In [29]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
df=pd.read_csv('wineQualityReds.csv')
```

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

In [6]:
```
df.tail()
```

Out[6]:

	Unnamed: 0	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density	pH	sulph
1594	1595	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	
1595	1596	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	
1596	1597	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	
1597	1598	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	
1598	1599	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	

In [7]:
```
df.sample(5)
```

Out[7]:

	Unnamed: 0	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density	pH	sulph
1158	1159	6.7	0.410	0.43	2.8	0.076	22.0	54.0	0.99572	3.42	
647	648	8.3	0.845	0.01	2.2	0.070	5.0	14.0	0.99670	3.32	
17	18	8.1	0.560	0.28	1.7	0.368	16.0	56.0	0.99680	3.11	
1472	1473	7.6	0.350	0.60	2.6	0.073	23.0	44.0	0.99656	3.38	
997	998	5.6	0.660	0.00	2.2	0.087	3.0	11.0	0.99378	3.71	

In [8]:
```
df.shape
```

Out[8]:
```
(1599, 13)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Unnamed: 0            1599 non-null   int64  
 1   fixed.acidity         1599 non-null   float64
 2   volatile.acidity      1599 non-null   float64
 3   citric.acid           1599 non-null   float64
 4   residual.sugar        1599 non-null   float64
 5   chlorides             1599 non-null   float64
 6   free.sulfur.dioxide   1599 non-null   float64
 7   total.sulfur.dioxide  1599 non-null   float64
 8   density               1599 non-null   float64
 9   pH                    1599 non-null   float64
 10  sulphates             1599 non-null   float64
 11  alcohol               1599 non-null   float64
 12  quality               1599 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 162.5 KB
```

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
Unnamed: 0              0
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 [13]:
```
df.shape
```

Out[13]: (1599, 13)

In [17]:
```
duplicate=df.duplicated()
print(duplicate.sum())
df[duplicate]
```

```
0
```

Out[17]:

Unnamed: 0	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density	pH	sulphates

In [18]:
```
df.dropna()
```

Out[18]:

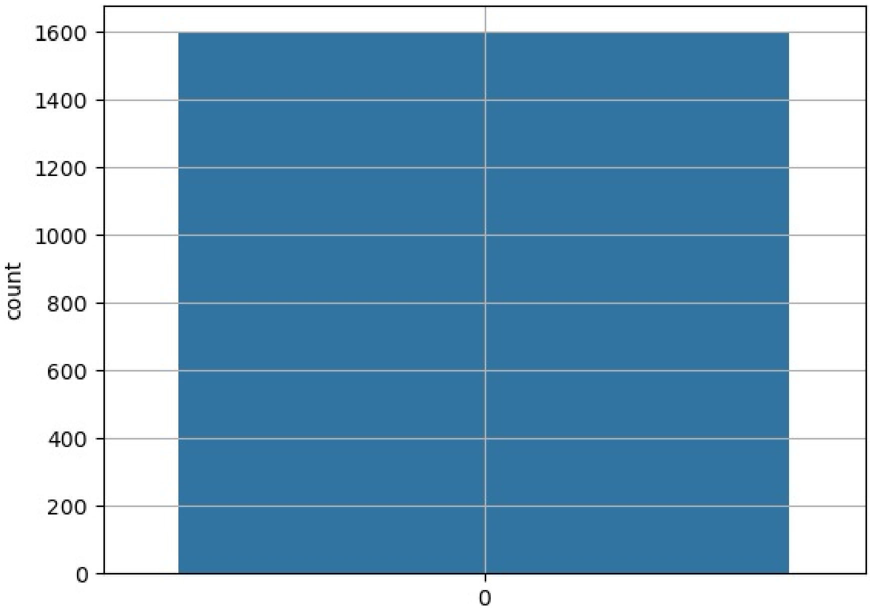
	Unnamed: 0	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density	pH	sulph
0	1	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	
1	2	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	
2	3	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	
3	4	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	
4	5	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	
...	...	...	...	...	...	...	...	...	...	...	
1594	1595	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	
1595	1596	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	
1596	1597	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	
1597	1598	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	
1598	1599	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	

1599 rows × 13 columns

In [19]:
```
print(df.quality.value_counts())
```

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

In [31]:
```
sns.countplot(df['quality'])
plt.grid()
plt.show()
```



In [24]:
```
df.corr()
```

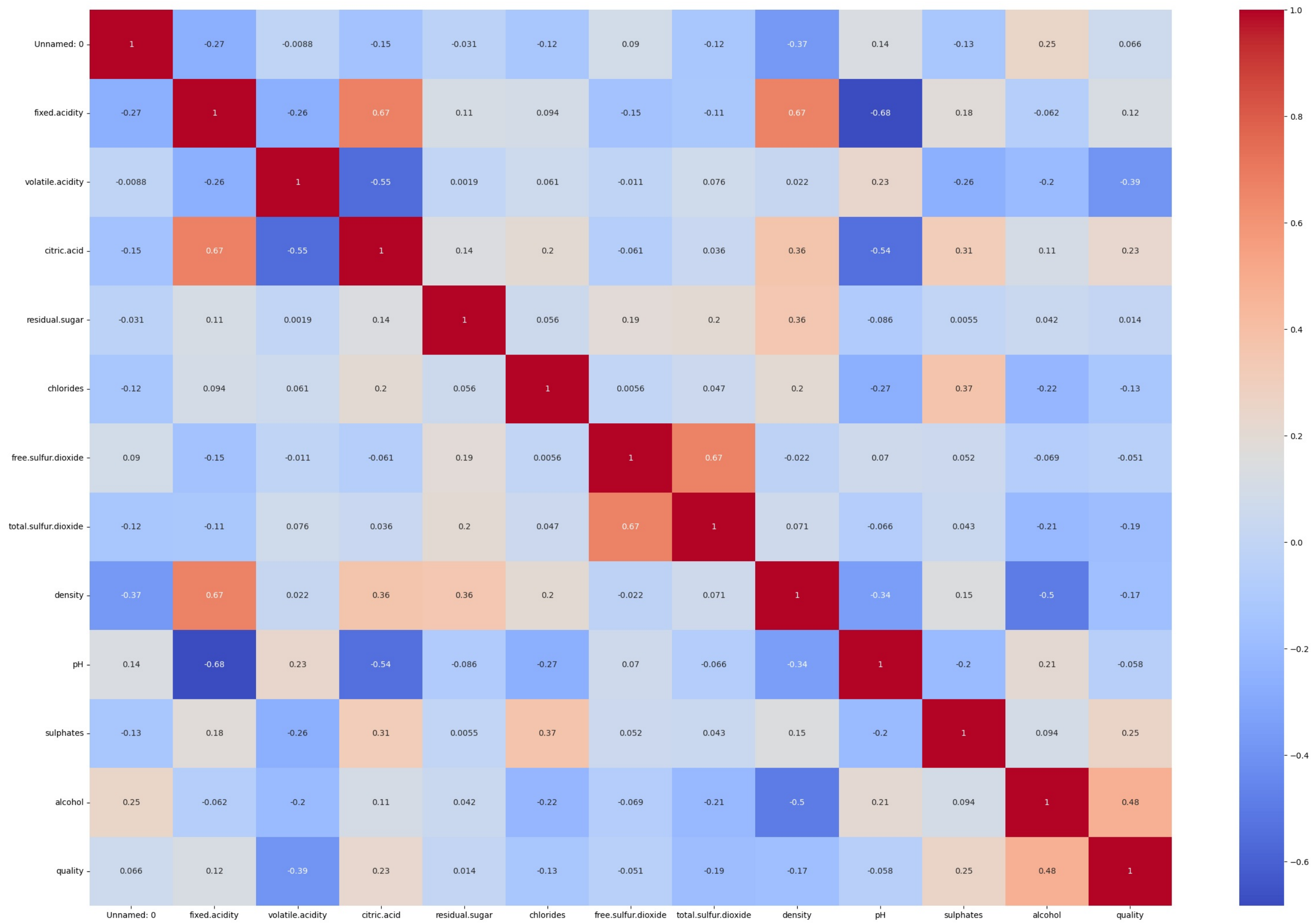
Out[24]:

	Unnamed: 0	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide	dens
Unnamed: 0	1.000000	-0.268484	-0.008815	-0.153551	-0.031261	-0.119869	0.090480	-0.117850	-0.3683
fixed.acidity	-0.268484	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794	-0.113181	0.6680
volatile.acidity	-0.008815	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504	0.076470	0.0220
citric.acid	-0.153551	0.671703	-0.552496	1.000000	0.143577	0.203823	-0.060978	0.035533	0.3649
residual.sugar	-0.031261	0.114777	0.001918	0.143577	1.000000	0.055610	0.187049	0.203028	0.3552
chlorides	-0.119869	0.093705	0.061298	0.203823	0.055610	1.000000	0.005562	0.047400	0.2006
free.sulfur.dioxide	0.090480	-0.153794	-0.010504	-0.060978	0.187049	0.005562	1.000000	0.667666	-0.0219
total.sulfur.dioxide	-0.117850	-0.113181	0.076470	0.035533	0.203028	0.047400	0.667666	1.000000	0.0712
density	-0.368372	0.668047	0.022026	0.364947	0.355283	0.200632	-0.021946	0.071269	1.0000
pH	0.136005	-0.682978	0.234937	-0.541904	-0.085652	-0.265026	0.070377	-0.066495	-0.3416
sulphates	-0.125307	0.183006	-0.260987	0.312770	0.005527	0.371260	0.051658	0.042947	0.1485
alcohol	0.245123	-0.061668	-0.202288	0.109903	0.042075	-0.221141	-0.069408	-0.205654	-0.4961
quality	0.066453	0.124052	-0.390558	0.226373	0.013732	-0.128907	-0.050656	-0.185100	-0.1749

In [30]:

```python
corr= df.corr()
plt.figure(figsize=(30,20))
sns.heatmap(corr,annot=True,cmap='coolwarm')
```

Out[30]: <Axes: >



In [32]:

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

In [33]:

```python
x.head()
```

Out[33]:

	Unnamed: 0	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density	pH	sulphates
0	1	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56
1	2	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68
2	3	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65
3	4	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58
4	5	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56

In [35]:
```
x.shape
```
Out[35]:
```
(1599, 12)
```

In [36]:
```
y.head()
```
Out[36]:
```
0    5
1    5
2    5
3    6
4    5
Name: quality, dtype: int64
```

In [37]:
```
y.shape
```
Out[37]:
```
(1599,)
```

In [38]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_res = sc.fit_transform(x)
```

In [39]:
```
x.head()
```
Out[39]:

	Unnamed: 0	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density	pH	sulphates
0	1	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56
1	2	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68
2	3	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65
3	4	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58
4	5	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56

In [40]:
```
x_res
```
Out[40]:
```
array([[-1.73096794, -0.52835961,  0.96187667, ...,  1.28864292,
        -0.57920652, -0.96024611],
       [-1.72880152, -0.29854743,  1.96744245, ..., -0.7199333 ,
         0.1289504 , -0.58477711],
       [-1.7266351 , -0.29854743,  1.29706527, ..., -0.33117661,
        -0.04808883, -0.58477711],
       ...,
       [ 1.7266351 , -1.1603431 , -0.09955388, ...,  0.70550789,
         0.54204194,  0.54162988],
       [ 1.72880152, -1.39015528,  0.65462046, ...,  1.6773996 ,
         0.30598963, -0.20930812],
       [ 1.73096794, -1.33270223, -1.21684919, ...,  0.51112954,
         0.01092425,  0.54162988]])
```

In [41]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif_data = pd.DataFrame()
vif_data["vif"] =[variance_inflation_factor(x_res,i) for i in range(x_res.shape[1])]
vif_data["Features"]=x.columns
vif_data
```

Out[41]:

	vif	Features
0	1.270873	Unnamed: 0
1	7.888640	fixed.acidity
2	1.790625	volatile.acidity
3	3.133061	citric.acid
4	1.704435	residual.sugar
5	1.485224	chlorides
6	2.060964	free.sulfur.dioxide
7	2.314581	total.sulfur.dioxide
8	6.369056	density
9	3.378271	pH
10	1.440547	sulphates
11	3.058106	alcohol

In [42]:
```
x_res.shape
```
Out[42]:
```
(1599, 12)
```

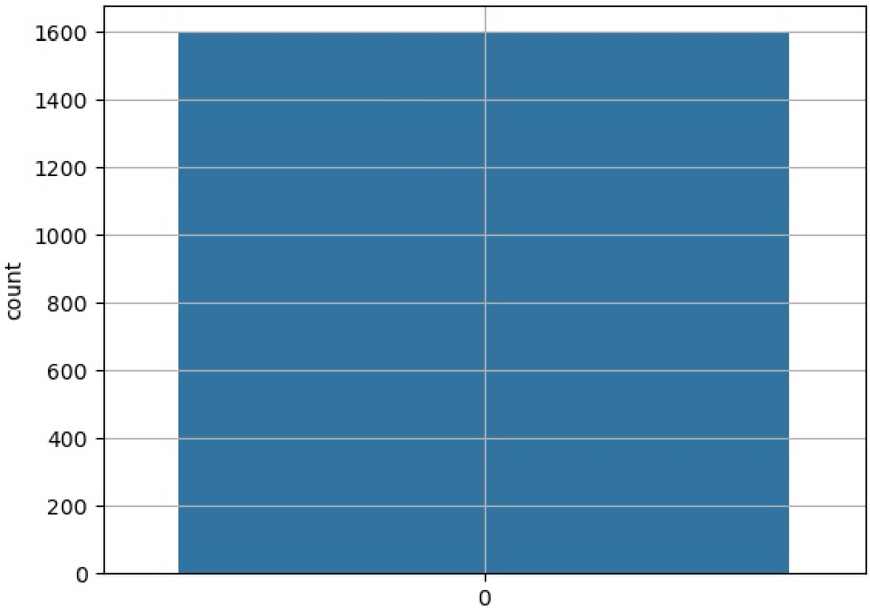
In [43]:
```
x1=x.drop(['residual.sugar','density'],axis=1)
x1.shape
```
Out[43]:
```
(1599, 10)
```

In [52]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
(scaler.fit(x1))
rescaledx = scaler.transform(x1)
```

In [53]:
```
rescaledx.shape
```
Out[53]:
```
(1599, 10)
```

In [54]:
```
y.value_counts()
```
Out[54]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

In [56]:
```
sns.countplot(df['quality'])
plt.grid()
plt.show()
```



In [57]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(rescaledx,y,test_size=0.2,random_state=7)
```

In [58]:
```
x_train.shape,y_train.shape
```

Out[58]: ((1279, 10), (1279,))

In [59]:
```
x_test.shape,y_test.shape
```
Out[59]: ((320, 10), (320,))

In [60]:
```
from sklearn.tree import DecisionTreeClassifier
dt= DecisionTreeClassifier()
dt.fit(x_train,y_train)
dt.train_pred=dt.predict(x_train)
dt_test_pred=dt.predict(x_test)
```

In [61]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
```

In [79]:
```
print('Train Accuracy:',accuracy_score(y_train,dt.train_pred)*100)
```
```
Train Accuracy: 100.0
```

In [81]:
```
print('Accuracy Score:',accuracy_score(y_test,dt_test_pred)*100)
```
```
Accuracy Score: 62.5
```

In [84]:
```
print(confusion_matrix(y_test,dt_test_pred))
```
```
[[ 0  0  0  0  0  0]
 [ 0  1  8  1  0  0]
 [ 1  5 82 30  3  2]
 [ 3  3 29 94 17  0]
 [ 0  1  4  9 21  2]
 [ 0  0  0  2  0  2]]
```

In [85]:
```
print(classification_report(y_test,dt_test_pred,digits=4))
```
```
              precision    recall  f1-score   support

           3     0.0000    0.0000    0.0000         0
           4     0.1000    0.1000    0.1000        10
           5     0.6667    0.6667    0.6667       123
           6     0.6912    0.6438    0.6667       146
           7     0.5122    0.5676    0.5385        37
           8     0.3333    0.5000    0.4000         4

    accuracy                         0.6250       320
   macro avg     0.3839    0.4130    0.3953       320
weighted avg     0.6381    0.6250    0.6308       320
```

In [89]:
```
(0.0000+0.1000+0.6667+0.6912+0.5122+0.3333)/6
```
Out[89]: 0.38389999999999996

In [87]:
```
from sklearn.metrics import precision_score,recall_score,f1_score,classification_report,confusion_matrix
print("precision score of macro is:",round(precision_score(y_test,dt_test_pred,average='macro')*100,2))
print("precision score of micro is:",round(precision_score(y_test,dt_test_pred,average='micro')*100,2))
print("precision score of weighted is:",round(precision_score(y_test,dt_test_pred,average='weighted')*100,2))
```
```
precision score of macro is: 38.39
precision score of micro is: 62.5
precision score of weighted is: 63.81
```

In [88]:
```
print("recall_score of macro is:",round(recall_score(y_test,dt_test_pred,average='macro')*100,2))
print("recall_score of micro is:",round(recall_score(y_test,dt_test_pred,average='micro')*100,2))
print("recall_score of weighted is:",round(recall_score(y_test,dt_test_pred,average='weighted')*100,2))
```
```
recall_score of macro is: 41.3
recall_score of micro is: 62.5
recall_score of weighted is: 62.5
```

In [90]:
```
print('f1_score of macro:',round(f1_score(y_test,dt_test_pred,average='macro')*100,2))
print('f1_score of micro:',round(f1_score(y_test,dt_test_pred,average='micro')*100,2))
print('f1_score of weighted:',round(f1_score(y_test,dt_test_pred,average='weighted')*100,2))
```
```
f1_score of macro: 39.53
f1_score of micro: 62.5
f1_score of weighted: 63.08
```

In [91]:
```
df.head()
```

Out[91]:

	Unnamed: 0	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density	pH	sulphates
0	1	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56
1	2	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68
2	3	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65
3	4	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58
4	5	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56

In [103…

```
input_data = (1,7.4,0.70,0.00,1.9,0.07,11.0,34.0,0.9978,3.51)
input_data_as_numpy_array=np.asarray(input_data)

input_data_reshaped=input_data_as_numpy_array.reshape(1,-1)

prediction = dt.predict(input_data_reshaped)

print(prediction)
```

```
[6]
```

In [111…

```
from sklearn.ensemble import RandomForestClassifier
clf= RandomForestClassifier(random_state=1)

clf.fit(x_train,y_train)

clf_train_pred = clf.predict(x_train)

clf_test_pred = clf.predict(x_test)
```

In [112…

```
print('Train_accuracy:',accuracy_score(y_train,clf_train_pred)*100)
```

```
Train_accuracy: 100.0
```

In [114…

```
print('Accuracy Score:',accuracy_score(y_test,clf_test_pred)*100)
```

```
Accuracy Score: 69.375
```

In [115…

```
print(confusion_matrix(y_test,clf_test_pred))
```

```
[[  1   8   0   1   0]
 [  0  98  23   2   0]
 [  1  26 100  19   0]
 [  0   3  12  22   0]
 [  0   0   1   2   1]]
```

In [117…

```
print("recall_score of macro is:",round(recall_score(y_test,clf_test_pred,average='macro')*100,2))
print("recall_score of micro is:",round(recall_score(y_test,clf_test_pred,average='micro')*100,2))
print("recall_score of weighted is:",round(recall_score(y_test,clf_test_pred,average='weighted')*100,2))
```

```
recall_score of macro is: 48.53
recall_score of micro is: 69.38
recall_score of weighted is: 69.38
```

In [118…

```
print('f1_score of macro:',round(f1_score(y_test,clf_test_pred,average='macro')*100,2))
print('f1_score of micro:',round(f1_score(y_test,clf_test_pred,average='micro')*100,2))
print('f1_score of weighted:',round(f1_score(y_test,clf_test_pred,average='weighted')*100,2))
```

```
f1_score of macro: 51.31
f1_score of micro: 69.38
f1_score of weighted: 68.71
```

In [127…

```
from sklearn import svm
```

In [128…

```
clf = svm.SVC(probability=True)
clf.fit(x_train,y_train)
```

Out[128]:

```
▾ SVC
SVC(probability=True)
```

In [143…

```
clf_train_pred=clf.predict(x_train)
clf_train_pred=clf.predict(x_test)
```

In [149…

```
print('Train Accuracy:',accuracy_score(y_test,clf_train_pred)*100)
```

```
Train Accuracy: 61.875
```

In [153…

```
print('Accuracy Score:',accuracy_score(y_test,clf_test_pred)*100)
```

```
Accuracy Score: 69.375
```

In [154…

```
print(confusion_matrix(y_test,clf_test_pred))
```

```
[[  1   8   0   1   0]
 [  0  98  23   2   0]
 [  1  26 100  19   0]
 [  0   3  12  22   0]
 [  0   0   1   2   1]]
```

In [155…

```
print("precision score of macro is:",round(precision_score(y_test,clf_test_pred,average='macro')*100,2))
print("precision score of micro is:",round(precision_score(y_test,clf_test_pred,average='micro')*100,2))
print("precision score of weighted is:",round(precision_score(y_test,clf_test_pred,average='weighted')*100,2))
```

```
precision score of macro is: 68.79
precision score of micro is: 69.38
precision score of weighted is: 69.79
```

In [156…

```
print("recall_score of macro is:",round(recall_score(y_test,clf_test_pred,average='macro')*100,2))
print("recall_score of micro is:",round(recall_score(y_test,clf_test_pred,average='micro')*100,2))
print("recall_score of weighted is:",round(recall_score(y_test,clf_test_pred,average='weighted')*100,2))
```

```
recall_score of macro is: 48.53
recall_score of micro is: 69.38
recall_score of weighted is: 69.38
```

In [157…

```
print('f1_score of macro:',round(f1_score(y_test,clf_test_pred,average='macro')*100,2))
print('f1_score of micro:',round(f1_score(y_test,clf_test_pred,average='micro')*100,2))
print('f1_score of weighted:',round(f1_score(y_test,clf_test_pred,average='weighted')*100,2))
```

```
f1_score of macro: 51.31
f1_score of micro: 69.38
f1_score of weighted: 68.71
```

In [ ]: