wine-quality-prediction

June 24, 2023

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

[2]:
```
wine_data = pd.read_excel("C:/Users/DeLL/Desktop/wine_data_csv.xlsx")
```

[3]:
```
wine_data.head(20)
```

[3]:
```
    fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0             7.0              0.27         0.36           20.70      0.045
1             6.3              0.30         0.34            1.60      0.049
2             8.1              0.28         0.40            6.90      0.050
3             7.2              0.23         0.32            8.50      0.058
4             7.2              0.23         0.32            8.50      0.058
5             8.1              0.28         0.40            6.90      0.050
6             6.2              0.32         0.16            7.00      0.045
7             7.0              0.27         0.36           20.70      0.045
8             6.3              0.30         0.34            1.60      0.049
9             8.1              0.22         0.43            1.50      0.044
10            8.1              0.27         0.41            1.45      0.033
11            8.6              0.23         0.40            4.20      0.035
12            7.9              0.18         0.37            1.20      0.040
13            6.6              0.16         0.40            1.50      0.044
14            8.3              0.42         0.62           19.25      0.040
15            6.6              0.17         0.38            1.50      0.032
16            6.3              0.48         0.04            1.10      0.046
17            6.2              0.66         0.48            1.20      0.029
18            7.4              0.34         0.42            1.10      0.033
19            6.5              0.31         0.14            7.50      0.044

    free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                  45.0                 170.0   1.0010  3.00       0.45
1                  14.0                 132.0   0.9940  3.30       0.49
2                  30.0                  97.0   0.9951  3.26       0.44
3                  47.0                 186.0   0.9956  3.19       0.40
4                  47.0                 186.0   0.9956  3.19       0.40
5                  30.0                  97.0   0.9951  3.26       0.44
6                  30.0                 136.0   0.9949  3.18       0.47
7                  45.0                 170.0   1.0010  3.00       0.45
```

```
8                 14.0                  132.0   0.9940  3.30       0.49
9                 28.0                  129.0   0.9938  3.22       0.45
10                11.0                   63.0   0.9908  2.99       0.56
11                17.0                  109.0   0.9947  3.14       0.53
12                16.0                   75.0   0.9920  3.18       0.63
13                48.0                  143.0   0.9912  3.54       0.52
14                41.0                  172.0   1.0002  2.98       0.67
15                28.0                  112.0   0.9914  3.25       0.55
16                30.0                   99.0   0.9928  3.24       0.36
17                29.0                   75.0   0.9892  3.33       0.39
18                17.0                  171.0   0.9917  3.12       0.53
19                34.0                  133.0   0.9955  3.22       0.50

    alcohol  quality
0       8.8        6
1       9.5        6
2      10.1        6
3       9.9        6
4       9.9        6
5      10.1        6
6       9.6        6
7       8.8        6
8       9.5        6
9      11.0        6
10     12.0        5
11      9.7        5
12     10.8        5
13     12.4        7
14      9.7        5
15     11.4        7
16      9.6        6
17     12.8        8
18     11.3        6
19      9.5        5
```

[4]:
```
wine_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
```

```
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
 11  quality               4898 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
```

[5]:
```
wine_data.isnull().sum()
```

[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
dtype: int64
```

[6]:
```
wine_data.quality.unique()
```

[6]:
```
array([6, 5, 7, 8, 4, 3, 9], dtype=int64)
```

[7]:
```
##converting quality column to categorical column
for i in range(len(wine_data.quality)):
    if wine_data.quality[i] <= 4:
        wine_data.quality[i] = 0
    elif wine_data.quality[i] > 4 and wine_data.quality[i] <= 6 :
        wine_data.quality[i] = 1
    else:
        wine_data.quality[i] = 2
wine_data.head(20)
```

```
C:\Users\DeLL\AppData\Local\Temp\ipykernel_25720\2478559651.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  wine_data.quality[i] = 1
C:\Users\DeLL\AppData\Local\Temp\ipykernel_25720\2478559651.py:8:
```

```
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  wine_data.quality[i] = 2
C:\Users\DeLL\AppData\Local\Temp\ipykernel_25720\2478559651.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  wine_data.quality[i] = 0
```

[7]:

```
    fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0             7.0              0.27         0.36           20.70      0.045
1             6.3              0.30         0.34            1.60      0.049
2             8.1              0.28         0.40            6.90      0.050
3             7.2              0.23         0.32            8.50      0.058
4             7.2              0.23         0.32            8.50      0.058
5             8.1              0.28         0.40            6.90      0.050
6             6.2              0.32         0.16            7.00      0.045
7             7.0              0.27         0.36           20.70      0.045
8             6.3              0.30         0.34            1.60      0.049
9             8.1              0.22         0.43            1.50      0.044
10            8.1              0.27         0.41            1.45      0.033
11            8.6              0.23         0.40            4.20      0.035
12            7.9              0.18         0.37            1.20      0.040
13            6.6              0.16         0.40            1.50      0.044
14            8.3              0.42         0.62           19.25      0.040
15            6.6              0.17         0.38            1.50      0.032
16            6.3              0.48         0.04            1.10      0.046
17            6.2              0.66         0.48            1.20      0.029
18            7.4              0.34         0.42            1.10      0.033
19            6.5              0.31         0.14            7.50      0.044

    free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                  45.0                 170.0   1.0010  3.00       0.45
1                  14.0                 132.0   0.9940  3.30       0.49
2                  30.0                  97.0   0.9951  3.26       0.44
3                  47.0                 186.0   0.9956  3.19       0.40
4                  47.0                 186.0   0.9956  3.19       0.40
5                  30.0                  97.0   0.9951  3.26       0.44
6                  30.0                 136.0   0.9949  3.18       0.47
7                  45.0                 170.0   1.0010  3.00       0.45
8                  14.0                 132.0   0.9940  3.30       0.49
9                  28.0                 129.0   0.9938  3.22       0.45
```

```
10              11.0                  63.0   0.9908  2.99       0.56
11              17.0                 109.0   0.9947  3.14       0.53
12              16.0                  75.0   0.9920  3.18       0.63
13              48.0                 143.0   0.9912  3.54       0.52
14              41.0                 172.0   1.0002  2.98       0.67
15              28.0                 112.0   0.9914  3.25       0.55
16              30.0                  99.0   0.9928  3.24       0.36
17              29.0                  75.0   0.9892  3.33       0.39
18              17.0                 171.0   0.9917  3.12       0.53
19              34.0                 133.0   0.9955  3.22       0.50

    alcohol  quality
0       8.8        1
1       9.5        1
2      10.1        1
3       9.9        1
4       9.9        1
5      10.1        1
6       9.6        1
7       8.8        1
8       9.5        1
9      11.0        1
10     12.0        1
11      9.7        1
12     10.8        1
13     12.4        2
14      9.7        1
15     11.4        2
16      9.6        1
17     12.8        2
18     11.3        1
19      9.5        1
```

[8]:
```
##some data visualisation
import matplotlib.pyplot as plt
import matplotlib
import plotly.express as px
import seaborn as sns
%matplotlib inline

#sns.set_style('darkgrid')
matplotlib.rcParams['font.size'] = 10
matplotlib.rcParams['figure.figsize'] = (8,6)
matplotlib.rcParams['figure.facecolor'] = '#00000000'
```

[9]:
```
x = wine_data.quality.value_counts()
x
```

[9]:
```
1    3655
2    1060
0     183
Name: quality, dtype: int64
```

[10]:
```
fig = px.histogram(wine_data,x = 'fixed acidity',color = 'quality')
fig.show()
```

[11]:
```
px.scatter(wine_data,x='pH',y='alcohol',color = 'quality')
```

[12]:
```
px.scatter(wine_data,x='fixed acidity',y='alcohol',color = 'quality')
```

[13]:
```
input_cols = list(wine_data.columns)[0:-1]
target_cols = 'quality'
input_cols
```

[13]:
```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol']
```

[14]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(wine_data[input_cols])
wine_data
```

[14]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	\
0	7.0	0.27	0.36	20.7	0.045	
1	6.3	0.30	0.34	1.6	0.049	
2	8.1	0.28	0.40	6.9	0.050	
3	7.2	0.23	0.32	8.5	0.058	
4	7.2	0.23	0.32	8.5	0.058	
…	…	…	…	…	…	
4893	6.2	0.21	0.29	1.6	0.039	
4894	6.6	0.32	0.36	8.0	0.047	
4895	6.5	0.24	0.19	1.2	0.041	
4896	5.5	0.29	0.30	1.1	0.022	
4897	6.0	0.21	0.38	0.8	0.020	

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	\

```
0                    45.0                   170.0  1.00100  3.00        0.45
1                    14.0                   132.0  0.99400  3.30        0.49
2                    30.0                    97.0  0.99510  3.26        0.44
3                    47.0                   186.0  0.99560  3.19        0.40
4                    47.0                   186.0  0.99560  3.19        0.40
…                     …                    …       …      …       …
4893                 24.0                    92.0  0.99114  3.27        0.50
4894                 57.0                   168.0  0.99490  3.15        0.46
4895                 30.0                   111.0  0.99254  2.99        0.46
4896                 20.0                   110.0  0.98869  3.34        0.38
4897                 22.0                    98.0  0.98941  3.26        0.32

      alcohol  quality
0         8.8        1
1         9.5        1
2        10.1        1
3         9.9        1
4         9.9        1
…        …      …
4893     11.2        1
4894      9.6        1
4895      9.4        1
4896     12.8        2
4897     11.8        1

[4898 rows x 12 columns]
```

[15]:
```python
wine_data[input_cols] = scaler.transform(wine_data[input_cols])
wine_data
```

[15]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chloridesw \
0          0.307692          0.186275     0.216867        0.308282   0.106825
1          0.240385          0.215686     0.204819        0.015337   0.118694
2          0.413462          0.196078     0.240964        0.096626   0.121662
3          0.326923          0.147059     0.192771        0.121166   0.145401
4          0.326923          0.147059     0.192771        0.121166   0.145401
…              …                 …           …              …         …
4893       0.230769          0.127451     0.174699        0.015337   0.089021
4894       0.269231          0.235294     0.216867        0.113497   0.112760
4895       0.259615          0.156863     0.114458        0.009202   0.094955
4896       0.163462          0.205882     0.180723        0.007669   0.038576
4897       0.211538          0.127451     0.228916        0.003067   0.032641

      free sulfur dioxide  total sulfur dioxide   density        pH  \
0                0.149826              0.373550  0.267785  0.254545
1                0.041812              0.285383  0.132832  0.527273
2                0.097561              0.204176  0.154039  0.490909
```

```
3                0.156794                 0.410673  0.163678  0.427273
4                0.156794                 0.410673  0.163678  0.427273
…                   …                      …        …        …
4893             0.076655                 0.192575  0.077694  0.500000
4894             0.191638                 0.368910  0.150183  0.390909
4895             0.097561                 0.236659  0.104685  0.245455
4896             0.062718                 0.234339  0.030461  0.563636
4897             0.069686                 0.206497  0.044342  0.490909

      sulphates   alcohol  quality
0      0.267442  0.129032        1
1      0.313953  0.241935        1
2      0.255814  0.338710        1
3      0.209302  0.306452        1
4      0.209302  0.306452        1
…           …        …       …
4893   0.325581  0.516129        1
4894   0.279070  0.258065        1
4895   0.279070  0.225806        1
4896   0.186047  0.774194        2
4897   0.116279  0.612903        1

[4898 rows x 12 columns]
```

[16]:
```
from sklearn.model_selection import train_test_split
train_wd ,test_wd = train_test_split(wine_data,test_size = 0.2 , random_state =␣
↪42)
```

[17]:
```
train_wd.shape
```

[17]:
```
(3918, 12)
```

[18]:
```
test_wd.shape
```

[18]:
```
(980, 12)
```

[19]:
```
train_wd.quality.value_counts()
```

[19]:
```
1    2932
2     833
0     153
Name: quality, dtype: int64
```

[20]:
```
### fitting logistic model
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression(solver = 'liblinear')
logistic_model.fit(train_wd[input_cols],train_wd[target_cols])
```

[20]: LogisticRegression(solver='liblinear')

[21]:
```
logistic_model.coef_
```

[21]:
```
array([[ 2.14755807,  4.64141012, -0.47417683, -2.77934879,  0.41507708,
        -2.84902665, -2.34919539, -0.15062508,  0.01732111, -0.73756408,
        -2.32486903],
       [-1.05051633, -0.03872233,  0.84806422, -0.40763787,  1.3727226 ,
        -1.03913945,  1.6200581 ,  1.1052533 , -0.963344  , -0.71449952,
        -3.26125082],
       [-0.04807052, -2.96657137, -0.97505704,  2.13817511, -2.59292788,
         2.266616  , -0.90877459, -1.2990104 ,  0.88598542,  0.96047654,
         4.62086368]])
```

[22]:
```
logistic_model.intercept_
```

[22]:
```
array([-2.45142667,  2.7105759 , -3.07528366])
```

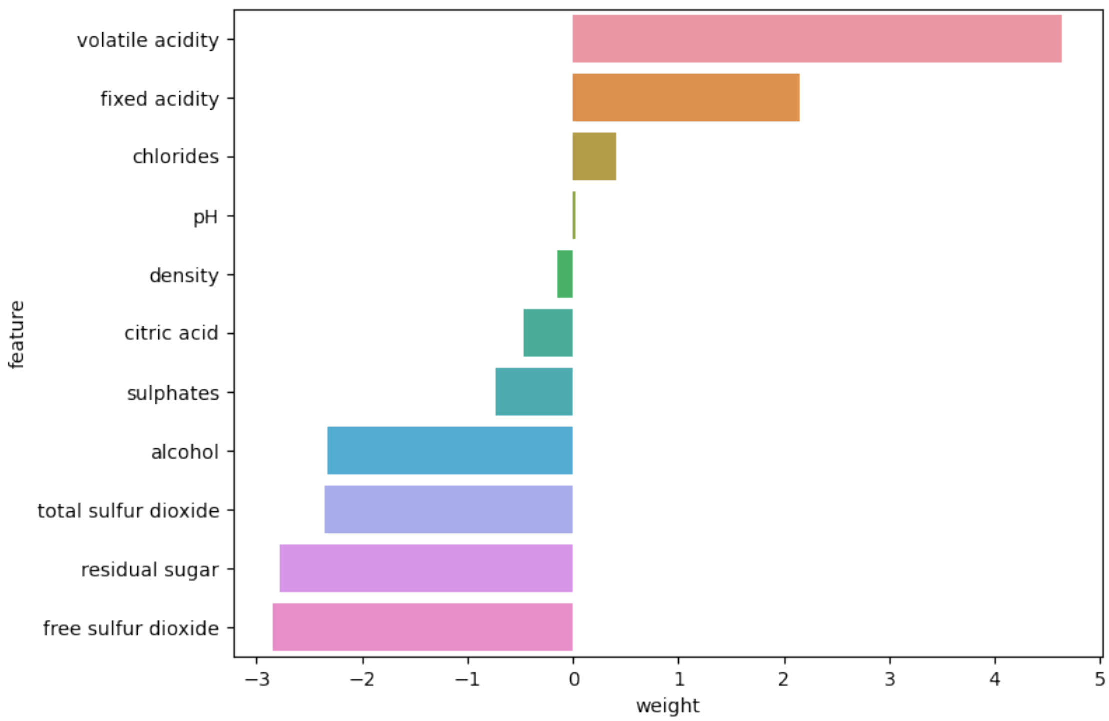
[23]:
```
weight_df = pd.DataFrame({
    'feature':input_cols,
    'weight' : logistic_model.coef_.tolist()[0]
})
weight_df
```

[23]:

	feature	weight
0	fixed acidity	2.147558
1	volatile acidity	4.641410
2	citric acid	-0.474177
3	residual sugar	-2.779349
4	chlorides	0.415077
5	free sulfur dioxide	-2.849027
6	total sulfur dioxide	-2.349195
7	density	-0.150625
8	pH	0.017321
9	sulphates	-0.737564
10	alcohol	-2.324869

[24]:
```
sns.barplot(data = weight_df.sort_values('weight',ascending = False),x =␣
 ↪'weight' , y = 'feature')
```

[24]: <AxesSubplot:xlabel='weight', ylabel='feature'>



[25]:
```
train_preds1 = logistic_model.predict(train_wd[input_cols])
```

[26]:
```
train_preds1
```

[26]:
```
array([2, 1, 1, …, 1, 1, 1], dtype=int64)
```

[27]:
```
from sklearn.metrics import accuracy_score
```

[28]:
```
accuracy_score(train_preds1,train_wd[target_cols])
```

[28]:
```
0.7738642164369577
```

[29]:
```
test_preds1 = logistic_model.predict(test_wd[input_cols])
```

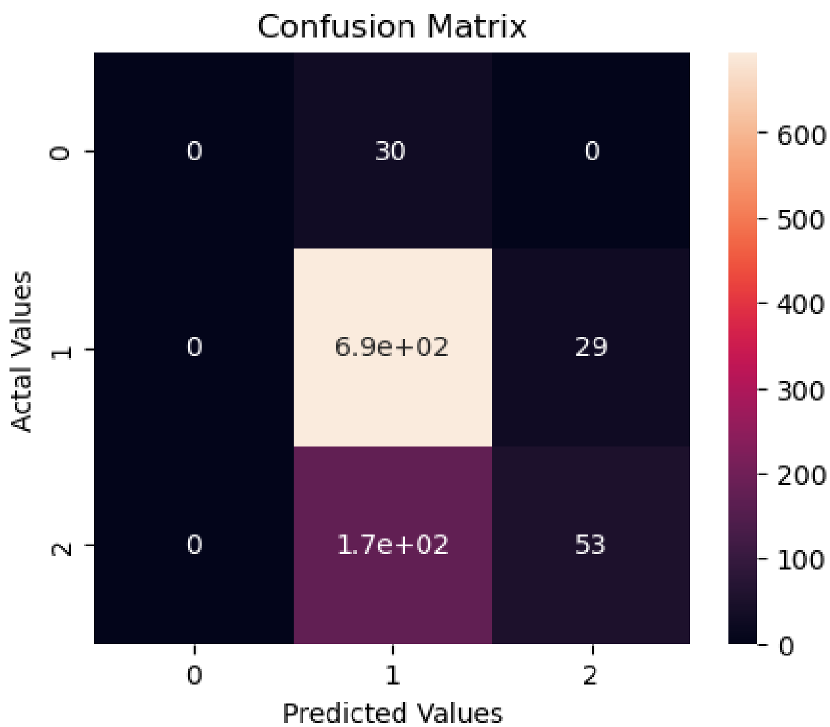
[30]:
```
accuracy_score(test_preds1,test_wd[target_cols])
```

[30]:
```
0.7622448979591837
```

[31]:
```
from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(test_wd.quality, test_preds1)
confusion_matrix
cm_df = pd.DataFrame(confusion_matrix,
                     index = [0,1,2],
                     columns = [0,1,2])
```

[32]:

```
plt.figure(figsize=(5,4))
sns.heatmap(cm_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



[33]:

```
from sklearn.metrics import classification_report, roc_auc_score, roc_curve
print(classification_report(test_wd.quality, test_preds1))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        30
           1       0.77      0.96      0.86       723
           2       0.65      0.23      0.34       227

    accuracy                           0.76       980
   macro avg       0.47      0.40      0.40       980
weighted avg       0.72      0.76      0.71       980
```

```
C:\Users\DeLL\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
```

```
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

C:\Users\DeLL\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

C:\Users\DeLL\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
```

[34]:

```
### Decision Tree method ###
parameter = {'criterion':['gini','entropy','log_loss'],
             'splitter':['best','random'],
             'max_depth':[1,3,5,7,9],
             'max_features':['auto','sqrt','log2']}
```

[35]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
decision_tree = DecisionTreeClassifier()
```

[36]:

```
decision_tree = GridSearchCV(decision_tree,param_grid = parameter , cv␣
 ↪=3,scoring = 'accuracy',verbose = 5)
decision_tree.fit(train_wd[input_cols],train_wd[target_cols])
```

```
Fitting 3 folds for each of 90 candidates, totalling 270 fits
[CV 1/3] END criterion=gini, max_depth=1, max_features=auto, splitter=best;,
score=0.749 total time=   0.0s
[CV 2/3] END criterion=gini, max_depth=1, max_features=auto, splitter=best;,
score=0.748 total time=   0.0s
[CV 3/3] END criterion=gini, max_depth=1, max_features=auto, splitter=best;,
score=0.748 total time=   0.0s
[CV 1/3] END criterion=gini, max_depth=1, max_features=auto, splitter=random;,
score=0.749 total time=   0.0s
[CV 2/3] END criterion=gini, max_depth=1, max_features=auto, splitter=random;,
score=0.748 total time=   0.0s
[CV 3/3] END criterion=gini, max_depth=1, max_features=auto, splitter=random;,
score=0.755 total time=   0.0s
[CV 1/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=best;,
score=0.749 total time=   0.0s
[CV 2/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=best;,
score=0.748 total time=   0.0s
[CV 3/3] END criterion=gini, max_depth=1, max_features=sqrt, splitter=best;,
```

```
C:\Users\DeLL\anaconda3\lib\site-packages\sklearn\tree\_classes.py:269:
FutureWarning:

`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features='sqrt'`.

C:\Users\DeLL\anaconda3\lib\site-packages\sklearn\tree\_classes.py:269:
FutureWarning:

`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features='sqrt'`.

C:\Users\DeLL\anaconda3\lib\site-packages\sklearn\tree\_classes.py:269:
FutureWarning:

`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features='sqrt'`.

C:\Users\DeLL\anaconda3\lib\site-packages\sklearn\tree\_classes.py:269:
FutureWarning:

`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features='sqrt'`.

C:\Users\DeLL\anaconda3\lib\site-packages\sklearn\tree\_classes.py:269:
FutureWarning:

`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features='sqrt'`.
```

[36]:
```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy', 'log_loss'],
                         'max_depth': [1, 3, 5, 7, 9],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'splitter': ['best', 'random']},
             scoring='accuracy', verbose=5)
```

[37]:
```
decision_tree.best_params_
```

[37]:
```
{'criterion': 'gini',
 'max_depth': 9,
 'max_features': 'auto',
 'splitter': 'best'}
```

[38]:
```
decision_tree = DecisionTreeClassifier(criterion = 'gini',max_depth= 7,
```

```
                                            max_features= 'log2',splitter =␣
↪'best',random_state=40)
decision_tree.fit(train_wd[input_cols],train_wd[target_cols])
```

[38]: DecisionTreeClassifier(max_depth=7, max_features='log2', random_state=40)

[39]:
```
train_preds2 = decision_tree.predict(train_wd[input_cols])
```

[40]:
```
accuracy_score(train_preds2,train_wd[target_cols])
```

[40]: 0.8052577845839715

[41]:
```
test_preds2 = decision_tree.predict(test_wd[input_cols])
```

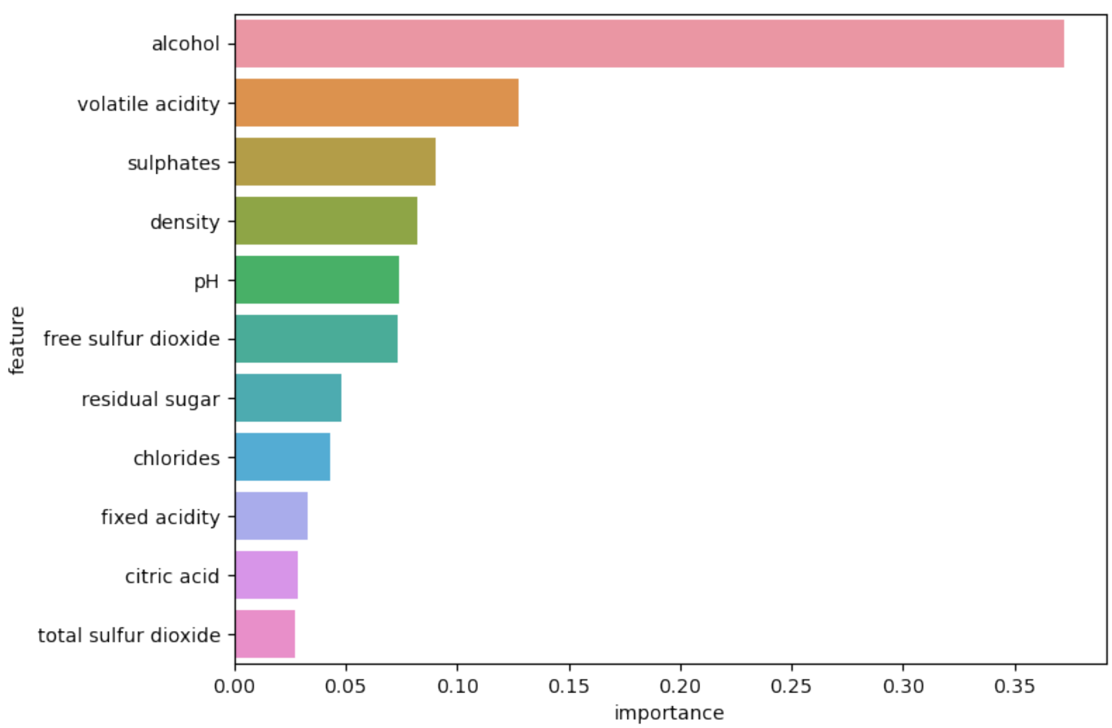
[42]:
```
accuracy_score(test_preds2,test_wd[target_cols])
```

[42]: 0.7653061224489796

[43]:
```
importance_df = pd.DataFrame({
    'feature':input_cols,
    'importance': decision_tree.feature_importances_
}).sort_values('importance',ascending = False)
```

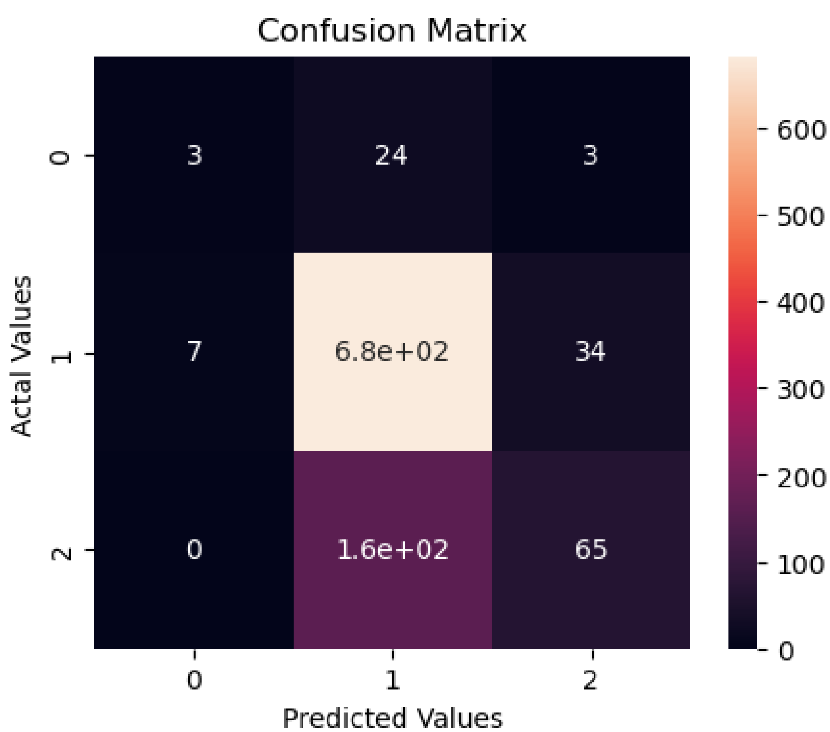
[44]:
```
sns.barplot(data=importance_df,x = 'importance',y = 'feature')
```

[44]: <AxesSubplot:xlabel='importance', ylabel='feature'>



[45]:
```
## confusion matrix
confusion_matrix = metrics.confusion_matrix(test_wd.quality, test_preds2)
confusion_matrix
cm_df = pd.DataFrame(confusion_matrix,
                     index = [0,1,2],
                     columns = [0,1,2])
```

[46]:
```
plt.figure(figsize=(5,4))
sns.heatmap(cm_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



[47]:
```
print(classification_report(test_wd.quality, test_preds2))
```

```
              precision    recall  f1-score   support

           0       0.30      0.10      0.15        30
           1       0.79      0.94      0.86       723
           2       0.64      0.29      0.40       227
```

```
    accuracy                           0.77       980
   macro avg       0.57      0.44      0.47       980
weighted avg       0.74      0.77      0.73       980
```

[48]:
```python
from sklearn.ensemble import RandomForestClassifier
```

[49]:
```python
para_value = {'n_estimators': [25, 50, 100, 150],
     'max_features': ['sqrt', 'log2', None],
     'max_depth': [3, 6, 9],
     'max_leaf_nodes': [3, 6, 9]}
```

[50]:
```python
random_forest = RandomForestClassifier()
```

[51]:
```python
random_forest = GridSearchCV(random_forest,para_value , cv =3,scoring =␣
↪'accuracy',verbose = 5)
random_forest.fit(train_wd[input_cols],train_wd[target_cols])
```

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
[CV 1/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=3, n_estimators=25;,
score=0.749 total time=   0.0s
[CV 2/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=3, n_estimators=25;,
score=0.748 total time=   0.0s
[CV 3/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=3, n_estimators=25;,
score=0.748 total time=   0.0s
[CV 1/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=3, n_estimators=50;,
score=0.749 total time=   0.0s
[CV 2/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=3, n_estimators=50;,
score=0.748 total time=   0.0s
[CV 3/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=3, n_estimators=50;,
score=0.748 total time=   0.0s
[CV 1/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=3,
n_estimators=100;, score=0.749 total time=   0.1s
[CV 2/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=3,
n_estimators=100;, score=0.748 total time=   0.1s
[CV 3/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=3,
n_estimators=100;, score=0.748 total time=   0.2s
[CV 1/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=3,
n_estimators=150;, score=0.749 total time=   0.3s
[CV 2/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=3,
n_estimators=150;, score=0.748 total time=   0.3s
[CV 3/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=3,
n_estimators=150;, score=0.748 total time=   0.2s
[CV 1/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=6, n_estimators=25;,
score=0.760 total time=   0.0s
[CV 2/3] END max_depth=3, max_features=sqrt, max_leaf_nodes=6, n_estimators=25;,
score=0.753 total time=   0.0s
```

```
                              'max_features': ['sqrt', 'log2', None],
                              'max_leaf_nodes': [3, 6, 9],
                              'n_estimators': [25, 50, 100, 150]},
             scoring='accuracy', verbose=5)
```

[52]:
```
random_forest.best_params_
```

[52]:
```
{'max_depth': 6,
 'max_features': None,
 'max_leaf_nodes': 9,
 'n_estimators': 150}
```

[53]:
```
random_forest = RandomForestClassifier(max_depth = 6,
 max_features = None,
 max_leaf_nodes=9,
 n_estimators = 50)
random_forest.fit(train_wd[input_cols],train_wd[target_cols])
```

[53]:
```
RandomForestClassifier(max_depth=6, max_features=None, max_leaf_nodes=9,
                       n_estimators=50)
```

[54]:
```
train_preds3 = random_forest.predict(train_wd[input_cols])
```

[55]:
```
accuracy_score(train_wd[target_cols],train_preds3)
```

[55]:
```
0.7871362940275651
```

[56]:
```
test_preds3 = random_forest.predict(test_wd[input_cols])
```

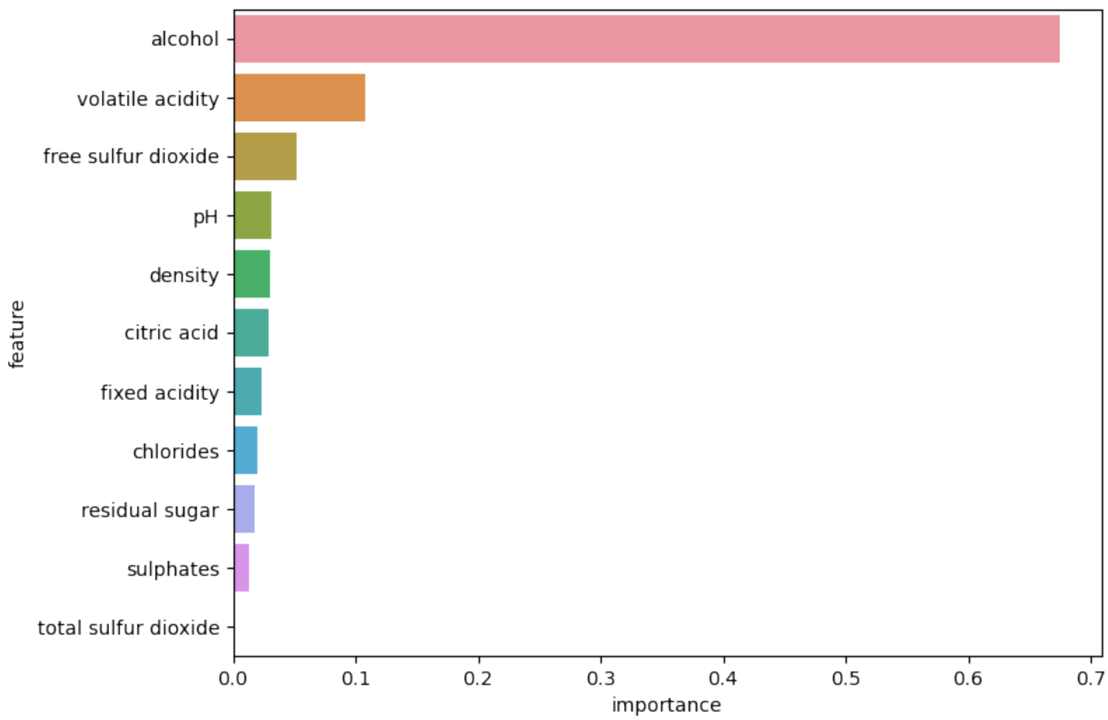
[57]:
```
accuracy_score(test_preds3,test_wd[target_cols])
```

[57]:
```
0.773469387755102
```

[58]:
```
importance_df = pd.DataFrame({
    'feature':input_cols,
    'importance': random_forest.feature_importances_
}).sort_values('importance',ascending = False)
```

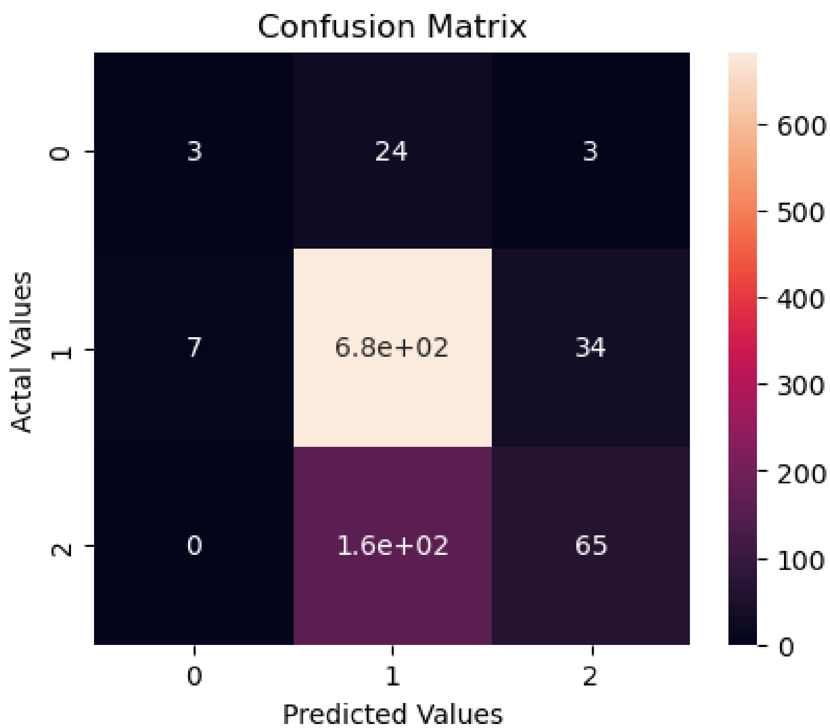
[59]:
```
sns.barplot(data=importance_df,x = 'importance',y = 'feature')
```

[59]:
```
<AxesSubplot:xlabel='importance', ylabel='feature'>
```



[60]:
```
## confusion matrix
confusion_matrix = metrics.confusion_matrix(test_wd.quality, test_preds2)
confusion_matrix
cm_df = pd.DataFrame(confusion_matrix,
                     index = [0,1,2],
                     columns = [0,1,2])
```

[61]:
```
plt.figure(figsize=(5,4))
sns.heatmap(cm_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



[62]:
```
print(classification_report(test_wd.quality, test_preds3))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        30
           1       0.78      0.96      0.86       723
           2       0.68      0.30      0.41       227

    accuracy                           0.77       980
   macro avg       0.49      0.42      0.42       980
weighted avg       0.74      0.77      0.73       980
```

```
C:\Users\DeLL\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

C:\Users\DeLL\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
```

```
C:\Users\DeLL\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
```

[63]:
```python
from sklearn.svm import SVC
```

[64]:
```python
parameters = {'C':[0.1,1,10,100,1000],
              'gamma':[1,0.1,0.01,0.001,0.0001],
              'kernel':['rbf','linear','polynomial']}
```

[65]:
```python
grid = GridSearchCV(SVC(),parameters,refit = True,verbose=3)
grid.fit(train_wd[input_cols],train_wd[target_cols])
```

```
Fitting 5 folds for each of 75 candidates, totalling 375 fits
[CV 1/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.749 total time=   0.3s
[CV 2/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.749 total time=   0.3s
[CV 3/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.747 total time=   0.3s
[CV 4/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.748 total time=   0.4s
[CV 5/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.748 total time=   0.3s
[CV 1/5] END …C=0.1, gamma=1, kernel=linear;, score=0.749 total time=   0.1s
[CV 2/5] END …C=0.1, gamma=1, kernel=linear;, score=0.749 total time=   0.1s
[CV 3/5] END …C=0.1, gamma=1, kernel=linear;, score=0.747 total time=   0.1s
[CV 4/5] END …C=0.1, gamma=1, kernel=linear;, score=0.748 total time=   0.1s
[CV 5/5] END …C=0.1, gamma=1, kernel=linear;, score=0.748 total time=   0.1s
[CV 1/5] END …C=0.1, gamma=1, kernel=polynomial;, score=nan total time=   0.0s
[CV 2/5] END …C=0.1, gamma=1, kernel=polynomial;, score=nan total time=   0.0s
[CV 3/5] END …C=0.1, gamma=1, kernel=polynomial;, score=nan total time=   0.0s
[CV 4/5] END …C=0.1, gamma=1, kernel=polynomial;, score=nan total time=   0.0s
[CV 5/5] END …C=0.1, gamma=1, kernel=polynomial;, score=nan total time=   0.0s
[CV 1/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.749 total time=   0.3s
[CV 2/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.749 total time=   0.3s
[CV 3/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.747 total time=   0.3s
[CV 4/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.748 total time=   0.3s
[CV 5/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.748 total time=   0.3s
[CV 1/5] END …C=0.1, gamma=0.1, kernel=linear;, score=0.749 total time=   0.2s
[CV 2/5] END …C=0.1, gamma=0.1, kernel=linear;, score=0.749 total time=   0.1s
[CV 3/5] END …C=0.1, gamma=0.1, kernel=linear;, score=0.747 total time=   0.1s
[CV 4/5] END …C=0.1, gamma=0.1, kernel=linear;, score=0.748 total time=   0.1s
[CV 5/5] END …C=0.1, gamma=0.1, kernel=linear;, score=0.748 total time=   0.1s
[CV 1/5] END .C=0.1, gamma=0.1, kernel=polynomial;, score=nan total time=   0.0s
[CV 2/5] END .C=0.1, gamma=0.1, kernel=polynomial;, score=nan total time=   0.0s
[CV 3/5] END .C=0.1, gamma=0.1, kernel=polynomial;, score=nan total time=   0.0s
[CV 4/5] END .C=0.1, gamma=0.1, kernel=polynomial;, score=nan total time=   0.0s
[CV 5/5] END .C=0.1, gamma=0.1, kernel=polynomial;, score=nan total time=   0.0s
```

```
 0.77718352 0.74834102        nan 0.74910633 0.74834102        nan
 0.74834102 0.74834102        nan 0.74834102 0.74834102        nan
 0.74834102 0.74834102        nan 0.78432963 0.74834102        nan
 0.7725881  0.74834102        nan 0.74834102 0.74834102        nan
 0.74834102 0.74834102        nan 0.74834102 0.74834102        nan
 0.78688228 0.74834102        nan 0.77794882 0.74834102        nan
 0.74987294 0.74834102        nan 0.74834102 0.74834102        nan
 0.74834102 0.74834102        nan]
```

[65]:
```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf', 'linear', 'polynomial']},
             verbose=3)
```

[66]:
```
grid.best_params_
```

[66]:
```
{'C': 1000, 'gamma': 1, 'kernel': 'rbf'}
```

[67]:
```
svm_classifier = SVC(C=1000,gamma = 1 , kernel = 'rbf')
svm_classifier.fit(train_wd[input_cols],train_wd[target_cols])
```

[67]:
```
SVC(C=1000, gamma=1)
```

[68]:
```
train_preds4 = svm_classifier.predict(train_wd[input_cols])
```

[69]:
```
accuracy_score(train_preds4,train_wd[target_cols])
```

[69]:
```
0.8208269525267994
```

[70]:
```
test_preds4 = svm_classifier.predict(test_wd[input_cols])
```

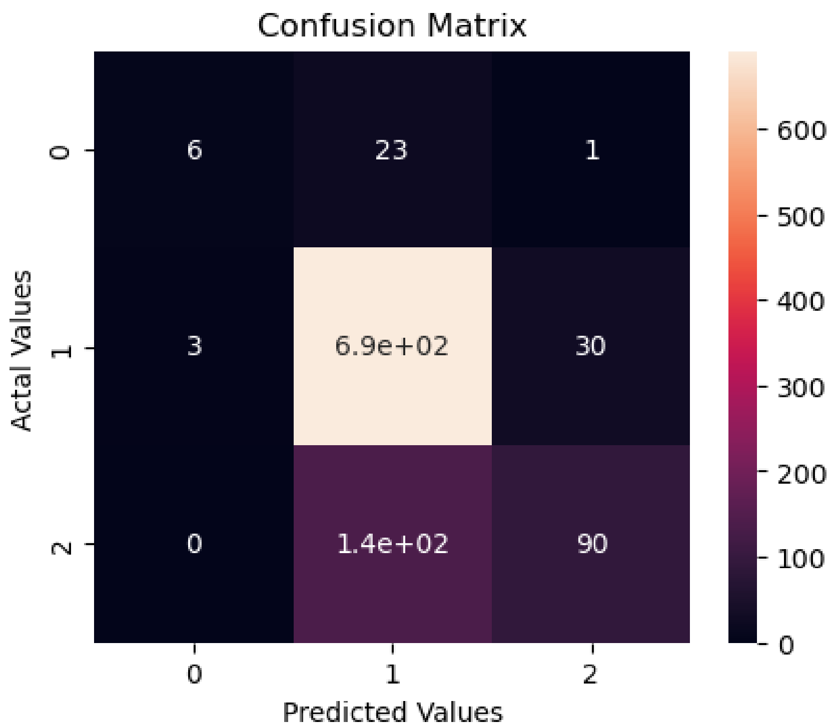
[71]:
```
accuracy_score(test_preds4,test_wd[target_cols])
```

[71]:
```
0.8020408163265306
```

[72]:
```
## confusion matrix
confusion_matrix = metrics.confusion_matrix(test_wd.quality, test_preds4)
confusion_matrix
cm_df = pd.DataFrame(confusion_matrix,
                     index = [0,1,2],
                     columns = [0,1,2])
```

[73]:
```
plt.figure(figsize=(5,4))
sns.heatmap(cm_df, annot=True)
plt.title('Confusion Matrix')
```

```
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



[74]:
```
print(classification_report(test_wd.quality, test_preds4))
```

```
              precision    recall  f1-score   support

           0       0.67      0.20      0.31        30
           1       0.81      0.95      0.88       723
           2       0.74      0.40      0.52       227

    accuracy                           0.80       980
   macro avg       0.74      0.52      0.57       980
weighted avg       0.79      0.80      0.78       980
```

[75]:
```
####conclusion
print('Accuracy of Logistic regression model:'
      ,round(accuracy_score(test_preds1,test_wd[target_cols])*100,2),'%')
print('Accuracy of Decision Tree model:'
      ,round(accuracy_score(test_preds2,test_wd[target_cols])*100,2),'%')
print('Accuracy of Random forest model:'
```

```
      ,round(accuracy_score(test_preds3,test_wd[target_cols])*100,2),'%')
print('Accuracy of SVM model:'
      ,round(accuracy_score(test_preds4,test_wd[target_cols])*100,2),'%')
```

```
Accuracy of Logistic regression model: 76.22 %
Accuracy of Decision Tree model: 76.53 %
Accuracy of Random forest model: 77.35 %
Accuracy of SVM model: 80.2 %
```