out[120]:

▾ DecisionTreeClassifier

▶ Parameters

In [121...
```
model.fit(x_train_scaled,y_train)
```

Out[121]:

▾ DecisionTreeClassifier

▶ Parameters

In [122...
```
y_pred = model.predict(x_test_scaled)
```

In [123...
```
y_pred
```

Out[123]:
```
array([8, 3, 5, 4, 7, 3, 7, 8, 5, 4, 6, 4, 3, 6, 3, 4, 6, 8, 8, 8, 7, 5,
       3, 6, 3, 8, 4, 5, 4, 3, 7, 4, 3, 5, 7, 7, 3, 4, 4, 4, 3, 3, 3, 3,
       8, 4, 8, 5, 7, 8, 4, 3, 8, 6, 4, 4, 3, 3, 3, 4, 3, 5, 4, 6, 6, 3,
       6, 7, 4, 5, 3, 5, 5, 3, 7, 6, 8, 5, 4, 5, 3, 8, 3, 4, 3, 8, 5, 4,
       7, 3, 7, 5, 5, 7, 3, 7, 4, 3, 7, 7, 6, 4, 3, 3, 4, 7, 7, 8, 4, 7,
       7, 5, 4, 6, 6, 7, 5, 7, 8, 4, 7, 4, 7, 6, 4, 4, 4, 3, 7, 4, 4, 8,
       4, 3, 8, 7, 6, 4, 8, 3, 8, 8, 4, 6, 4, 3, 6, 4, 8, 5, 4, 7, 6, 7,
       4, 7, 8, 4, 4, 6, 3, 7, 4, 4, 8, 8, 8, 5, 5, 8, 3, 5, 3, 3, 5, 4,
       8, 5, 7, 7, 8, 5, 3, 7, 5, 6, 7, 7, 8, 3, 4, 6, 3, 6, 6, 6, 8, 6,
       6, 6, 5, 7, 6, 4, 5, 3, 3, 5, 7, 6, 8, 8, 4, 8, 5, 5, 7, 4, 4, 5,
       3, 8, 8, 5, 3, 7, 8, 7, 3, 3, 3, 8, 8, 4, 3, 7, 6, 6, 7, 4, 8, 5,
       4, 6, 7, 7, 5, 8, 7, 4, 7, 5, 4, 6, 4, 6, 7, 5, 4, 5, 4, 4, 6, 6,
       8, 5, 6, 3, 7, 7, 3, 6, 3, 3, 6, 5, 7, 4, 6, 4, 7, 6, 7, 6, 8, 8,
       4, 4, 5, 5, 4, 4, 6, 5, 5, 4, 6, 3, 5, 8, 5, 5, 8, 6, 6, 8, 3, 4,
       8, 8, 7, 6, 7, 3, 6, 7, 3, 6, 5, 8, 3, 7, 6, 6, 3, 7, 3, 8, 7, 6,
       3, 5, 5, 8, 4, 8, 5, 3, 3, 3, 6, 3, 4, 5, 3, 3, 5, 5, 4, 5, 8, 6,
       7, 3, 4, 3, 7, 3, 5, 7, 6, 8, 7, 5, 5, 4, 6, 3, 7, 6, 4, 7, 8, 6,
       4, 5, 4, 5, 8, 7, 7, 8, 7, 7, 6, 7, 7, 4, 8, 8, 4, 4, 5, 5, 3, 5,
       5, 7, 4, 6, 7, 3, 8, 5, 5, 6, 6, 5, 6, 3, 4, 3, 6, 5, 4, 3, 7, 5,
       4, 7, 7, 4, 7, 6, 8, 7, 5, 7, 8, 8, 3, 6, 5, 8, 7, 8, 5, 7, 5, 4,
       8, 8, 5, 6, 6, 3, 8, 4, 5, 5, 7, 8, 5, 8, 5, 6, 3, 7, 6, 6, 7, 8,
       6, 8, 8, 8, 8, 7, 3, 5, 8, 6, 4, 5, 5, 6, 6, 4, 7, 7, 5, 6, 8, 6,
       6, 5, 4, 6, 8, 6, 8, 7, 8, 7, 3, 3, 4, 6, 8, 5, 5, 6, 4, 7, 6, 8,
       4, 8, 3, 3, 6, 3, 5, 6, 7, 8, 4, 3, 3, 6, 8, 6, 8, 5, 4, 5, 5, 4,
       4, 6, 8, 5, 8, 7, 8, 7, 6, 7, 8, 6, 8, 7, 3, 6, 8, 7, 5, 5, 8, 7,
       6, 6, 3, 7, 8, 3, 7, 5, 6, 7, 7, 7, 8, 5, 4, 7, 5, 8, 3, 4, 8, 3,
       4, 7, 3, 6, 6, 7, 3, 4], dtype=int64)
```

In [124...
```
from sklearn.metrics import classification_report,accuracy_score,precision_
cr   =classification_report(y_test,y_pred)
acc_dt = accuracy_score(y_test,y_pred)
re= recall_score(y_test,y_pred,average='micro')
```

```
f1  =  f1_score(y_test,y_pred,average='micro')
pr =precision_score(y_test,y_pred,average='micro')
```

In [125...

```
print(cr)
```

```
              precision    recall  f1-score   support

           3       0.95      0.91      0.93        97
           4       0.79      0.84      0.81        92
           5       0.68      0.59      0.63       108
           6       0.54      0.54      0.54        95
           7       0.79      0.84      0.81        96
           8       0.91      0.96      0.93        92

    accuracy                           0.77       580
   macro avg       0.77      0.78      0.78       580
weighted avg       0.77      0.77      0.77       580
```

In [126...

```
re
```

Out[126]:

```
0.7741379310344828
```

In [127...

```
pr
```

Out[127]:

```
0.7741379310344828
```

In [128...

```
acc_dt
```

Out[128]:

```
0.7741379310344828
```

In [129...

```
f1
```

Out[129]:

```
0.7741379310344828
```

In [130...

```
### Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rc  = RandomForestClassifier()
```

In [131...

```
rc
```

Out[131]:

```
▾ RandomForestClassifier  i ?
▶ Parameters
```

In [132...

```
rc.fit(x_train_scaled,y_train)
```

Out[132]:

```
▾ RandomForestClassifier  i ?
▶ Parameters
```

In [133...

```
y_pred  =  rc.predict(x_test_scaled)
```

In [134...

```
y_pred
```

Out[134]:

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

In [135...

```python
from sklearn.metrics import classification_report,accuracy_score,precision_
cr   =classification_report(y_test,y_pred)
acc_rf  =  accuracy_score(y_test,y_pred)
re=  recall_score(y_test,y_pred,average='micro')
f1  =  f1_score(y_test,y_pred,average='micro')
pr   =precision_score(y_test,y_pred,average='micro')
cm  =  confusion_matrix(y_test,y_pred)
```

In [136...

```python
print(cr)
```

```
              precision    recall  f1-score   support

           3       1.00      1.00      1.00        97
           4       0.91      0.97      0.94        92
           5       0.80      0.70      0.75       108
           6       0.62      0.58      0.60        95
           7       0.81      0.92      0.86        96
           8       0.99      1.00      0.99        92

    accuracy                           0.86       580
   macro avg       0.85      0.86      0.86       580
weighted avg       0.85      0.86      0.85       580
```

In [137...

```python
acc_rf
```

Out[137]:

```
0.8568965517241379
```

In [138...

```python
re
```

Out[138]:

```
0.8568965517241379
```

In [139...

```python
f1
```

Out[139]:

```
0.8568965517241379
```

In [140...

```
pr
```

Out[140]:

```
0.8568965517241379
```

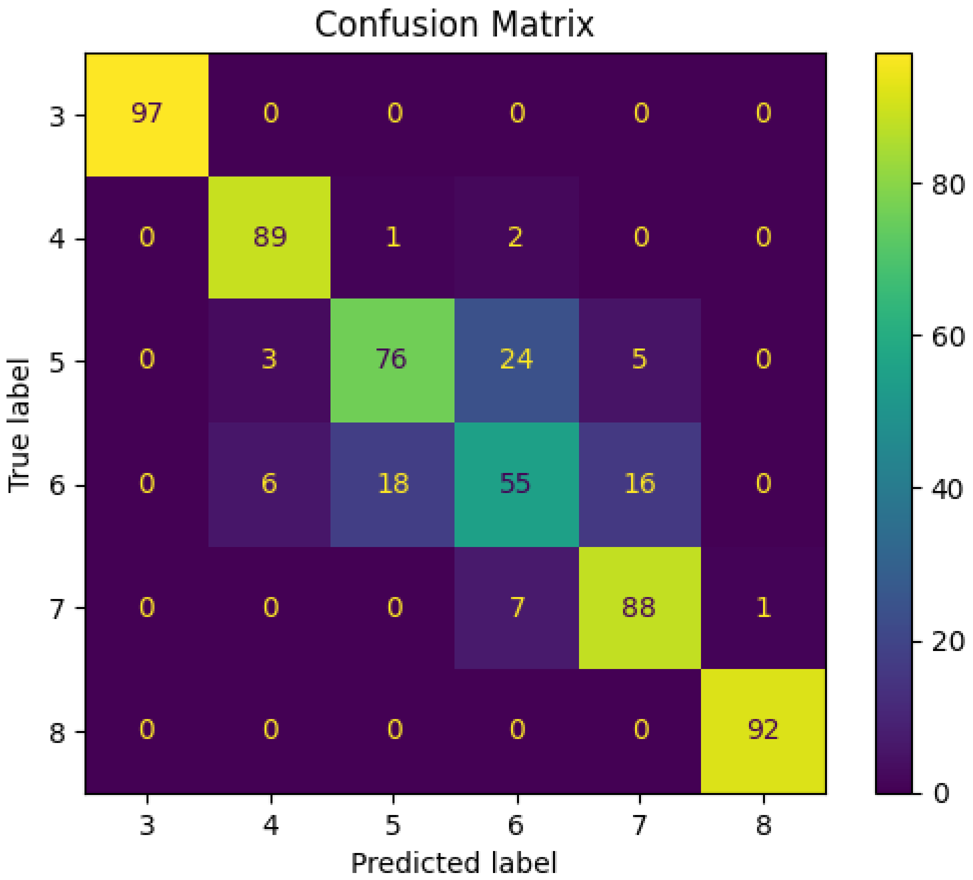
In [141...

```
print(cm)
```

```
[[97  0  0  0  0  0]
 [ 0 89  1  2  0  0]
 [ 0  3 76 24  5  0]
 [ 0  6 18 55 16  0]
 [ 0  0  0  7 88  1]
 [ 0  0  0  0  0 92]]
```

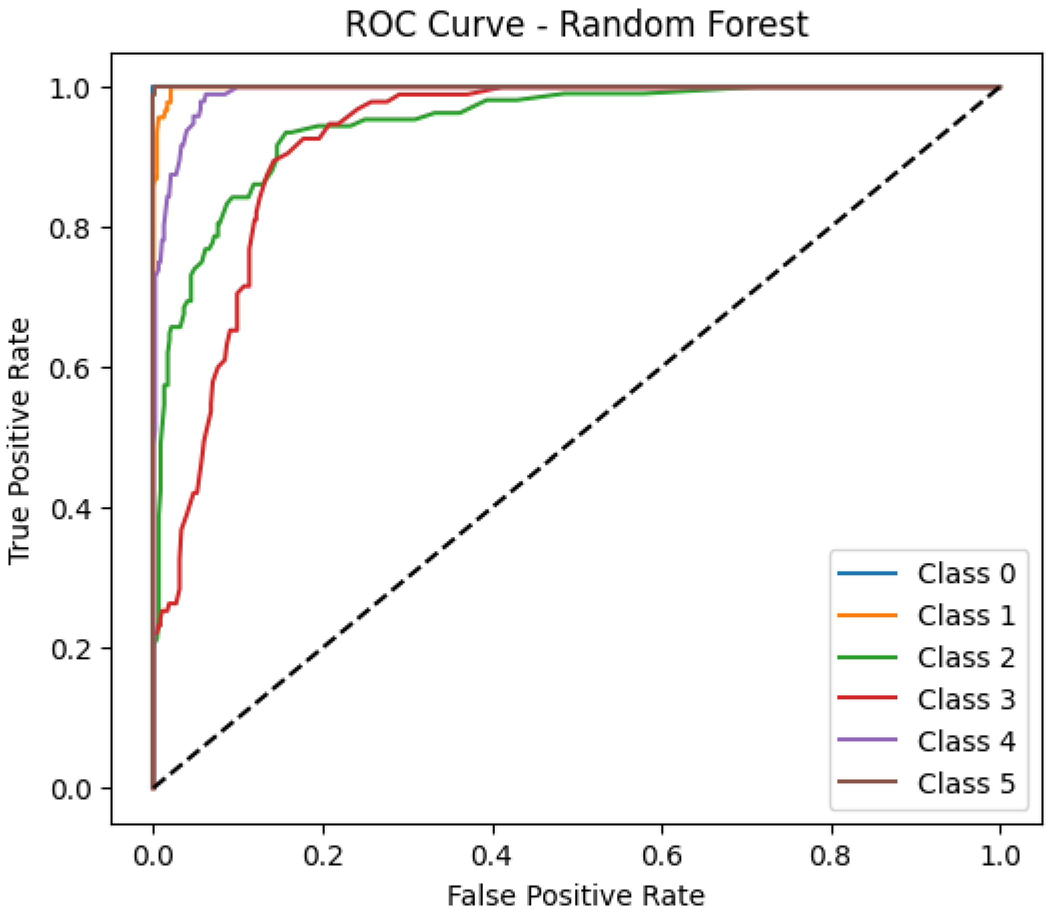
In [142...

```
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.cla
disp.plot()
plt.title("Confusion Matrix")
plt.show()
```



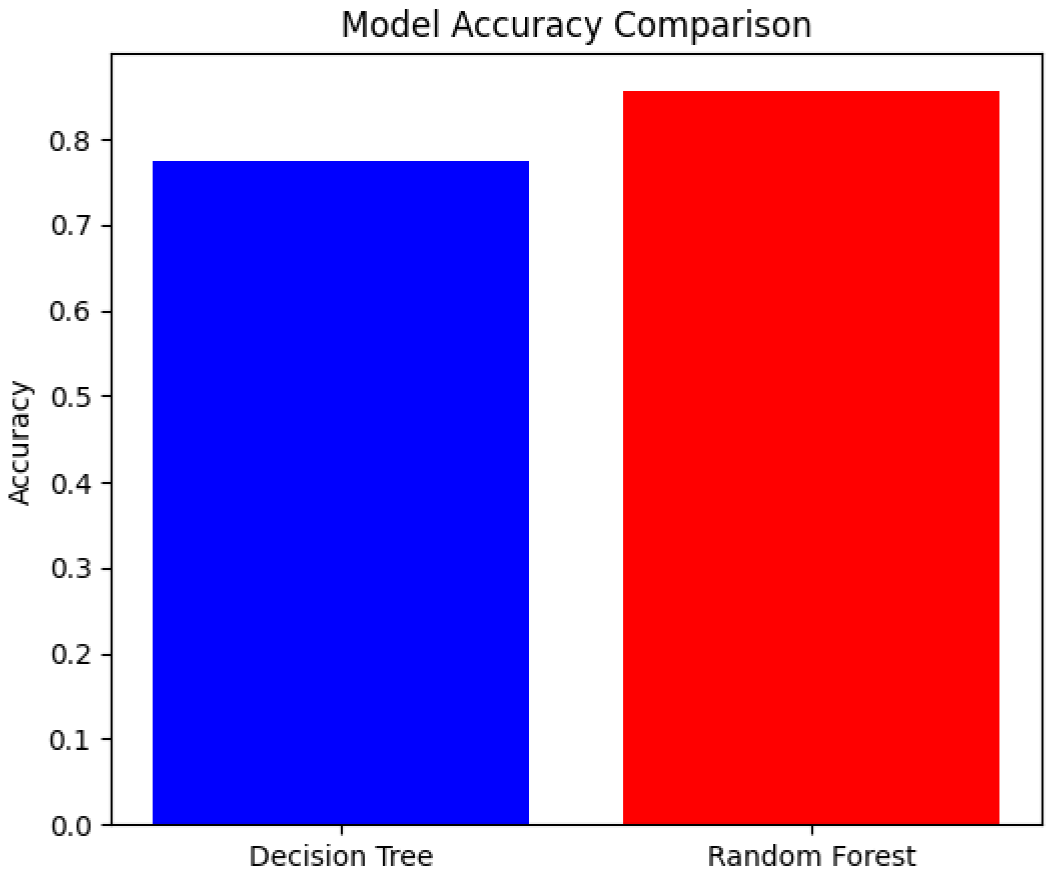
In [143...

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

y_test_bin = label_binarize(y_test, classes=sorted(df['quality'].unique()))

y_score = rc.predict_proba(x_test_scaled)

plt.figure(figsize=(6,5))
for i in range(y_test_bin.shape[1]):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    plt.plot(fpr, tpr, label=f"Class {i}")

plt.plot([0,1], [0,1], "k--")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC - Random Forest")
plt.legend()
plt.show()
```



In [145...

```
plt.figure(figsize=(6,5))
plt.bar(["Decision Tree", "Random Forest"], [acc_dt, acc_rf], color=["blue"
plt.ylabel("Accuracy")
plt.title("Model Accuracy Comparison")
plt.show()
```


Model Accuracy Comparison
Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Decision Tree
Random Forest

```
In [ ]:
```