In [1]:
```
#importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#load the dataset
df =pd.read_csv(r"C:\Users\anike\Downloads\wheat (1).csv")
```

In [3]:
```
# read the data
df
```

Out[3]:

	area	perimeter	compactness	length	width	asymmetry coefficient	groove length	category
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1.0
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1.0
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1.0
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1.0
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1.0
...	...	...	...	...	...	...	...	...
205	12.19	13.20	0.8783	5.137	2.981	3.631	4.870	3.0
206	11.23	12.88	0.8511	5.140	2.795	4.325	5.003	3.0
207	13.20	13.66	0.8883	5.236	3.232	8.315	5.056	3.0
208	11.84	13.21	0.8521	5.175	2.836	3.598	5.044	3.0
209	12.30	13.34	0.8684	5.243	2.974	5.637	5.063	3.0

210 rows × 8 columns

Info about data

In [4]:
```
# to check no of rows and col
df.shape
```

Out[4]: (210, 8)

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

Out[5]:

	area	perimeter	compactness	length	width	asymmetry coefficient	groove length
count	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000
mean	14.847524	14.559286	0.870999	5.628533	3.258605	3.700201	5.408071
std	2.909699	1.305959	0.023629	0.443063	0.377714	1.503557	0.491480
min	10.590000	12.410000	0.808100	4.899000	2.630000	0.765100	4.519000
25%	12.270000	13.450000	0.856900	5.262250	2.944000	2.561500	5.045000
50%	14.355000	14.320000	0.873450	5.523500	3.237000	3.599000	5.223000
75%	17.305000	15.715000	0.887775	5.979750	3.561750	4.768750	5.877000
max	21.180000	17.250000	0.918300	6.675000	4.033000	8.456000	6.550000

In [6]:
```
# to check the dtype, column name , not null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210 entries, 0 to 209
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   area                   210 non-null    float64
 1   perimeter              210 non-null    float64
 2   compactness            210 non-null    float64
 3   length                 210 non-null    float64
 4   width                  210 non-null    float64
 5   asymmetry coefficient  210 non-null    float64
 6   groove length          210 non-null    float64
 7   category               210 non-null    float64
dtypes: float64(8)
memory usage: 13.3 KB
```

In [7]:
```
# top 5 rows of dataset
df.head()
```

Out[7]:

	area	perimeter	compactness	length	width	asymmetry coefficient	groove length	category
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1.0
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1.0
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1.0
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1.0
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1.0

In [8]:
```
# bottom 5 rows of dataset
df.tail()
```

Out[8]:

	area	perimeter	compactness	length	width	asymmetry coefficient	groove length	category
205	12.19	13.20	0.8783	5.137	2.981	3.631	4.870	3.0
206	11.23	12.88	0.8511	5.140	2.795	4.325	5.003	3.0
207	13.20	13.66	0.8883	5.236	3.232	8.315	5.056	3.0
208	11.84	13.21	0.8521	5.175	2.836	3.598	5.044	3.0
209	12.30	13.34	0.8684	5.243	2.974	5.637	5.063	3.0

In [9]:
```
#any null value present
df.isnull().values.any()

```

Out[9]: False

In [10]:
```
# to view overall null value
df.isnull()
```

Out[10]:

	area	perimeter	compactness	length	width	asymmetry coefficient	groove length	category
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...
205	False	False	False	False	False	False	False	False
206	False	False	False	False	False	False	False	False
207	False	False	False	False	False	False	False	False
208	False	False	False	False	False	False	False	False
209	False	False	False	False	False	False	False	False

210 rows × 8 columns

In [11]:
```
df['category'] = df['category'].astype(float)
```

In [12]:
```
1 df.dtypes
```

Out[12]:
```
area                    float64
perimeter               float64
compactness             float64
length                  float64
width                   float64
asymmetry coefficient   float64
groove length           float64
category                float64
dtype: object
```

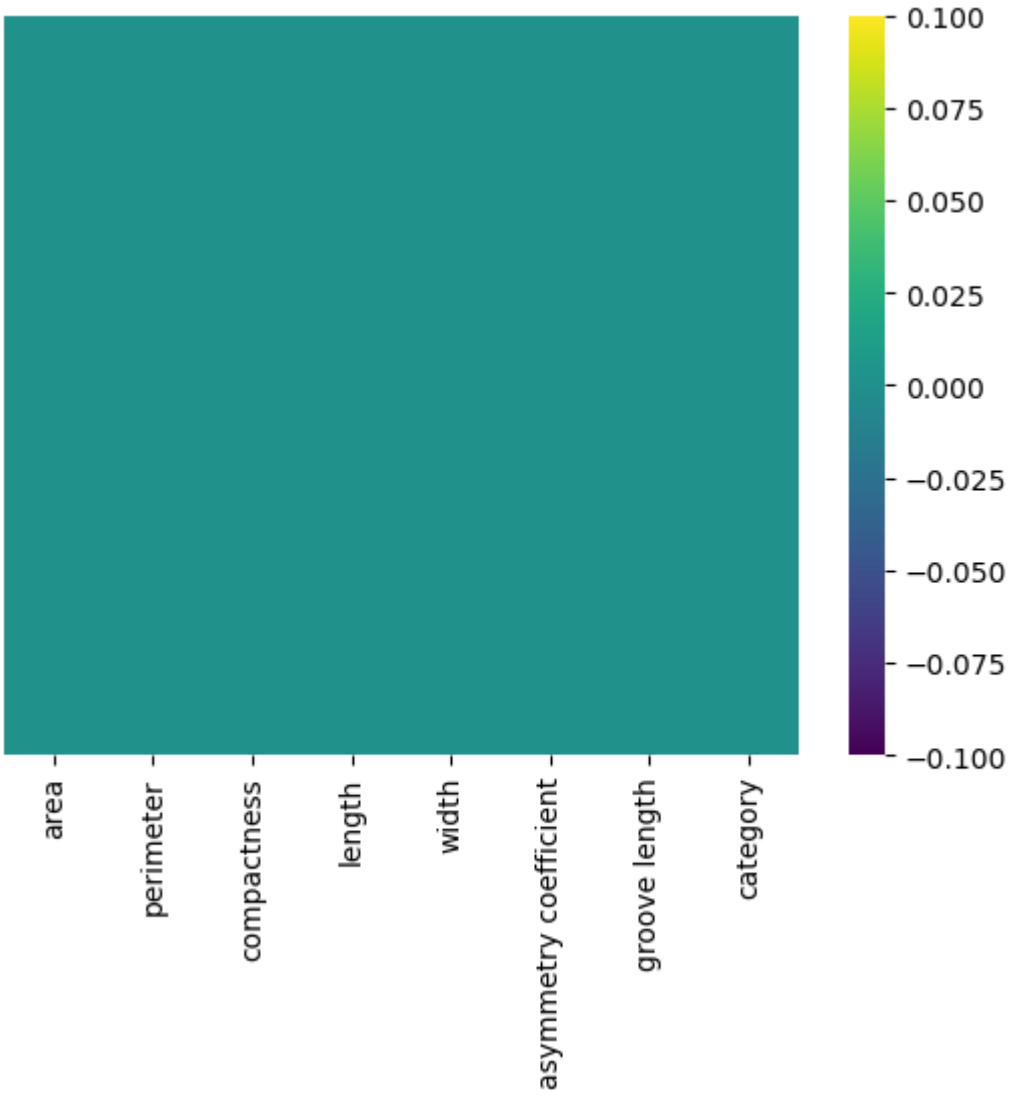
In [13]:
```
1 df.dtypes
```

Out[13]:
```
area                    float64
perimeter               float64
compactness             float64
length                  float64
width                   float64
asymmetry coefficient   float64
groove length           float64
category                float64
dtype: object
```

In [14]:
```
1 # to check how many null value are present acc to col
2 df.isnull().sum()
```

Out[14]:
```
area                    0
perimeter               0
compactness             0
length                  0
width                   0
asymmetry coefficient   0
groove length           0
category                0
dtype: int64
```

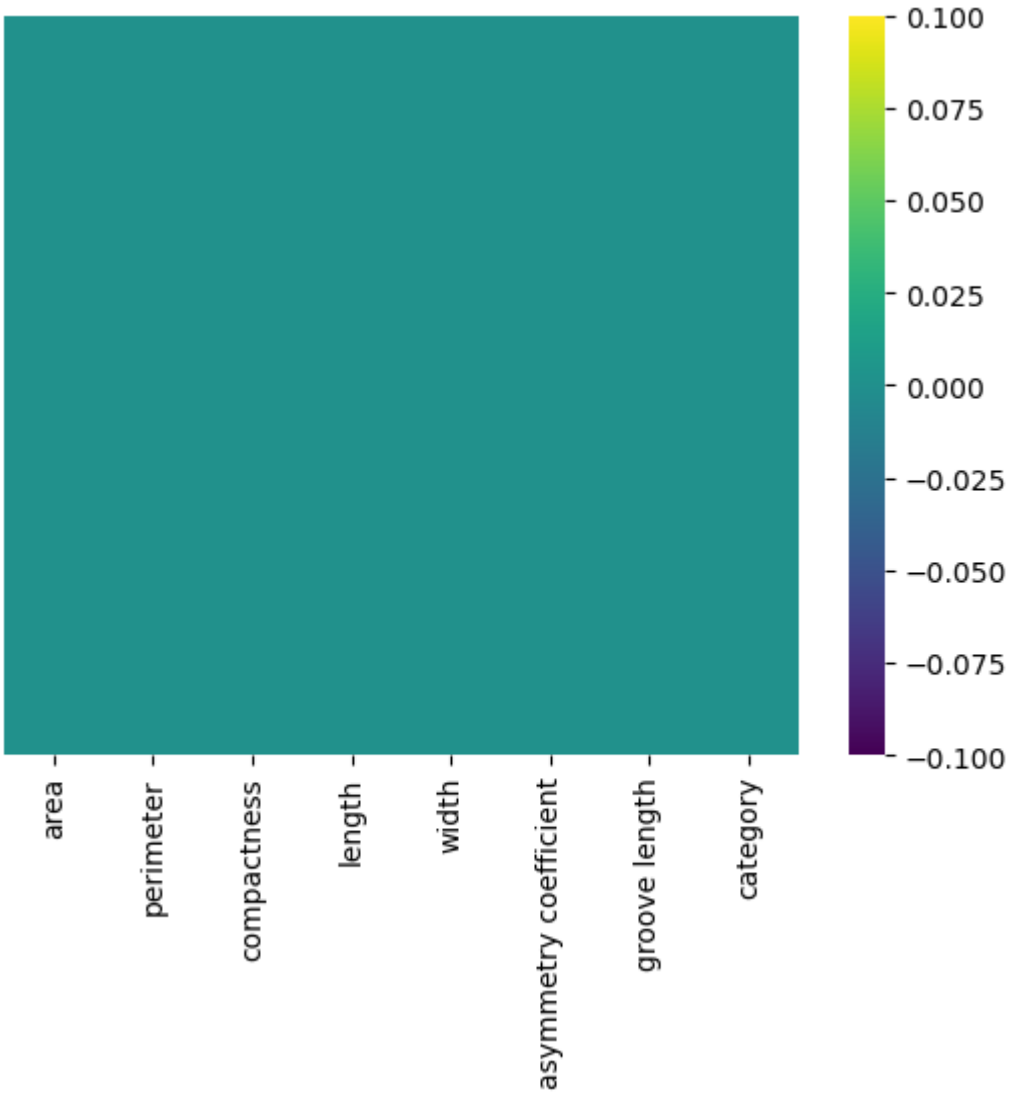
In [15]:
```
# to plot the missing values
sns.heatmap(df.isnull(), yticklabels=False,cmap="viridis")
plt.show()
```



Treating missing values

In [16]:
```
df['area'].bfill(axis=0,inplace=True)
df['perimeter'].ffill(axis=0,inplace=True)
df['compactness']=df['compactness'].fillna(df['compactness'].mean())
df['length'].bfill(axis=0,inplace=True)
df['width'].ffill(axis=0,inplace=True)
df['asymmetry coefficient'].bfill(axis=0,inplace=True)
df['groove length'].ffill(axis=0,inplace=True)
```

In [17]:
```
# to plot heatmap after treating missing value
sns.heatmap(df.isnull(), yticklabels=False,cmap="viridis")
plt.show()
```



In [18]:
```
# number of null values in each column after treating missing value
df.isnull().sum()
```

Out[18]:
```
area                    0
perimeter               0
compactness             0
length                  0
width                   0
asymmetry coefficient   0
groove length           0
category                0
dtype: int64
```

In [19]:
```
df['category'].value_counts()
```

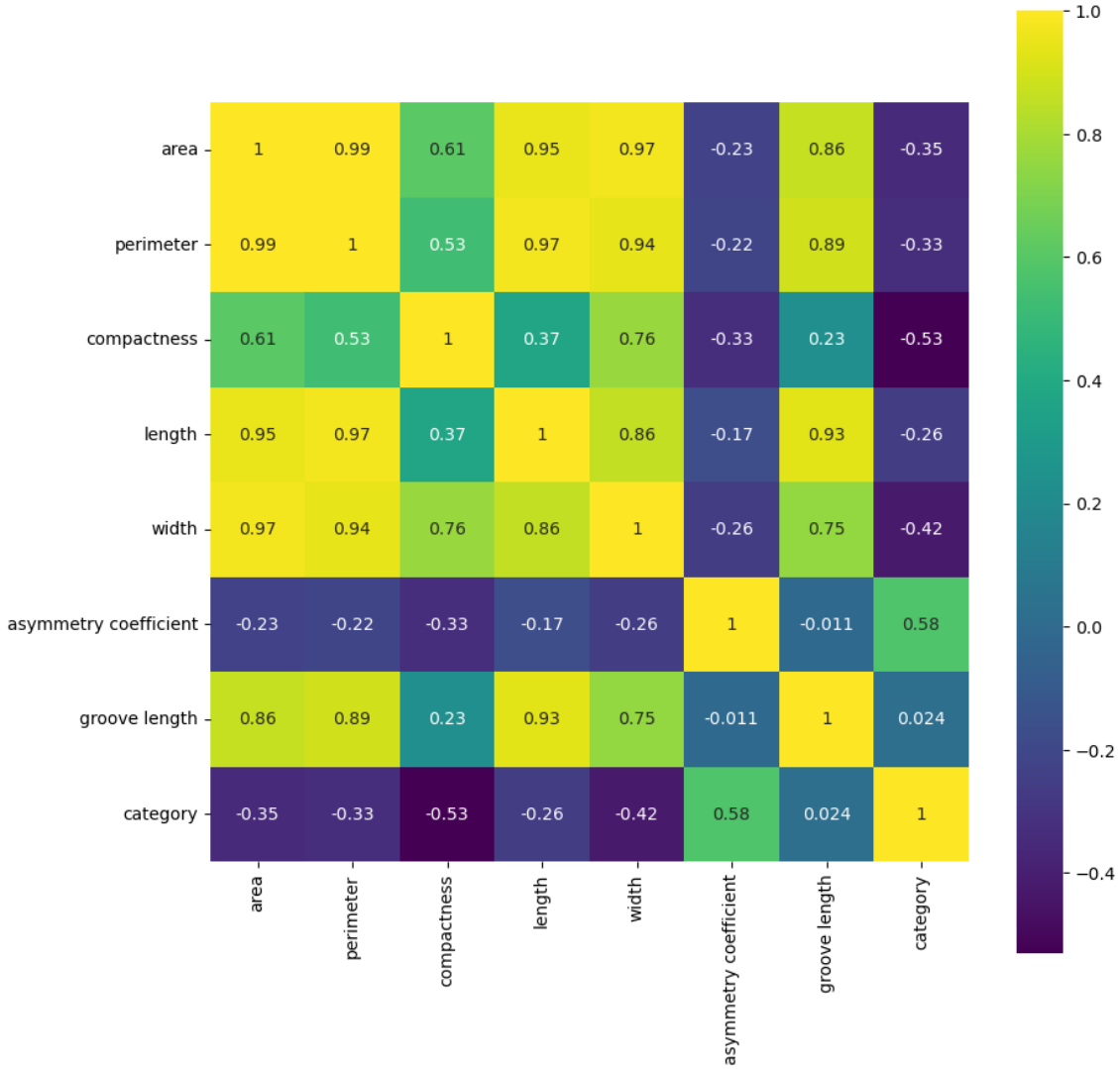
Out[19]:
```
1.0    70
2.0    70
3.0    70
Name: category, dtype: int64
```

In [20]:
```
1 df.columns
```

Out[20]:
```
Index(['area', 'perimeter', 'compactness', 'length', 'width',
       'asymmetry coefficient', 'groove length', 'category'],
      dtype='object')
```

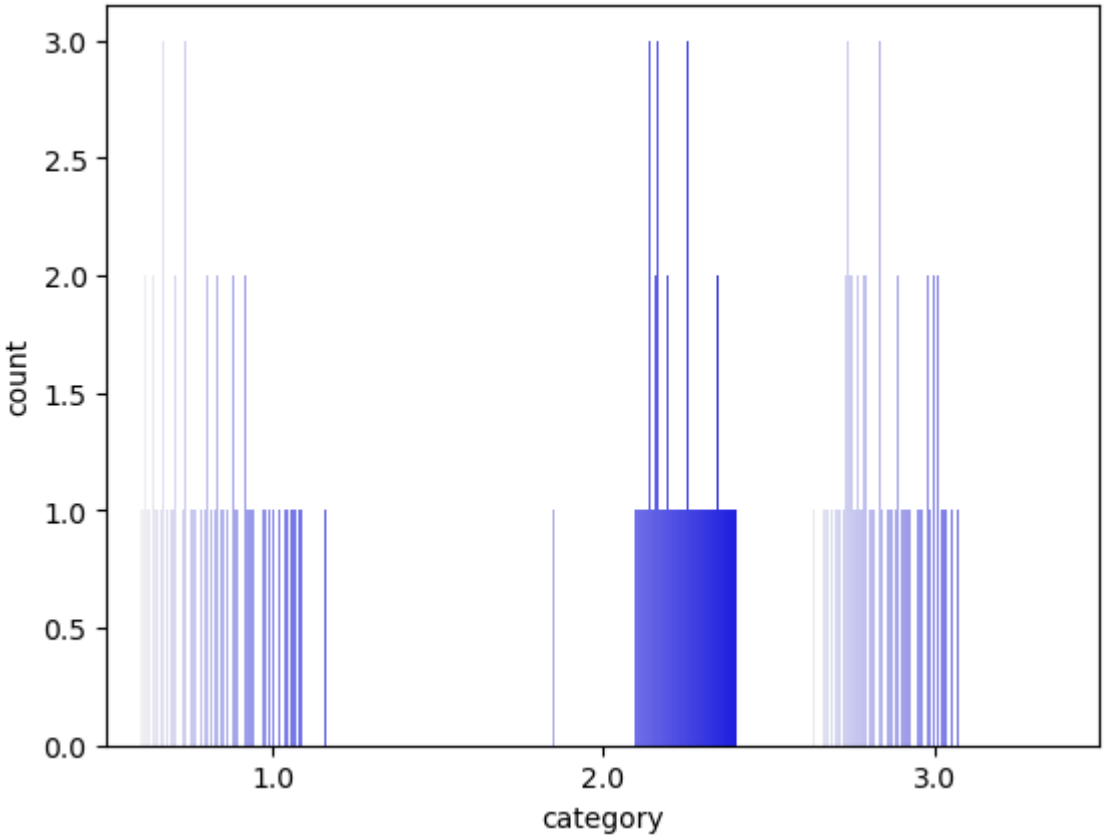
Data Visualization

In [21]:
```
1 #Examaning a correlation matrix of all the features
2 corrmat = df.corr()
3 plt.subplots(figsize=(10,10))
4 sns.heatmap(corrmat,cmap="viridis", square=True,annot=True)
5 plt.show()
```



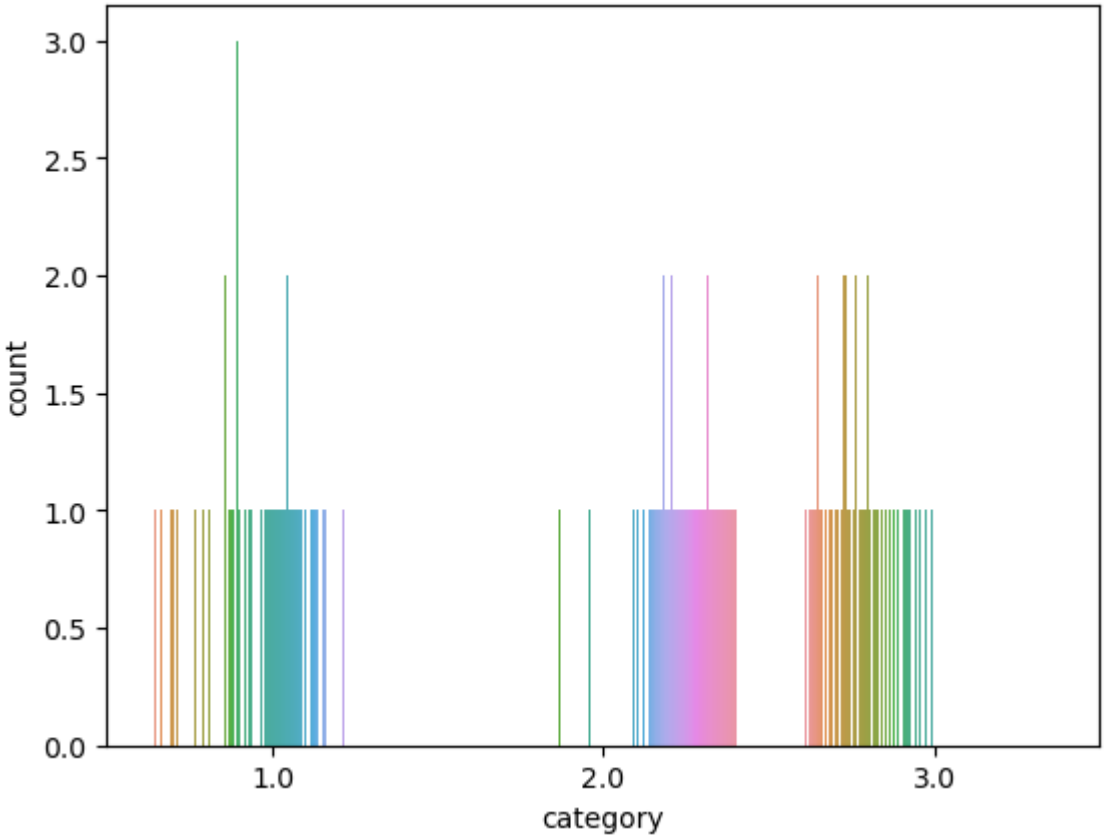
In [22]:

```
1
2 ax = sns.countplot(x=df['category'], hue=df['groove length'], color='bl
3
4 # Remove the legend
5 ax.get_legend().remove()
6
7 plt.show()
```



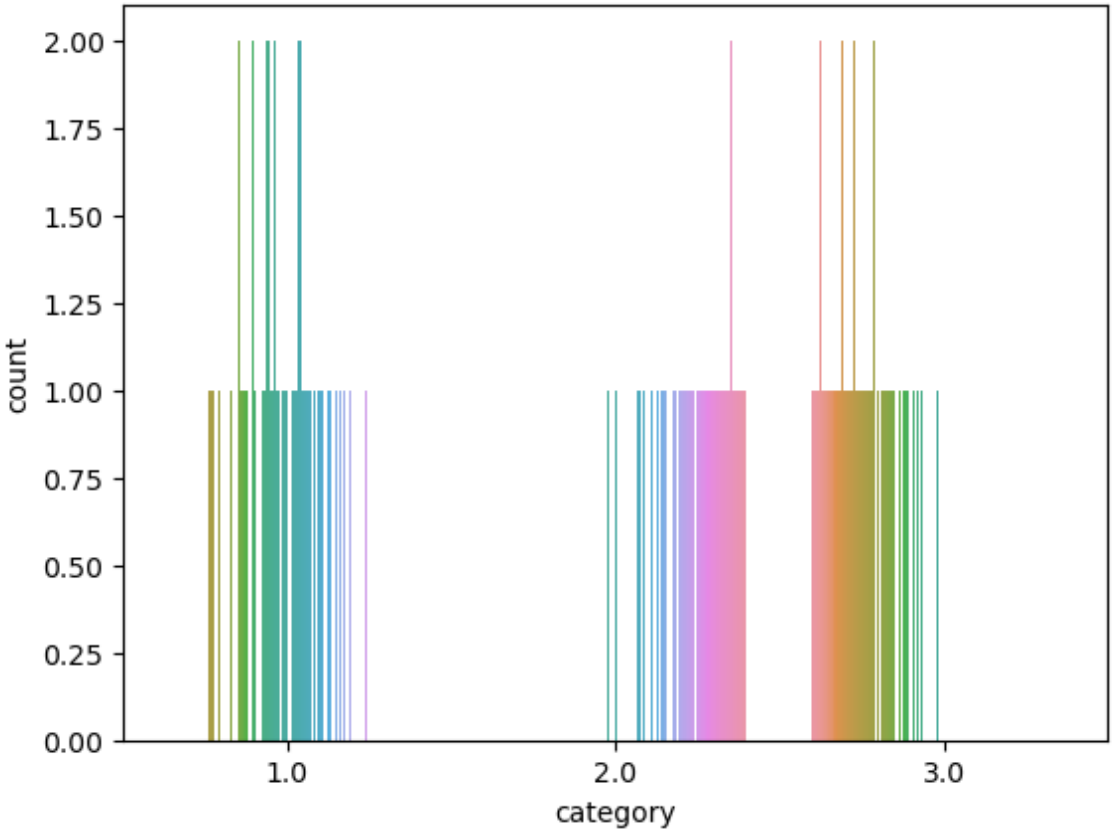
In [23]:

```
ax = sns.countplot(x=df['category'], hue=df['length'],width=0.8)

# Remove the legend
ax.get_legend().remove()

plt.show()
```



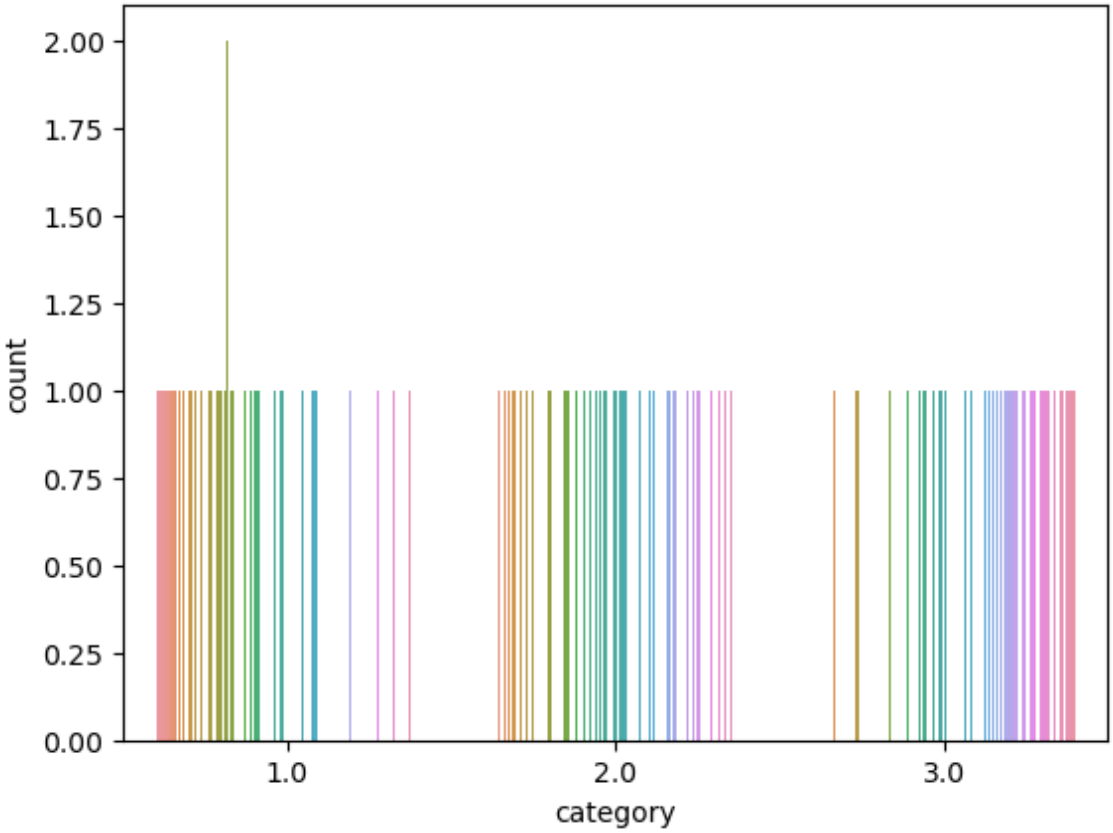
In [24]:

```
1
2 ax = sns.countplot(x=df['category'], hue=df['width'],width=0.8)
3
4 # Remove the legend
5 ax.get_legend().remove()
6
7 plt.show()
```



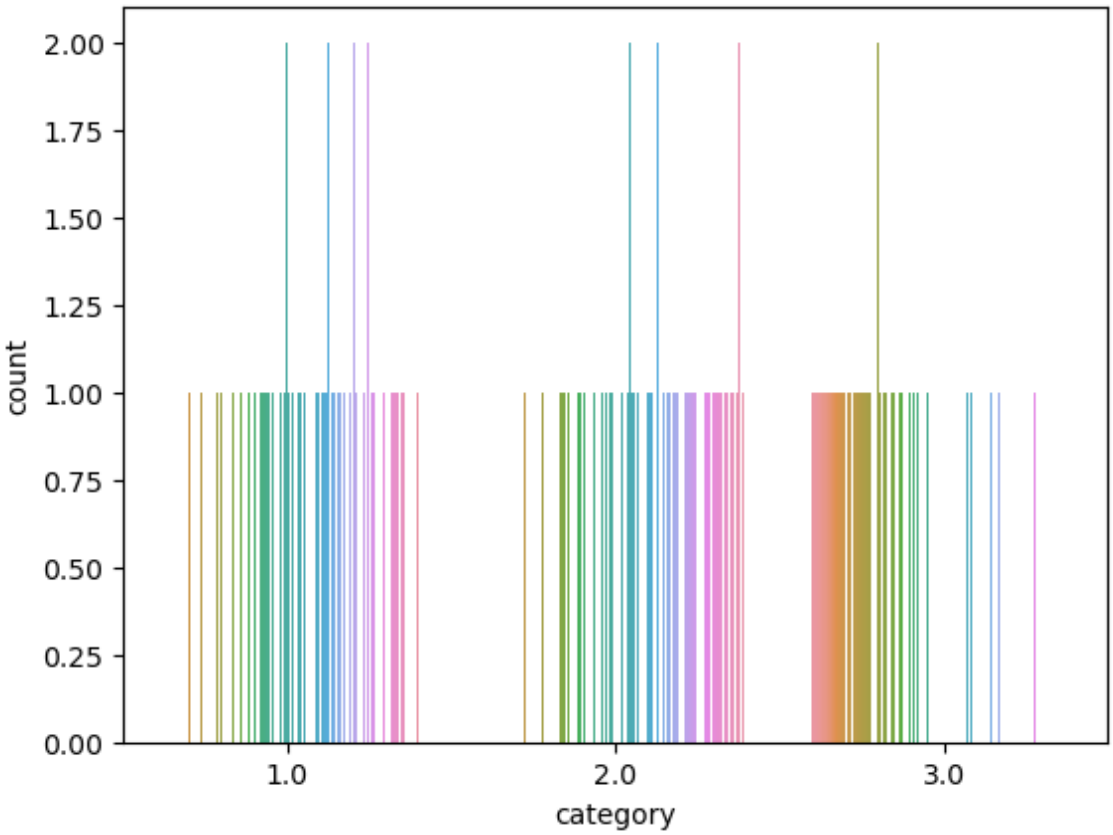
In [25]:

```
1
2 ax = sns.countplot(x=df['category'], hue=df['asymmetry coefficient'],wi
3
4 # Remove the legend
5 ax.get_legend().remove()
6
7 plt.show()
```

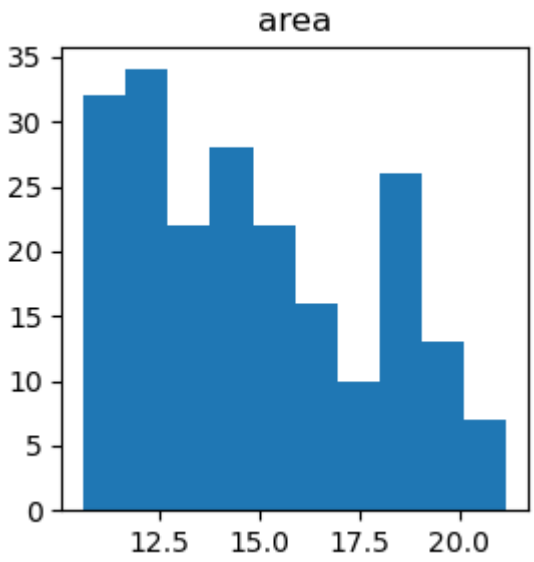


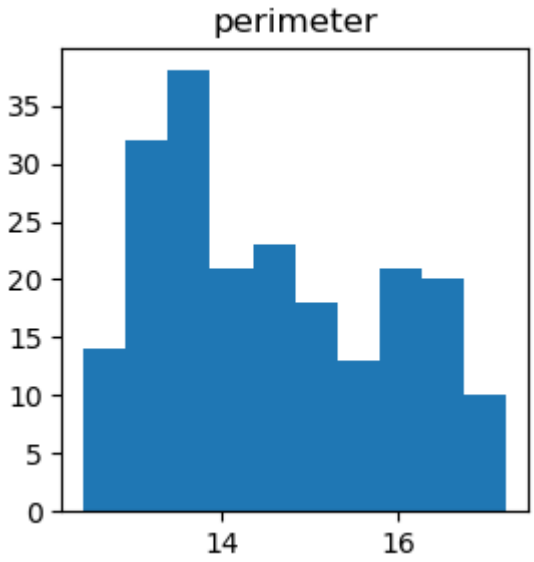
In [26]:

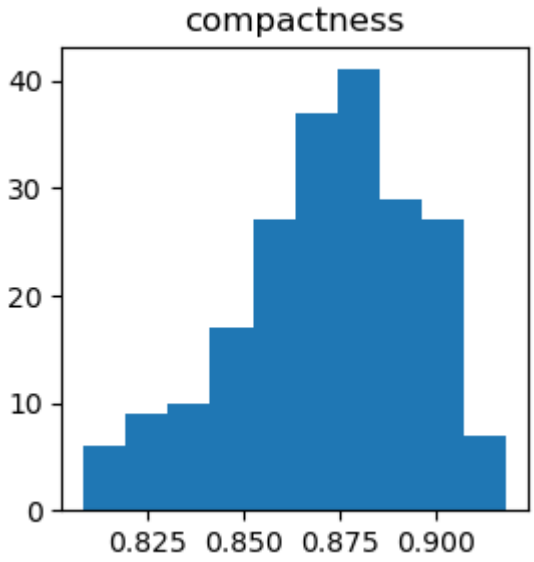
```
ax = sns.countplot(x=df['category'], hue=df['compactness'],width=0.8)

# Remove the Legend
ax.get_legend().remove()

plt.show()
```

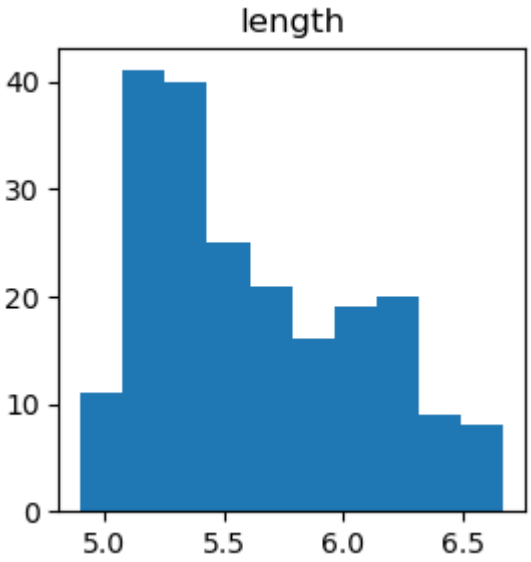


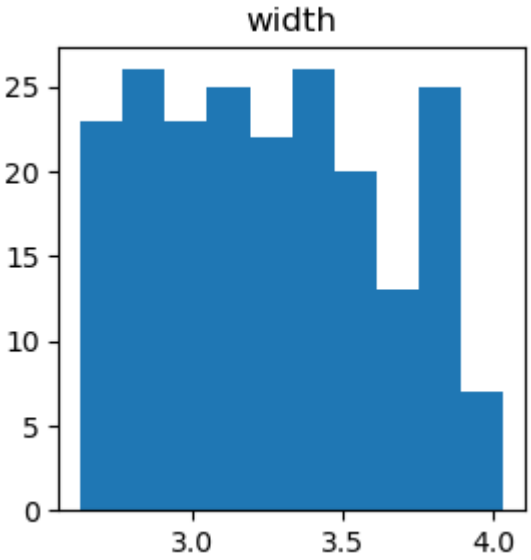
In [27]:
```
column=['area','perimeter','compactness','length','width','asymmetry co
for category in column:
    plt.figure(figsize=(3,3))
    plt.hist(df[category])
    plt.title(category)
    plt.show()
```

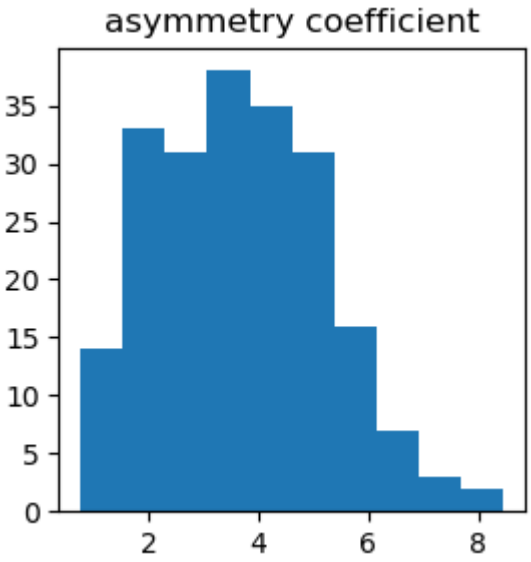


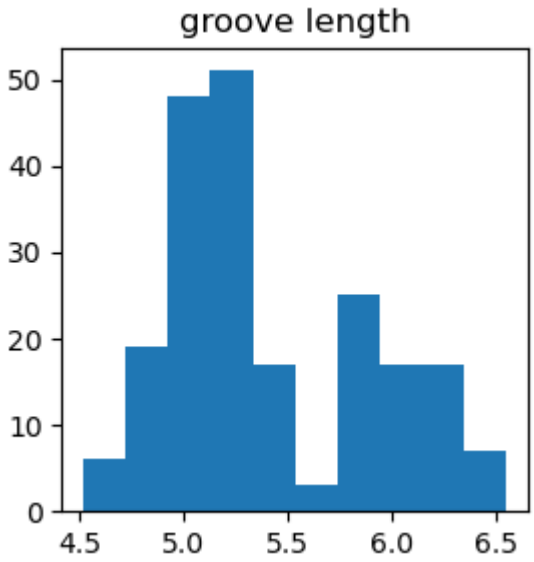




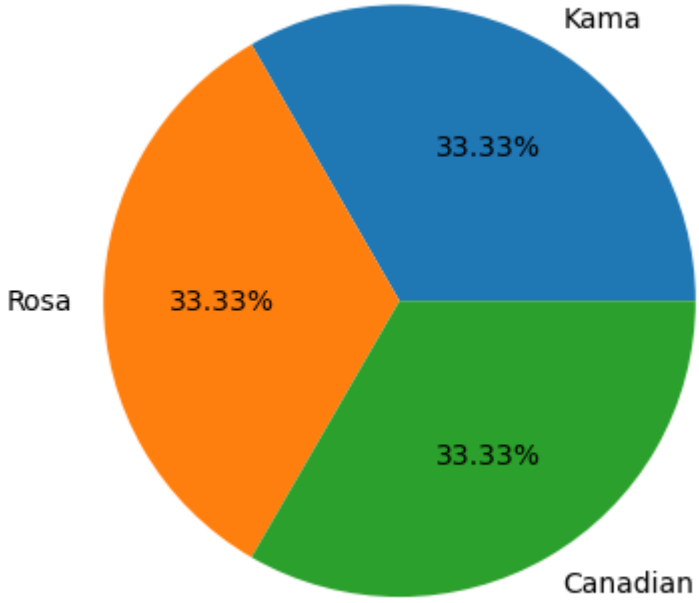

length
40
30
20
10
0
5.0
5.5
6.0
6.5


width
25
20
15
10
5
0
3.0
3.5
4.0


asymmetry coefficient
35
30
25
20
15
10
5
0
2
4
6
8

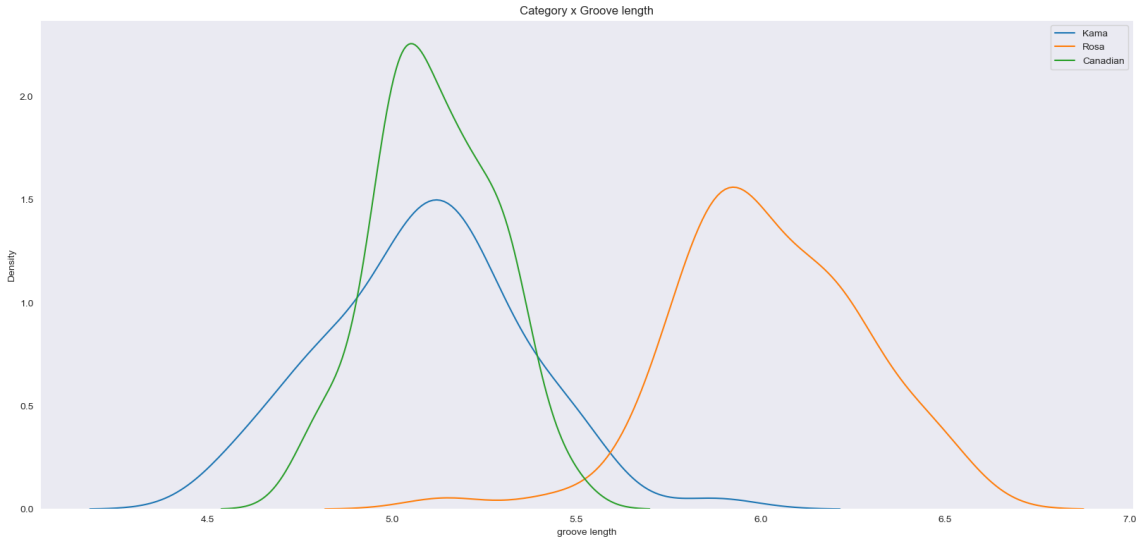


In [28]:
```
1 category_name=['Kama', 'Rosa', 'Canadian']
2 plt.pie(df['category'].value_counts(),labels=category_name,autopct="%.2
3 plt.show()
```



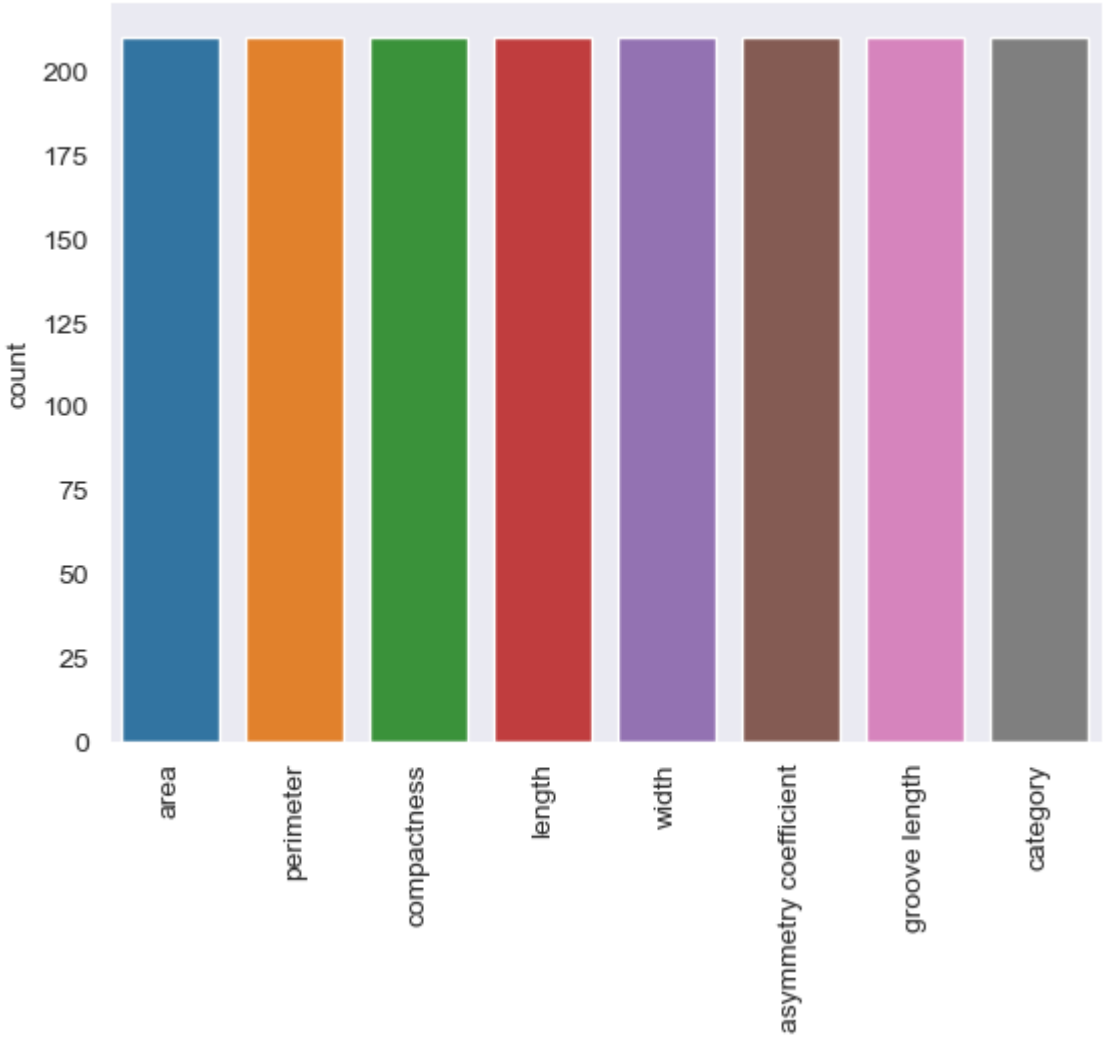
In [29]:

```
# Default distribution according to annual salaries
fig = plt.figure(figsize = (20, 9))
sns.set_style("dark")
sns.kdeplot(df[df['category']==1]['groove length'])
sns.kdeplot(df[df['category']==2]['groove length'])
sns.kdeplot(df[df['category']==3]['groove length'])
plt.title('Category x Groove length')
plt.legend(labels=['Kama', 'Rosa','Canadian'])
plt.show()
```

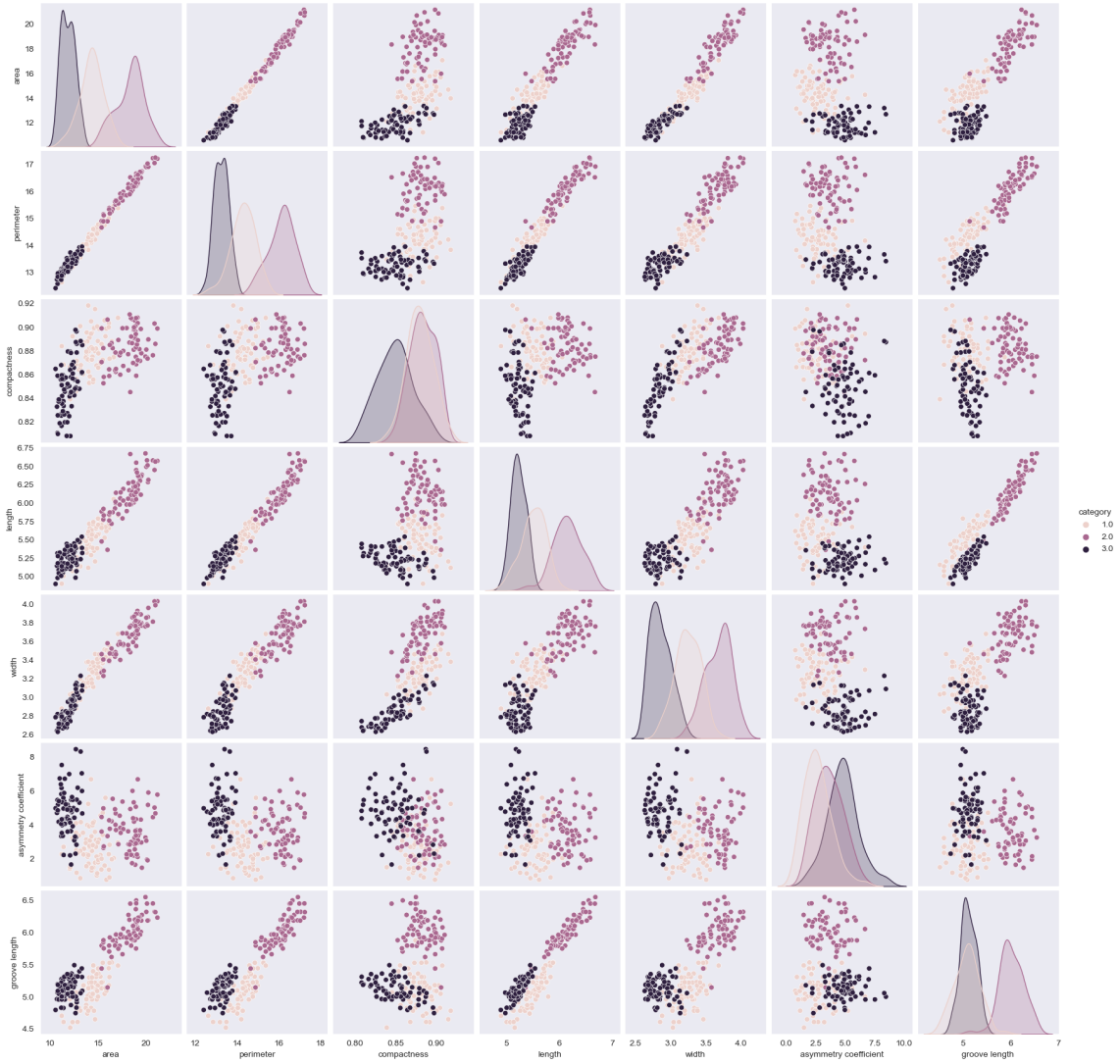


In [30]:
```
1 sns.countplot(df)
2 plt.xticks(rotation=90)
```

Out[30]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7]),
 [Text(0, 0, 'area'),
  Text(1, 0, 'perimeter'),
  Text(2, 0, 'compactness'),
  Text(3, 0, 'length'),
  Text(4, 0, 'width'),
  Text(5, 0, 'asymmetry coefficient'),
  Text(6, 0, 'groove length'),
  Text(7, 0, 'category')])
```

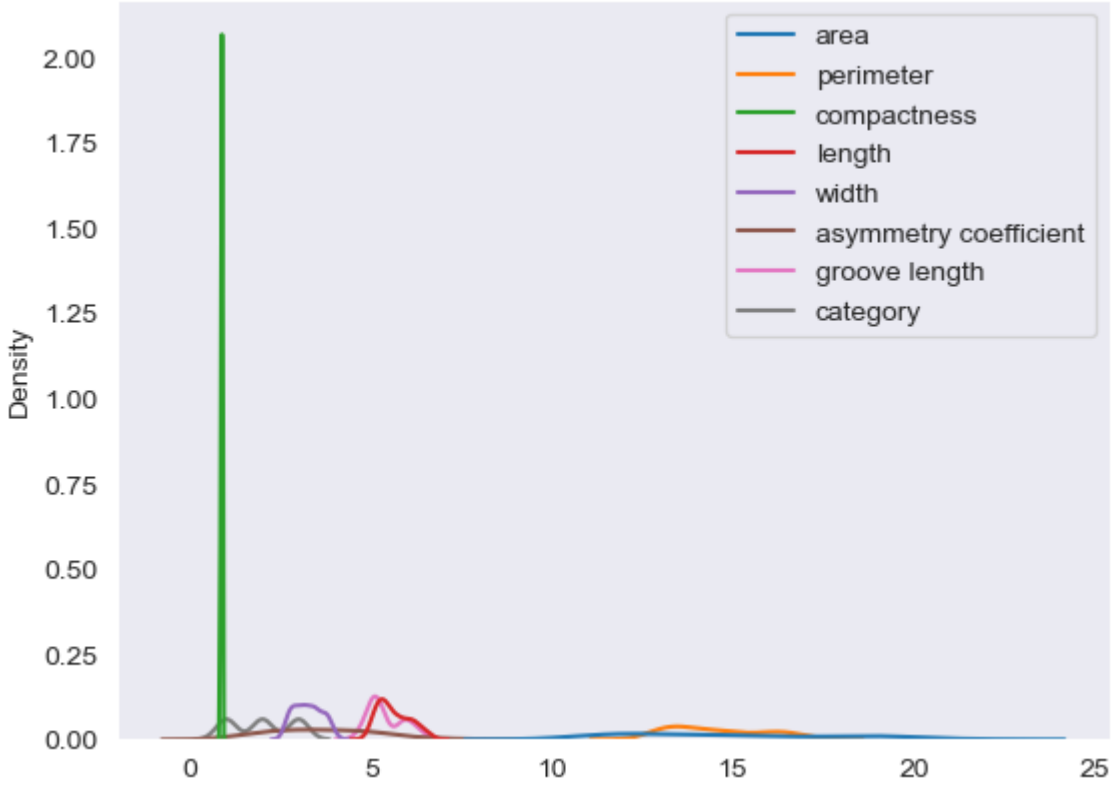


In [31]:
```
1 sns.pairplot(df, hue="category")
2 plt.show()
3
```



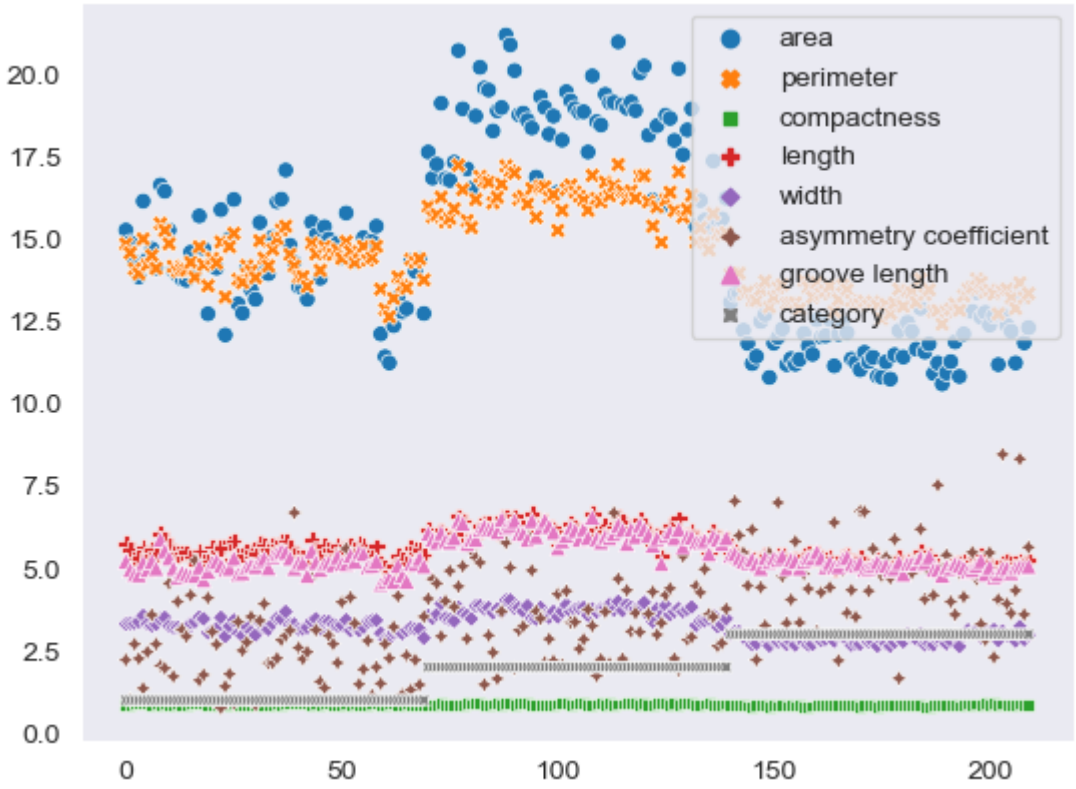
In [32]:
```
1 sns.kdeplot(df)
```

Out[32]: <Axes: ylabel='Density'>



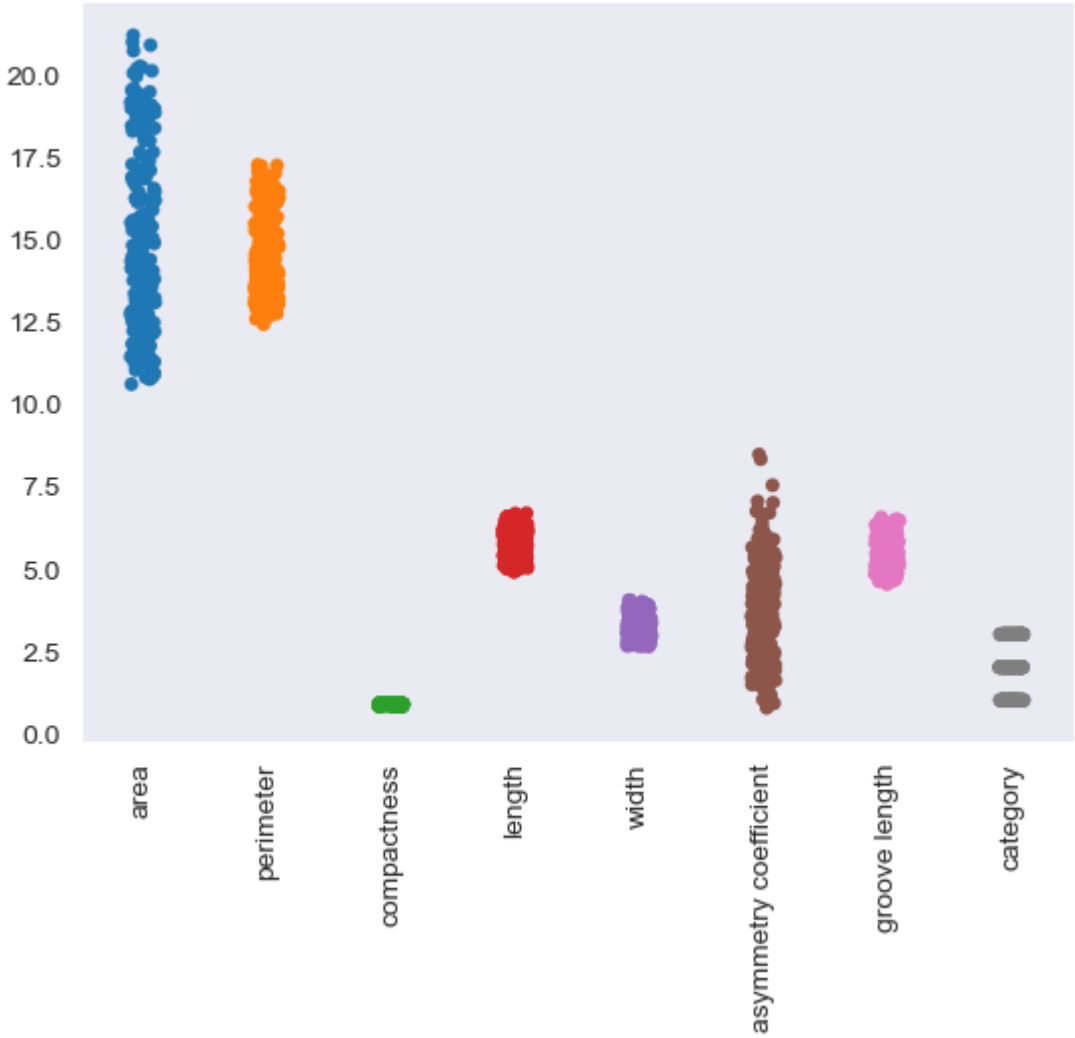
In [33]:
```
1 sns.scatterplot(df)
```

Out[33]: <Axes: >



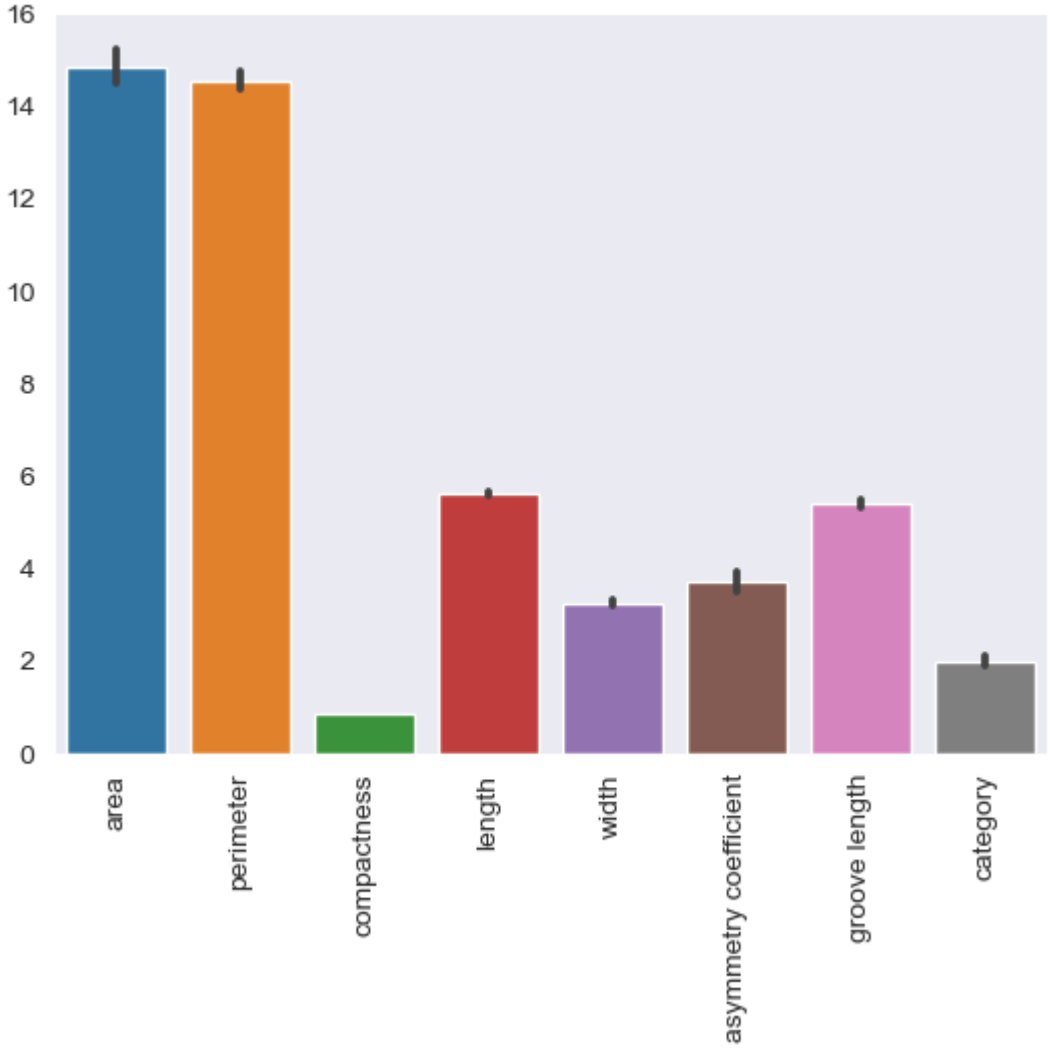
In [34]:
```
1 sns.stripplot(df)
2 plt.xticks(rotation=90)
```

Out[34]:
```
([0, 1, 2, 3, 4, 5, 6, 7],
 [Text(0, 0, 'area'),
  Text(1, 0, 'perimeter'),
  Text(2, 0, 'compactness'),
  Text(3, 0, 'length'),
  Text(4, 0, 'width'),
  Text(5, 0, 'asymmetry coefficient'),
  Text(6, 0, 'groove length'),
  Text(7, 0, 'category')])
```



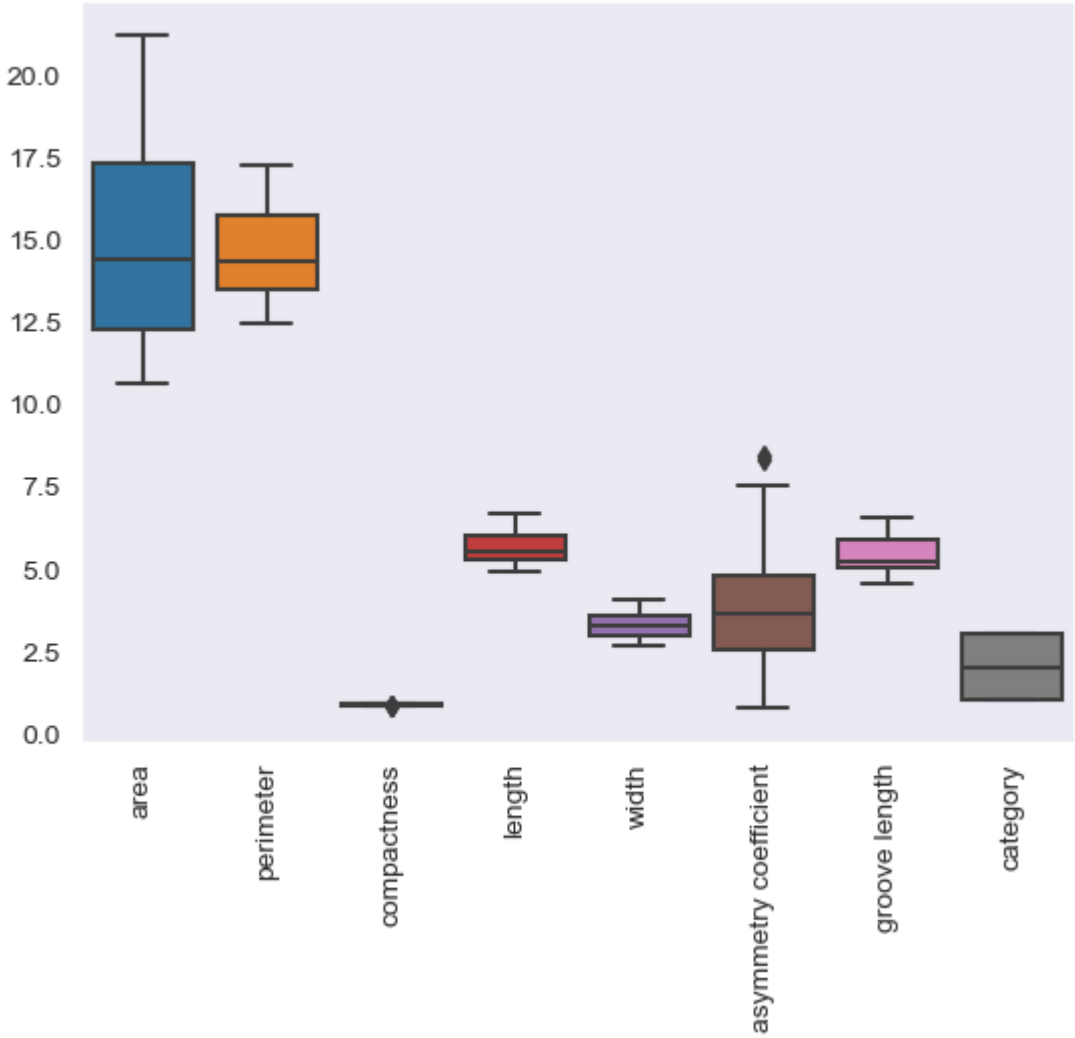
In [35]:
```
1 sns.barplot(df)
2 plt.xticks(rotation=90)
```

Out[35]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7]),
 [Text(0, 0, 'area'),
  Text(1, 0, 'perimeter'),
  Text(2, 0, 'compactness'),
  Text(3, 0, 'length'),
  Text(4, 0, 'width'),
  Text(5, 0, 'asymmetry coefficient'),
  Text(6, 0, 'groove length'),
  Text(7, 0, 'category')])
```



In [36]:
```
1 sns.boxplot(df)
2 plt.xticks(rotation=90)
```

Out[36]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7]),
 [Text(0, 0, 'area'),
  Text(1, 0, 'perimeter'),
  Text(2, 0, 'compactness'),
  Text(3, 0, 'length'),
  Text(4, 0, 'width'),
  Text(5, 0, 'asymmetry coefficient'),
  Text(6, 0, 'groove length'),
  Text(7, 0, 'category')])
```



Logistics Regression

In [37]:
```
#Importing Libraries
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_scc

from sklearn.model_selection import RandomizedSearchCV
```

In [38]:
```
df.columns
```

Out[38]:
```
Index(['area', 'perimeter', 'compactness', 'length', 'width',
       'asymmetry coefficient', 'groove length', 'category'],
      dtype='object')
```

In [39]:
```
from sklearn.preprocessing import StandardScaler
# Using Standard scaler technique to convert all the numerical values i
numerical_cols=['area', 'perimeter', 'compactness', 'length', 'width',
       'asymmetry coefficient', 'groove length']
scaler=StandardScaler()
scaled_cols=pd.DataFrame(scaler.fit_transform(df[numerical_cols]),colum
```

In [40]:
```
X=scaled_cols
y=df['category']
```

In [41]:
```
X
```

Out[41]:

	area	perimeter	compactness	length	width	asymmetry coefficient	groove length
0	0.142098	0.215462	0.000061	0.304218	0.141702	-0.986152	-0.383577
1	0.011188	0.008224	0.428515	-0.168625	0.197432	-1.788166	-0.922013
2	-0.192067	-0.360201	1.442383	-0.763637	0.208048	-0.667479	-1.189192
3	-0.347091	-0.475333	1.039381	-0.688978	0.319508	-0.960818	-1.229983
4	0.445257	0.330595	1.374509	0.066666	0.805159	-1.563495	-0.475356
...	...	...	...	...	...	...	...
205	-0.915515	-1.043321	0.309736	-1.112048	-0.736716	-0.046135	-1.097413
206	-1.246235	-1.288937	-0.844122	-1.105261	-1.230328	0.416540	-0.826156
207	-0.567571	-0.690247	0.733948	-0.888070	-0.070604	3.076588	-0.718060
208	-1.036090	-1.035645	-0.801701	-1.026077	-1.121521	-0.068135	-0.742535
209	-0.877620	-0.935864	-0.110235	-0.872233	-0.755292	1.291223	-0.703784

210 rows × 7 columns

In [42]:
```
1
2 y
```

Out[42]:
```
0      1.0
1      1.0
2      1.0
3      1.0
4      1.0
      ...
205    3.0
206    3.0
207    3.0
208    3.0
209    3.0
Name: category, Length: 210, dtype: float64
```

In [43]:
```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

In [44]:
```
1 log = LogisticRegression()
```

In [45]:
```
1 log.fit(X_train,y_train)
```

Out[45]:
```
▾ LogisticRegression

LogisticRegression()
```

In [46]:
```
1 #train Score
2 Train_score = log.score(X_train,y_train)
3 print('Train_score: ',Train_score)
```

```
Train_score:  0.9464285714285714
```

In [47]:
```
1 #test Score
2 Test_score = log.score(X_test, y_test)
3 print('Test_Score: ',Test_score)
```

```
Test_Score:  0.9285714285714286
```

In [48]:
```
1 pred_train = log.predict(X_train)
2 pred_test = log.predict(X_test)
```

In [49]:
```
1 accuracy_logistics = accuracy_score(y_test,pred_test)
2 accuracy_logistics
```

Out[49]:
```
0.9285714285714286
```

In [50]:
```
print(metrics.classification_report(y_test, pred_test))
```

```
              precision    recall  f1-score   support

         1.0       0.86      0.92      0.89        13
         2.0       1.00      1.00      1.00        12
         3.0       0.94      0.88      0.91        17

    accuracy                           0.93        42
   macro avg       0.93      0.94      0.93        42
weighted avg       0.93      0.93      0.93        42
```

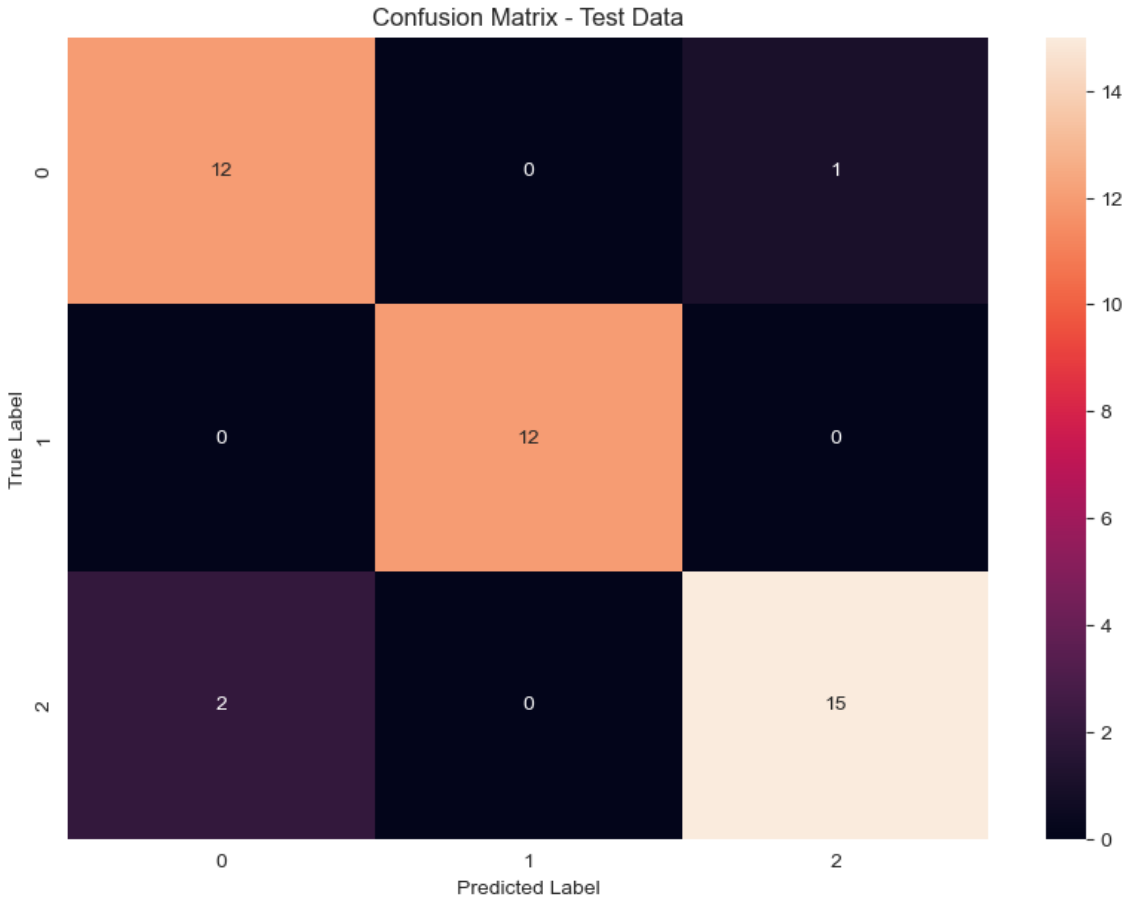
In [51]:
```
# Roc

```

In [52]:
```
from sklearn.metrics import matthews_corrcoef

mcc= matthews_corrcoef(y_test,pred_test)
print("MCC: ",mcc)
```

```
MCC:  0.8927068225145874
```

In [53]:
```
# Confusion Matrix
cm = confusion_matrix(y_test,pred_test)
print('Confusion Matrix: ',)
print(cm)
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt = '.3g')
plt.title('Confusion Matrix - Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
Confusion Matrix:
[[12  0  1]
 [ 0 12  0]
 [ 2  0 15]]
```



tuning

In [54]:
```
# Grid Search CV
```

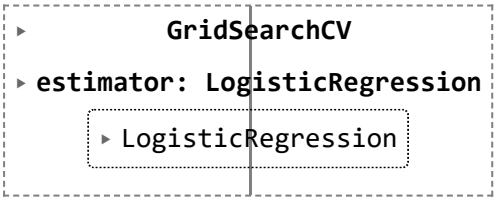
In [55]:
```
#grid search tuning parameters are deinfed, using gridsearchcv model is
param_grid = {
    'penalty':['l1', 'l2'],
    'C' : [0.1, 0.5, 1, 5, 10]
}
#l1 lasso l2 ridge to regularize the model , C common factor alpha LOS
```

In [56]:
```
grid = GridSearchCV(estimator = log, param_grid=param_grid, cv = 5)
```

In [57]:
```
grid.fit(X_train, y_train)
```

```
C:\Users\anike\anaconda3\anaconda\Lib\site-packages\sklearn\model_selectio
n\_validation.py:378: FitFailedWarning:
25 fits failed out of a total of 50.
The score on these train-test partitions for these parameters will be set
to nan.
If these failures are not expected, you can try to debug them by setting e
rror_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------
------
25 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\anike\anaconda3\anaconda\Lib\site-packages\sklearn\model_
selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\anike\anaconda3\anaconda\Lib\site-packages\sklearn\linear
_model\_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\anike\anaconda3\anaconda\Lib\site-packages\sklearn\linear
_model\_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 pe
nalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\anike\anaconda3\anaconda\Lib\site-packages\sklearn\model_selectio
n\_search.py:952: UserWarning: One or more of the test scores are non-fini
te: [       nan 0.92263815        nan 0.92263815        nan 0.9285205
        nan 0.92834225        nan 0.94010695]
  warnings.warn(
```

Out[57]:



In [58]:
```
# best paramters and best model is chosen
best_param = grid.best_params_
best_model = grid.best_estimator_
```

In [59]:
```
#Machine choses the best parameters
print('Best hyperparameters: ',best_param)
```

```
Best hyperparameters:  {'C': 10, 'penalty': 'l2'}
```

In [60]:
```
# predicted values of y are generated based on best model on the test d
gridlor_y_pred = best_model.predict(X_test)

```

In [61]:
```
Glor_acc=accuracy_score(y_test, gridlor_y_pred)
Glor_acc
```

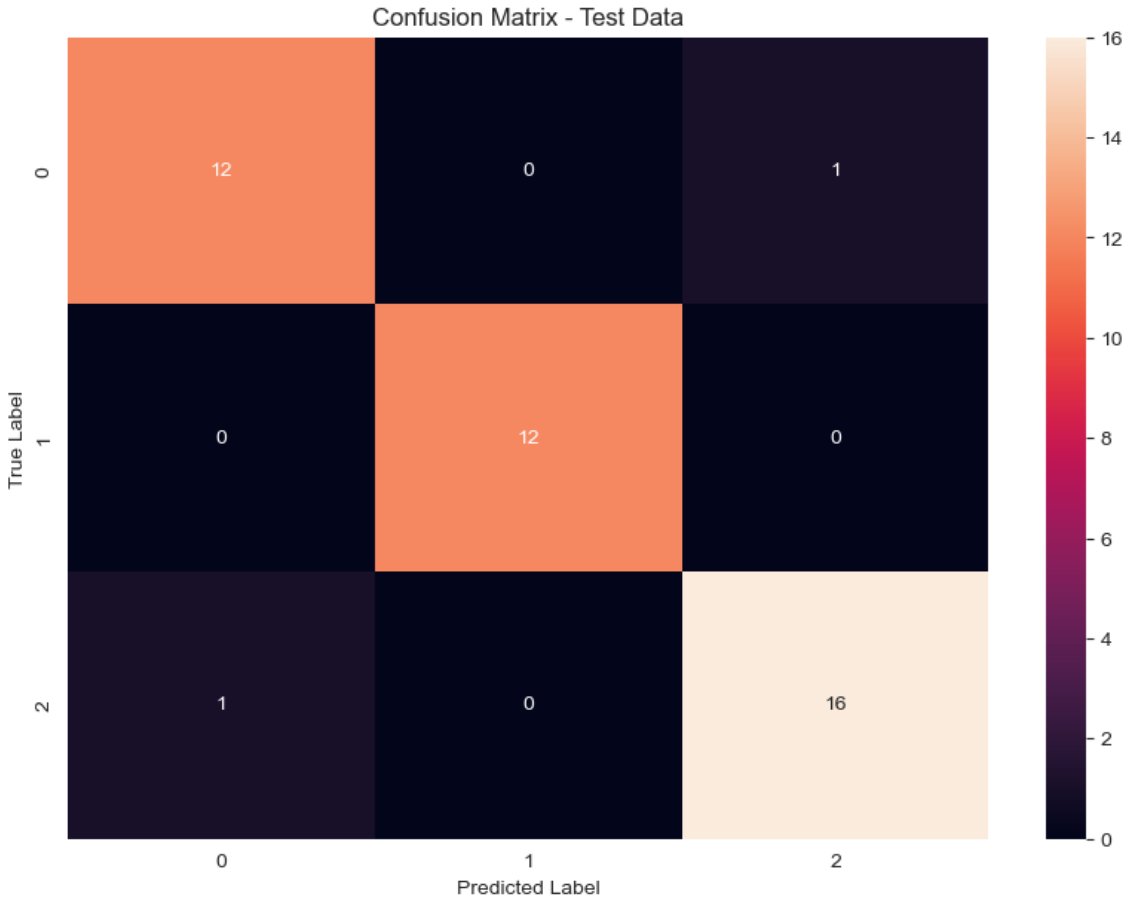
Out[61]: 0.9523809523809523

In [62]:
```
#classification report generated on the testing values
print(metrics.classification_report(y_test, gridlor_y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.92      0.92      0.92        13
         2.0       1.00      1.00      1.00        12
         3.0       0.94      0.94      0.94        17

    accuracy                           0.95        42
   macro avg       0.95      0.95      0.95        42
weighted avg       0.95      0.95      0.95        42
```

In [63]:
```
#confusion matrix
cm = confusion_matrix(y_test,gridlor_y_pred)
print('Confusion Matrix: ',)
print(cm)
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt = '.3g')
plt.title('Confusion Matrix - Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
Confusion Matrix:
[[12  0  1]
 [ 0 12  0]
 [ 1  0 16]]
```



In [64]:
```
#randomizedsearchcv
```

In [65]:
```
from scipy.stats import loguniform
from sklearn.model_selection import RandomizedSearchCV
#randomized search tuning parameters are deinfed, using randomized sear
logistic_random_param={'C':loguniform(1e-4,1e0),
                       'max_iter':(np.arange(100,800,10))}
grid=RandomizedSearchCV(estimator=log, param_distributions=logistic_rar
```

In [66]:
```
grid.fit(X_train,y_train)
```

Out[66]:

▸ RandomizedSearchCV

▸ estimator: LogisticRegression

▸ LogisticRegression

In [67]:
```
# best paramters and best model is chosen
best_param = grid.best_params_
best_model = grid.best_estimator_
```

In [68]:
```
#Machine choses the best parameters
print("Best Hyperparameters:", best_param)
```

```
Best Hyperparameters: {'C': 0.9754925706705061, 'max_iter': 220}
```

In [69]:
```
#predict y
ranlor_y_pred = best_model.predict(X_test)
```

In [70]:
```
Rlor_acc=accuracy_score(y_test, ranlor_y_pred)
Rlor_acc
```

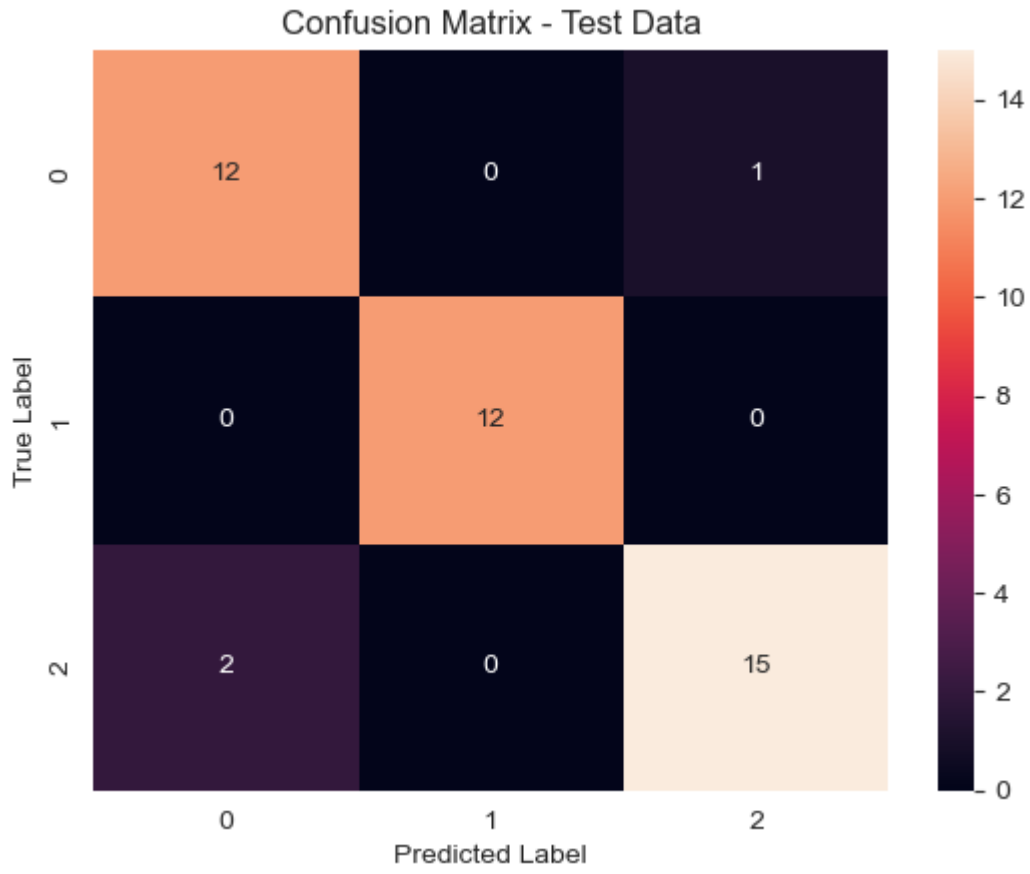
Out[70]:
```
0.9285714285714286
```

In [71]:
```
#classification report
print(metrics.classification_report(y_test, ranlor_y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.86      0.92      0.89        13
         2.0       1.00      1.00      1.00        12
         3.0       0.94      0.88      0.91        17

    accuracy                           0.93        42
   macro avg       0.93      0.94      0.93        42
weighted avg       0.93      0.93      0.93        42
```

In [72]:
```
cm = confusion_matrix(y_test,ranlor_y_pred)
print('Confusion Matrix: ')
print(cm)
sns.heatmap(cm, annot=True, fmt = '.3g')
plt.title('Confusion Matrix - Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
Confusion Matrix:
[[12  0  1]
 [ 0 12  0]
 [ 2  0 15]]
```



SVM

In [73]:
```
#importing important libraries
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,confu
from sklearn.metrics import roc_curve,auc
```

In [74]:
```
svcm= SVC()
```

In [75]:
```
#build the model
svcm.fit(X_train,y_train)
```

Out[75]:
```
SVC()
```

In [76]:
```
#predicting the values of Y based on X-test
svcm_y_pred = svcm.predict(X_test)
```

In [77]:
```
# accuracy
accuracy_SVM=accuracy_score(y_test,svcm_y_pred)
print('Accuracy:',accuracy_SVM)
```

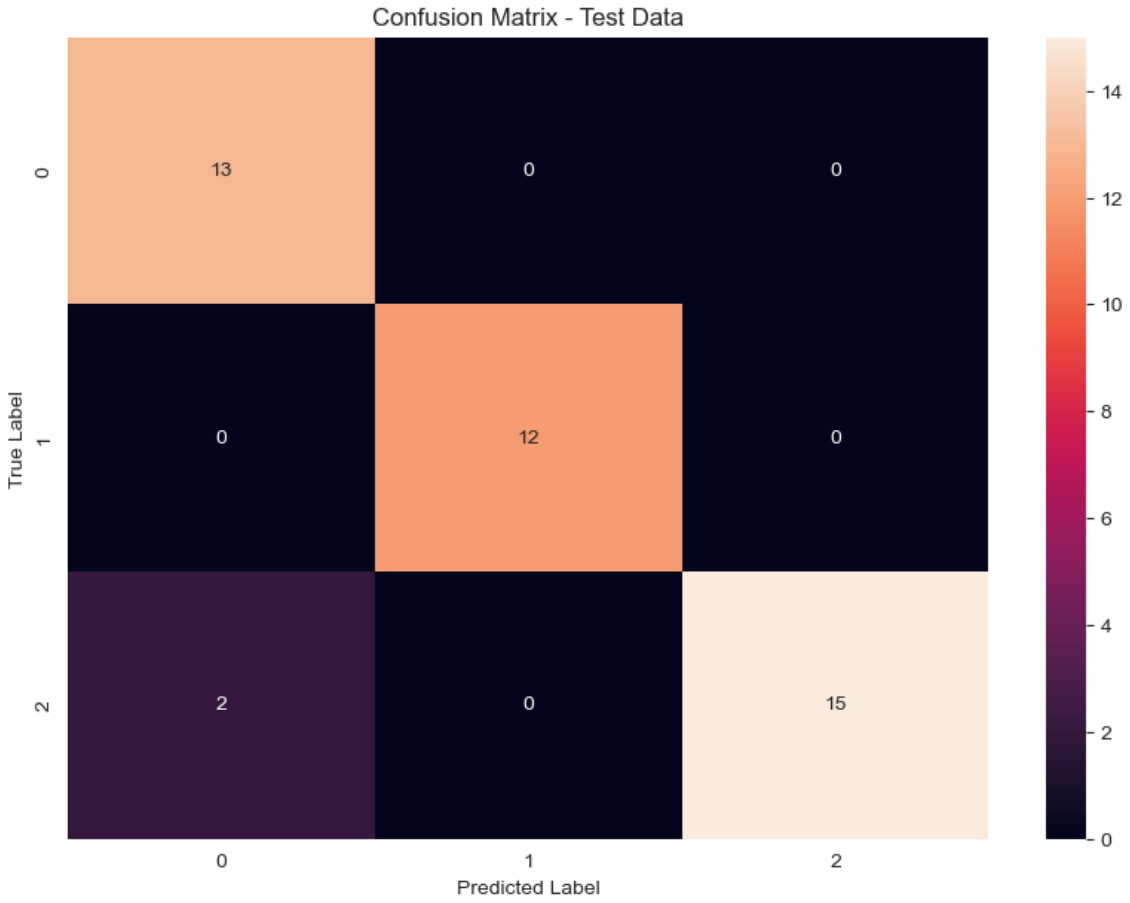
```
Accuracy: 0.9523809523809523
```

In [78]:
```
#classification report
print(classification_report(y_test,svcm_y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.87      1.00      0.93        13
         2.0       1.00      1.00      1.00        12
         3.0       1.00      0.88      0.94        17

    accuracy                           0.95        42
   macro avg       0.96      0.96      0.96        42
weighted avg       0.96      0.95      0.95        42
```

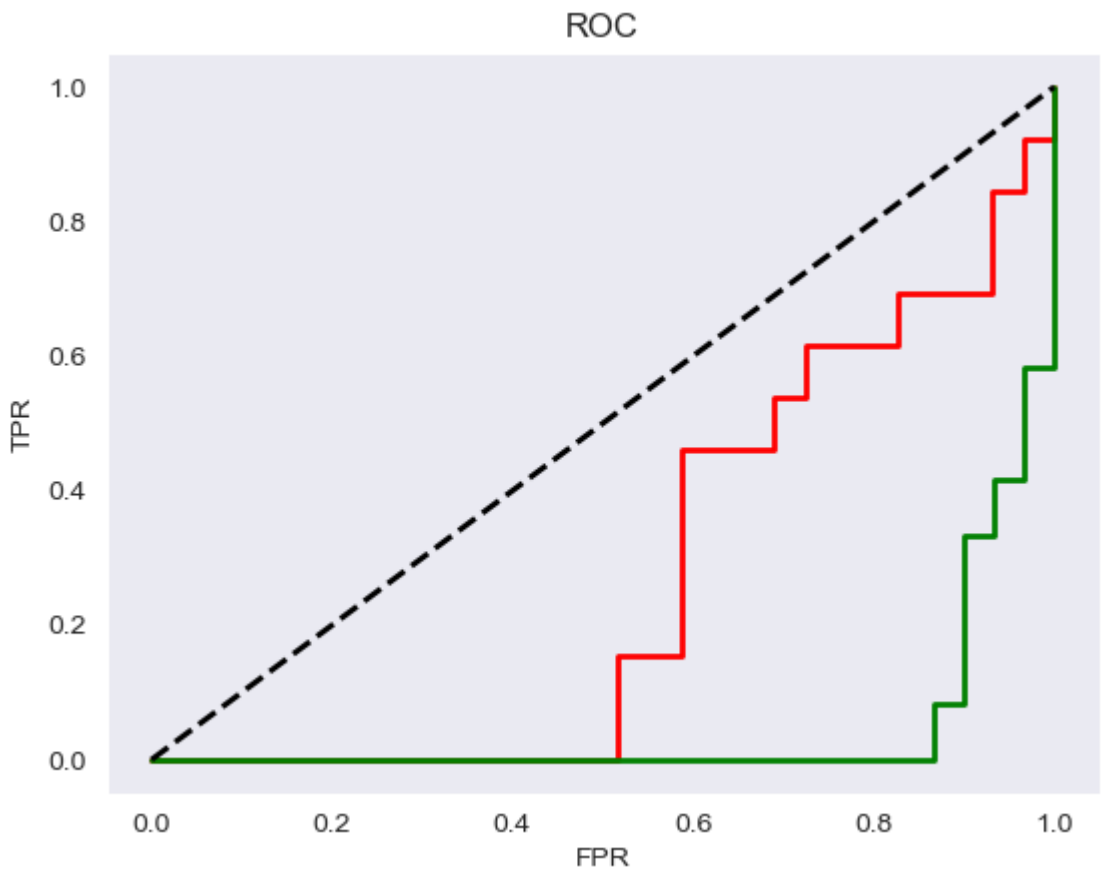
In [79]:
```
#confusion matrix
cm = confusion_matrix(y_test,svcm_y_pred)
print('Confusion Matrix: ')
print(cm)
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt='.3g')
plt.title('Confusion Matrix - Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
Confusion Matrix:
[[13  0  0]
 [ 0 12  0]
 [ 2  0 15]]
```



In [80]:
```
#roc curve
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier

yb= label_binarize(y, classes=[0,1,2])
nc= yb.shape[1]
classifier = OneVsRestClassifier(SVC(kernel='linear', probability=True,
y_score = classifier.fit(X_train, y_train).decision_function(X_test)
fpr=dict()
tpr=dict()
roc_auc = dict()
for i in range(nc):
    fpr[i], tpr[i], _ = roc_curve(y_test ==i, y_score[:,i])
    roc_auc[i] = auc(fpr[i], tpr[i])
plt.figure()
color = ['blue','red','green']
for i, color in zip(range(nc),color):
    plt.plot (fpr[i], tpr[i], color=color, lw=2,
              label='ROC(area= {: .2f}) for class{}'.format(roc_auc[i],c
plt.plot([0,1], [0,1], 'k--', lw=2)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC')
plt.show()
```

```
C:\Users\anike\anaconda3\anaconda\Lib\site-packages\sklearn\metrics\_ranki
ng.py:1029: UndefinedMetricWarning: No positive samples in y_true, true po
sitive value should be meaningless
  warnings.warn(
```



tuning

In [81]:
```
#importing important libraries
from sklearn.metrics import accuracy_score, precision_score, recall_sco

```

In [82]:
```
# grid search
```

In [83]:
```
#providing values for different parameters used in the gridsearchcv suc
param_grid = {
    'C': [0.1, 1, 10],           # Regularization parameter
    'gamma': [1,0.1,0.01,0.001],
    'kernel': ['linear', 'rbf','poly','sigmoid'],  # Kernel type

}
```

In [84]:
```

grid=GridSearchCV(svcm,param_grid,cv=5,verbose=3)
```

In [85]:
```
grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV 1/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.971 total tim
e=   0.0s
[CV 2/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.912 total tim
e=   0.0s
[CV 3/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.882 total tim
e=   0.0s
[CV 4/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.879 total tim
e=   0.0s
[CV 5/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.970 total tim
e=   0.0s
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.912 total tim
e=   0.0s
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.912 total tim
e=   0.0s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.824 total tim
e=   0.0s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.848 total tim
e=   0.0s
```

In [86]:
```
best_parameter = grid.best_params_
best_model = grid.best_estimator_
print('hyperparameters: ',best_parameter)
```

```
hyperparameters:  {'C': 1, 'gamma': 0.1, 'kernel': 'sigmoid'}
```

In [87]:
```
#predicting y
svm_grid_y_pred = best_model.predict(X_test)
```

In [88]:
```
#accuracy
Gsvm_acc=accuracy_score(y_test,svm_grid_y_pred)
print("Accuracy:",Gsvm_acc)
```

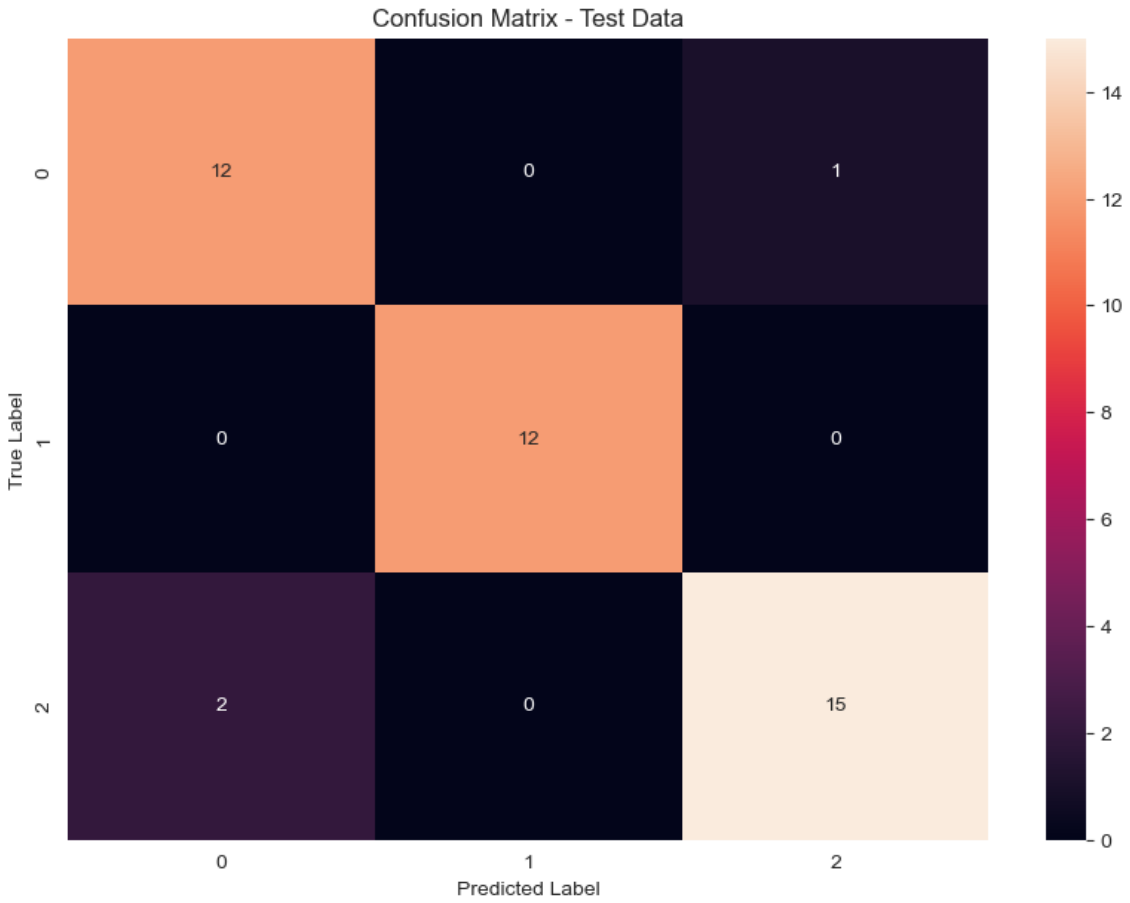
```
Accuracy: 0.9285714285714286
```

In [89]:
```
#classification report
print(classification_report(y_test,svm_grid_y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.86      0.92      0.89        13
         2.0       1.00      1.00      1.00        12
         3.0       0.94      0.88      0.91        17

    accuracy                           0.93        42
   macro avg       0.93      0.94      0.93        42
weighted avg       0.93      0.93      0.93        42
```

In [90]:
```
#confusion Matrix
cm = confusion_matrix(y_test,svm_grid_y_pred)
print('Confusion Matrix: ')
print(cm)
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt='.3g')
plt.title('Confusion Matrix - Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
Confusion Matrix:
[[12  0  1]
 [ 0 12  0]
 [ 2  0 15]]
```

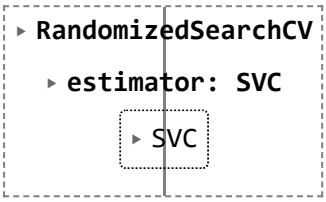


In [91]:
```
# randomizedsearchcv
```

In [92]:
```
param_grid = {
    'C': [0.1, 1, 10],            # Regularization parameter
    'kernel': ['linear', 'rbf','poly','sigmoid'],  # Kernel type

}
random=RandomizedSearchCV(svcm, param_grid, cv=5)
```

In [93]:
```
random.fit(X_train,y_train)
```

Out[93]:



In [94]:
```
best_parameters = random.best_params_
best_model = random.best_estimator_
print('Hyperparameters:',best_parameters)
```

```
Hyperparameters: {'kernel': 'linear', 'C': 10}
```

In [95]:
```
#predict of y
svmran_y_pred = best_model.predict(X_test)
```

In [96]:
```
#accuracy
Rsvm_acc=accuracy_score(y_test,svmran_y_pred)
print("Accuracy:",Rsvm_acc)
```

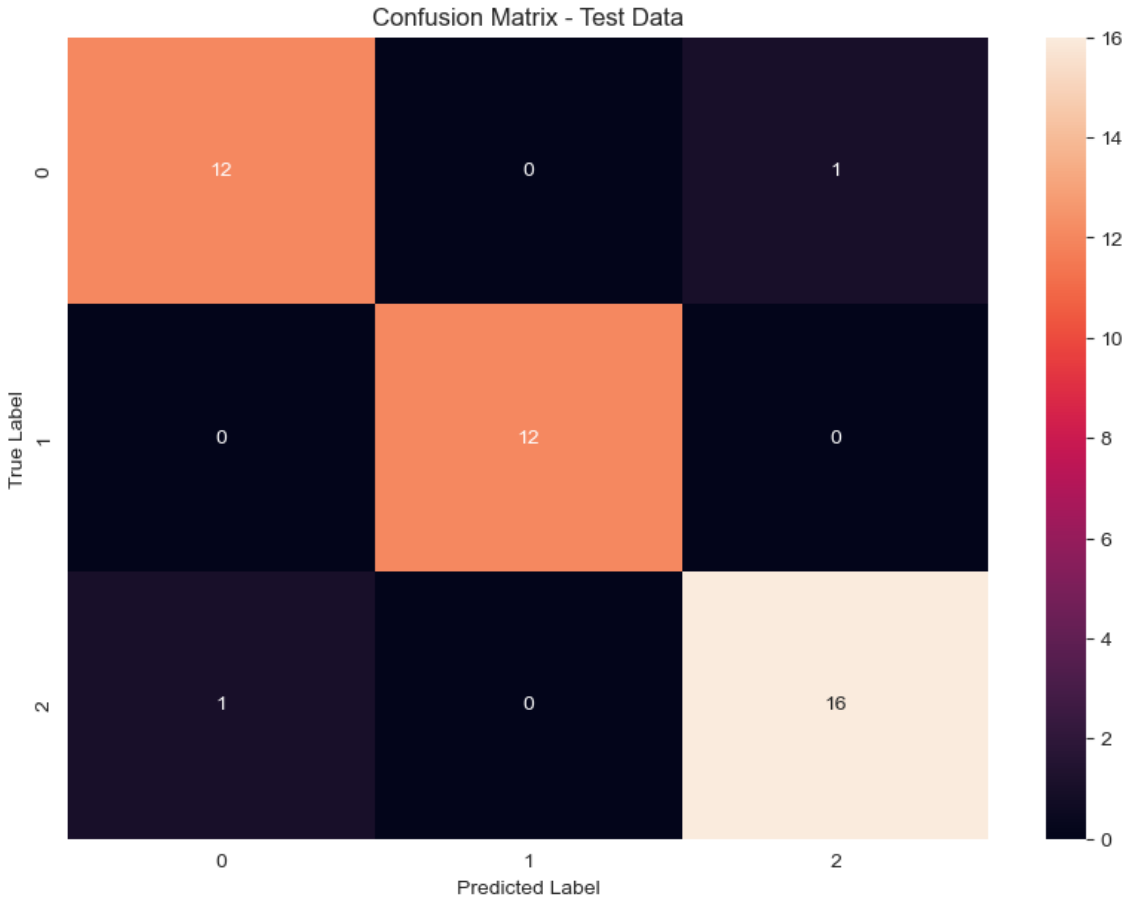
```
Accuracy: 0.9523809523809523
```

In [97]:
```
#classification report
print(classification_report(y_test,svmran_y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.92      0.92      0.92        13
         2.0       1.00      1.00      1.00        12
         3.0       0.94      0.94      0.94        17

    accuracy                           0.95        42
   macro avg       0.95      0.95      0.95        42
weighted avg       0.95      0.95      0.95        42
```

In [98]:
```
#confusion matrix
cm = confusion_matrix(y_test,svmran_y_pred)
print('Confusion Matrix: ')
print(cm)
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt='.3g')
plt.title('Confusion Matrix - Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
Confusion Matrix:
[[12  0  1]
 [ 0 12  0]
 [ 1  0 16]]
```



kNN

In [127]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [128]:
```
from sklearn.neighbors import KNeighborsClassifier

# Create the k-NN classifier with a specified number of neighbors (k)
k = 5  # You can adjust the value of k
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
```

Out[128]:
```
KNeighborsClassifier

KNeighborsClassifier()
```

In [129]:
```
knn.score(X_test,y_test)
```

Out[129]: 0.9523809523809523

In [130]:
```
knn_y_pred = knn.predict(X_test)

```

In [131]:
```
accuracy_kNN = accuracy_score(y_test, knn_y_pred)
print("Accuracy:", accuracy_kNN)
```

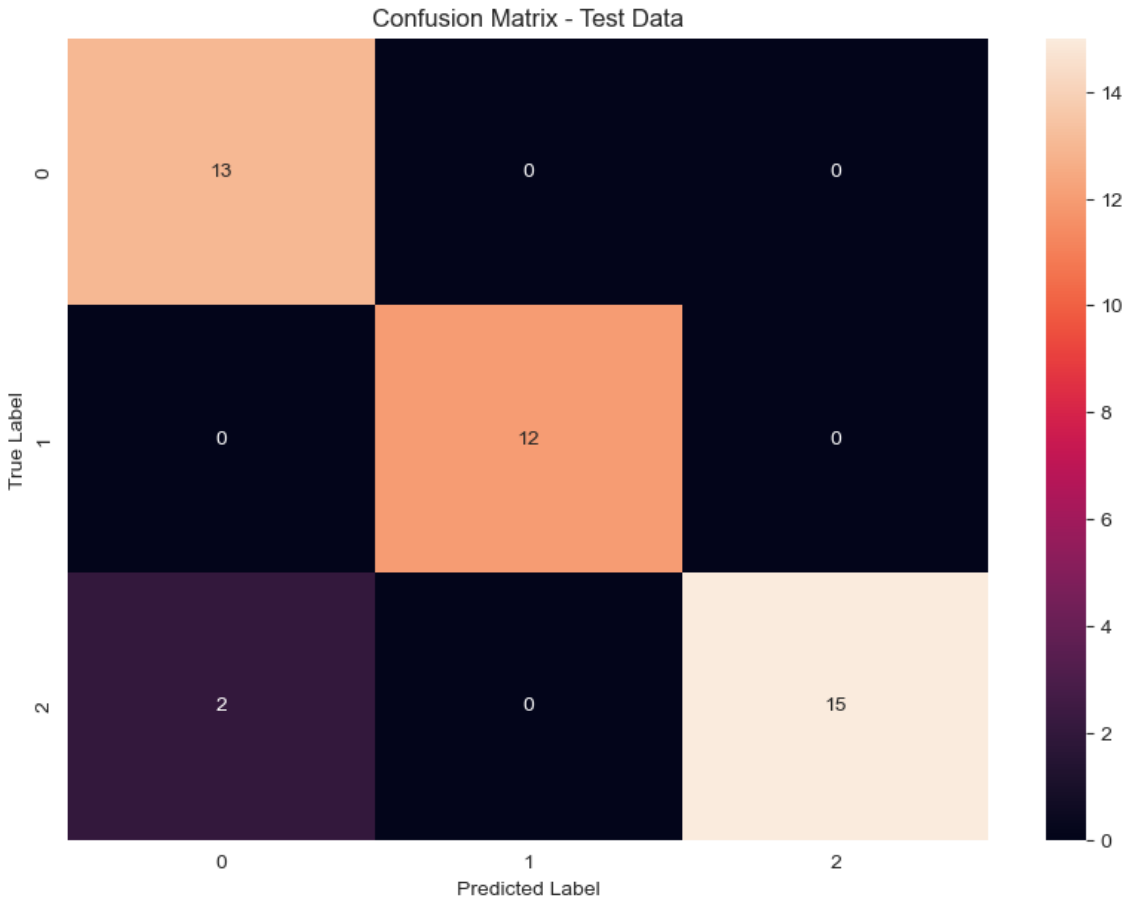
```
Accuracy: 0.9523809523809523
```

In [132]:
```
 print(classification_report(y_test, knn_y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.87      1.00      0.93        13
         2.0       1.00      1.00      1.00        12
         3.0       1.00      0.88      0.94        17

    accuracy                           0.95        42
   macro avg       0.96      0.96      0.96        42
weighted avg       0.96      0.95      0.95        42
```

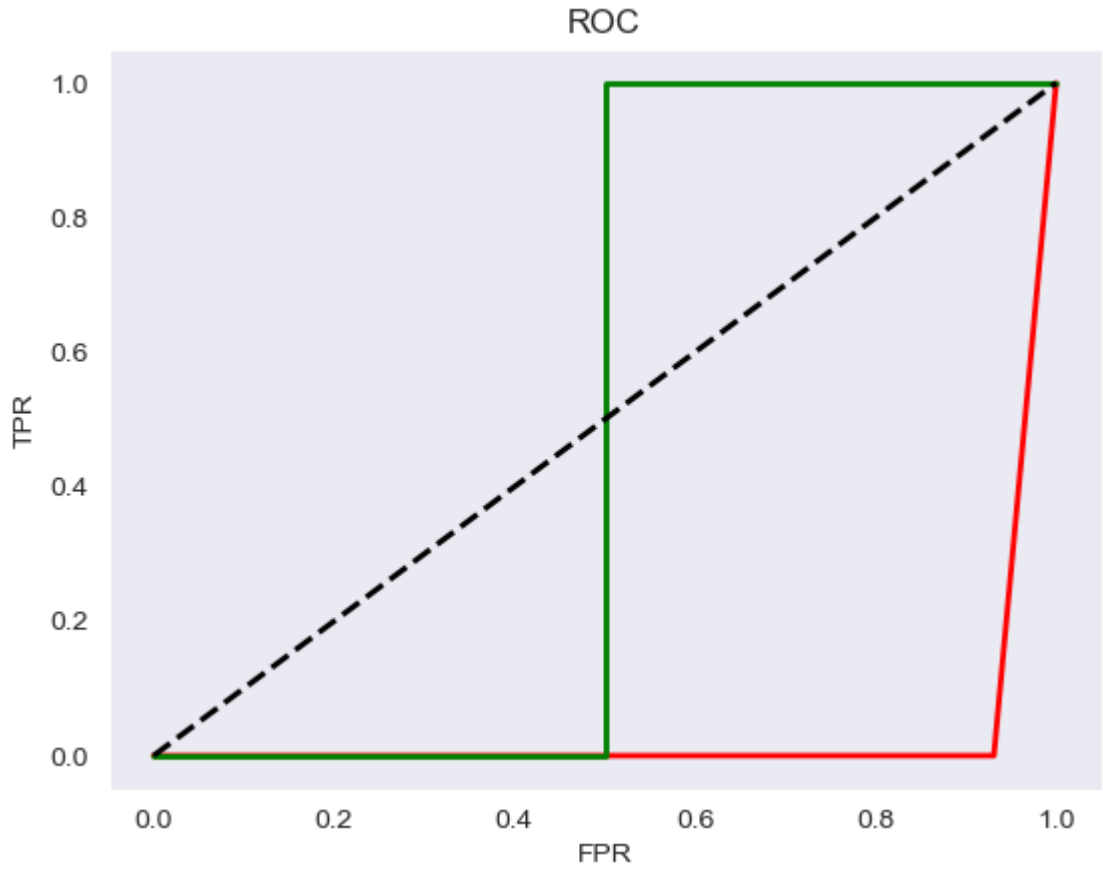
In [133]:
```
cm=confusion_matrix(y_test,knn_y_pred)
print(cm)
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt='.3g')
plt.title('Confusion Matrix - Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
[[13  0  0]
 [ 0 12  0]
 [ 2  0 15]]
```



In [134]:
```
yb=label_binarize(y, classes=[0,1,2])
nc = yb.shape[1]
classifier = OneVsRestClassifier(knn)
y_score=classifier.fit(X_train,y_train).predict(X_test)
fpr=dict()
tpr=dict()
roc_auc=dict()

for i in range (nc):
    fpr[i],tpr[i],_=roc_curve(y_test == i, y_score)
    roc_auc[i]=auc(fpr[i],tpr[i])
plt.figure()
color=['blue','red','green']
for i, color in zip(range(nc),color):
    plt.plot(fpr[i],tpr[i],color=color, lw=2, label='ROC (area={:.2f})
plt.plot([0,1],[0,1],'k--',lw=2)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC')
plt.show()
```

```
C:\Users\anike\anaconda3\anaconda\Lib\site-packages\sklearn\metrics\_ranki
ng.py:1029: UndefinedMetricWarning: No positive samples in y_true, true po
sitive value should be meaningless
  warnings.warn(
```



In [135]:
```
# Tuning kNN
```

In [136]:
```
#GridSearchCV
```

In [137]:
```
param_grid = {
    'n_neighbors': [3, 5, 7, 9],  # Test different values of k
    'weights': ['uniform', 'distance'],  # Weighting type
    'p': [1, 2],  # Distance metric (1 for Manhattan, 2 for Euclidean)
}
```

In [138]:
```
knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=5,scoring='accuracy')
grid_search.fit(X_train, y_train)
```

Out[138]:

GridSearchCV

estimator: KNeighborsClassifier

KNeighborsClassifier

In [139]:
```
best_param = grid_search.best_params_
best_knn = KNeighborsClassifier(n_neighbors = best_param['n_neighbors'])
best_knn.fit(X_train, y_train)
knn_grid_y_pred = best_knn.predict(X_test)
```

In [140]:
```
print("Best Hyperparameter : ", best_param)
```

```
Best Hyperparameter :  {'n_neighbors': 7, 'p': 1, 'weights': 'distance'}
```

In [141]:
```
Gknn_acc = accuracy_score(y_test, knn_grid_y_pred)
Gknn_acc
```

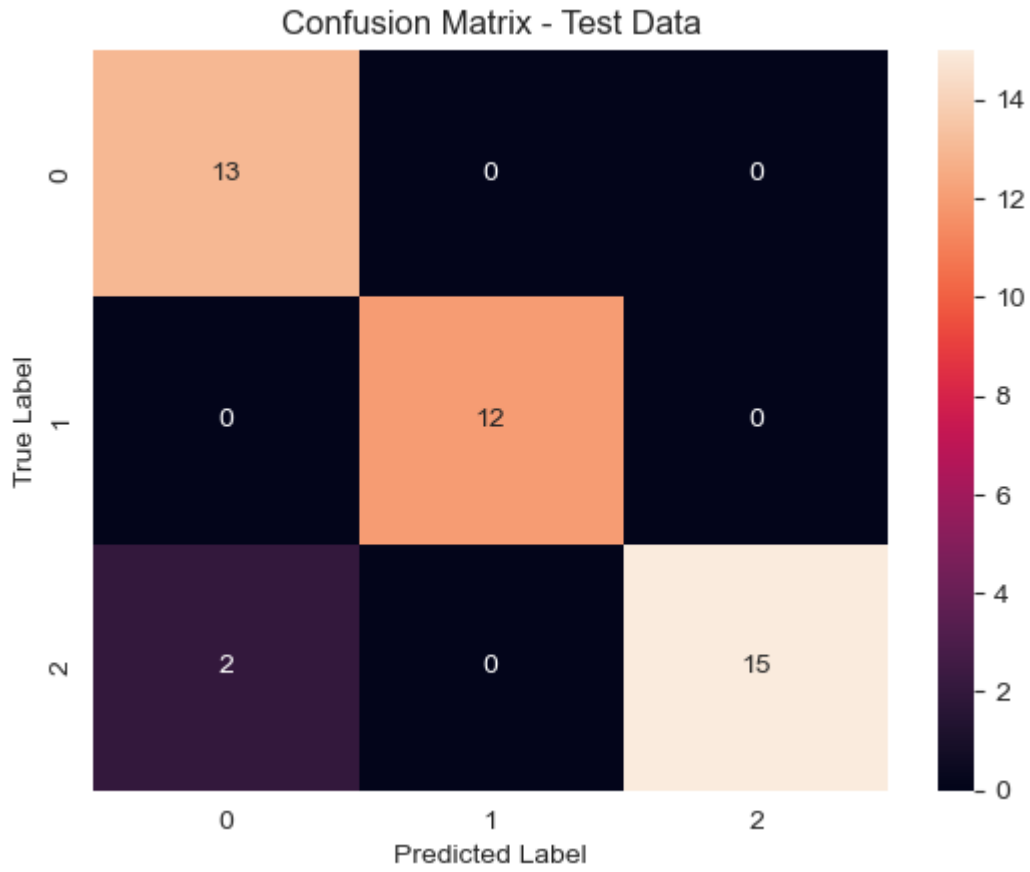
Out[141]:
```
0.9523809523809523
```

In [142]:
```
print (classification_report(y_test,knn_grid_y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.87      1.00      0.93        13
         2.0       1.00      1.00      1.00        12
         3.0       1.00      0.88      0.94        17

    accuracy                           0.95        42
   macro avg       0.96      0.96      0.96        42
weighted avg       0.96      0.95      0.95        42
```

In [143]:
```
cm=confusion_matrix(y_test,knn_grid_y_pred)
print(cm)
sns.heatmap(cm, annot=True, fmt='.3g')
plt.title('Confusion Matrix - Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
[[13  0  0]
 [ 0 12  0]
 [ 2  0 15]]
```



In [144]:
```
# Euclidean
```

In [145]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors=5, metric='euclidean'
```

In [146]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_classifier.fit(X_train, y_train)
```

Out[146]:
```
KNeighborsClassifier
KNeighborsClassifier(metric='euclidean')
```

In [147]:
```
y_pred = knn_classifier.predict(X_test)  # X_test: test features
```

In [148]:
```
accuracy = accuracy_score(y_test, y_pred)  # For classification
print('accuracy:',accuracy)
```

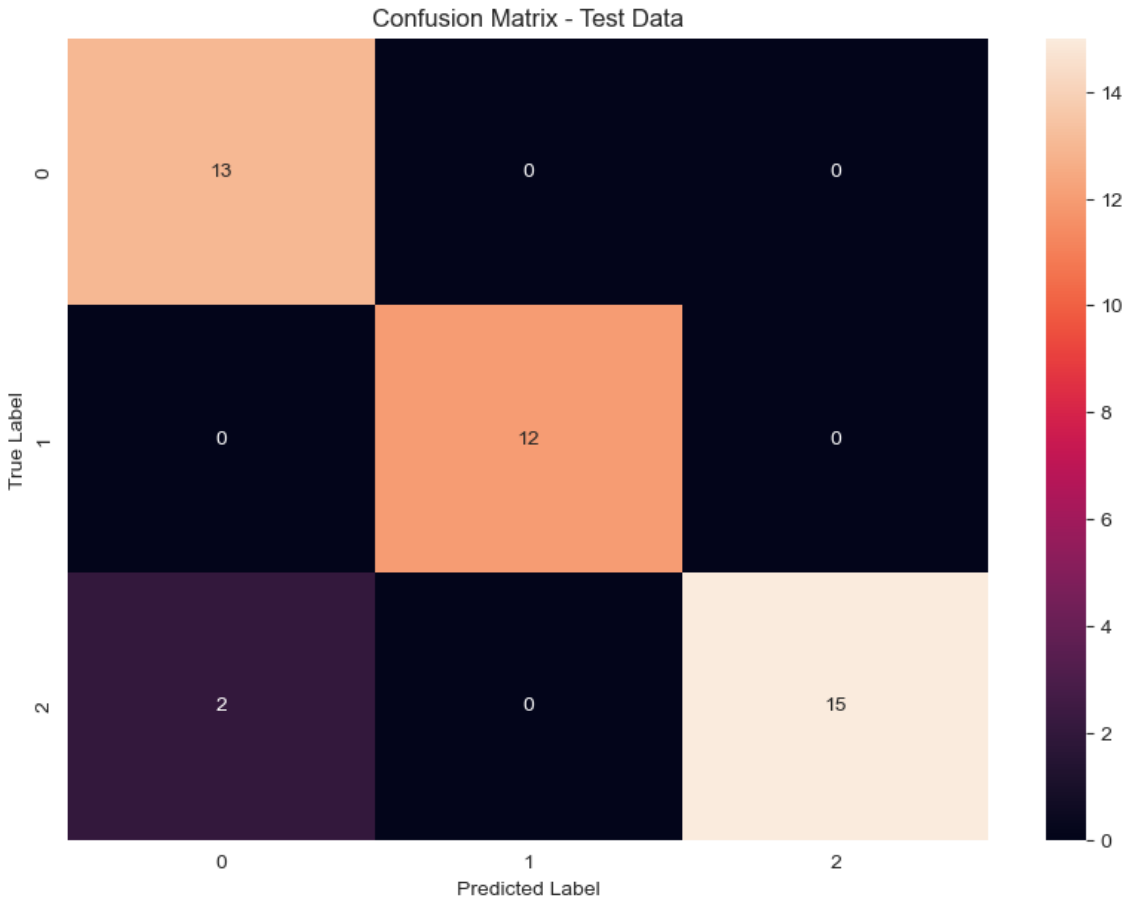
```
accuracy: 0.9523809523809523
```

In [149]:
```
print(classification_report(y_test, y_pred))  # For classification
```

```
              precision    recall  f1-score   support

         1.0       0.87      1.00      0.93        13
         2.0       1.00      1.00      1.00        12
         3.0       1.00      0.88      0.94        17

    accuracy                           0.95        42
   macro avg       0.96      0.96      0.96        42
weighted avg       0.96      0.95      0.95        42
```

In [150]:
```
cm=confusion_matrix(y_test,y_pred)
print(cm)
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt='.3g')
plt.title('Confusion Matrix - Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
[[13  0  0]
 [ 0 12  0]
 [ 2  0 15]]
```



In [151]:
```
#Manhatan
```

In [152]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=5, metric='manhattan'
```

In [153]:
```
knn_classifier.fit(X_train, y_train)  # X_train: training features, y_t
```

Out[153]:
```
KNeighborsClassifier
KNeighborsClassifier(metric='manhattan')
```

In [154]:
```
y_pred = knn_classifier.predict(X_test)  # X_test: test features
```

In [155]:
```
accuracy = accuracy_score(y_test, y_pred)  # For classification
print('accuracy:',accuracy)
```

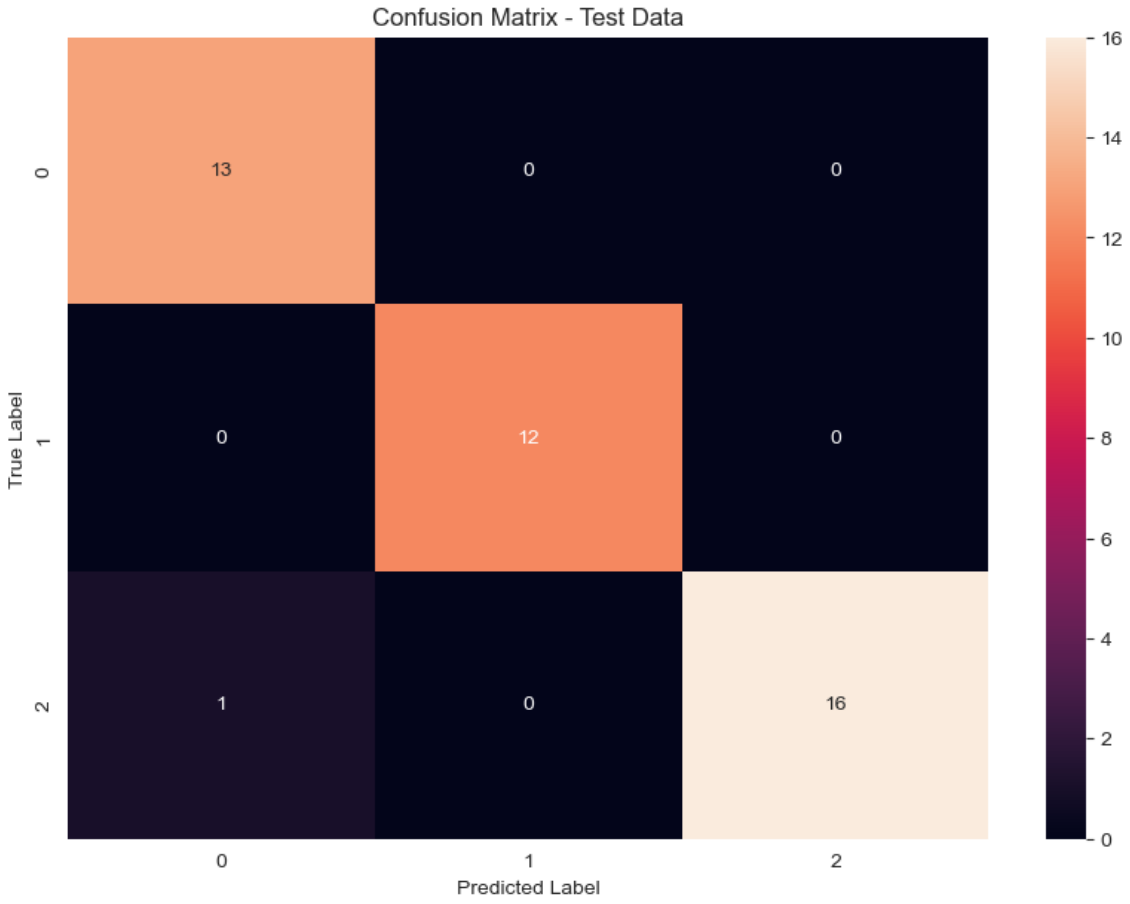
```
accuracy: 0.9761904761904762
```

In [156]:
```
print(classification_report(y_test, y_pred))  # For classification
```

```
              precision    recall  f1-score   support

         1.0       0.93      1.00      0.96        13
         2.0       1.00      1.00      1.00        12
         3.0       1.00      0.94      0.97        17

    accuracy                           0.98        42
   macro avg       0.98      0.98      0.98        42
weighted avg       0.98      0.98      0.98        42
```

In [157]:
```
#confusion Matrix
cm=confusion_matrix(y_test,y_pred)
print(cm)
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt='.3g')
plt.title('Confusion Matrix - Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
[[13  0  0]
 [ 0 12  0]
 [ 1  0 16]]
```



In [158]:
```
# Minkowsky
```

In [159]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski'

```

In [160]:
```

from sklearn.neighbors import KNeighborsClassifier
knn_classifier.fit(X_train, y_train)
```

Out[160]:
```
KNeighborsClassifier

KNeighborsClassifier()
```

In [161]:
```
#prediction of y
y_pred = knn_classifier.predict(X_test)  # X_test: test fea
```

In [162]:
```
#accuracy
accuracy = accuracy_score(y_test, y_pred)  # For classification
print('accuracy:',accuracy)
```

```
accuracy: 0.9523809523809523
```

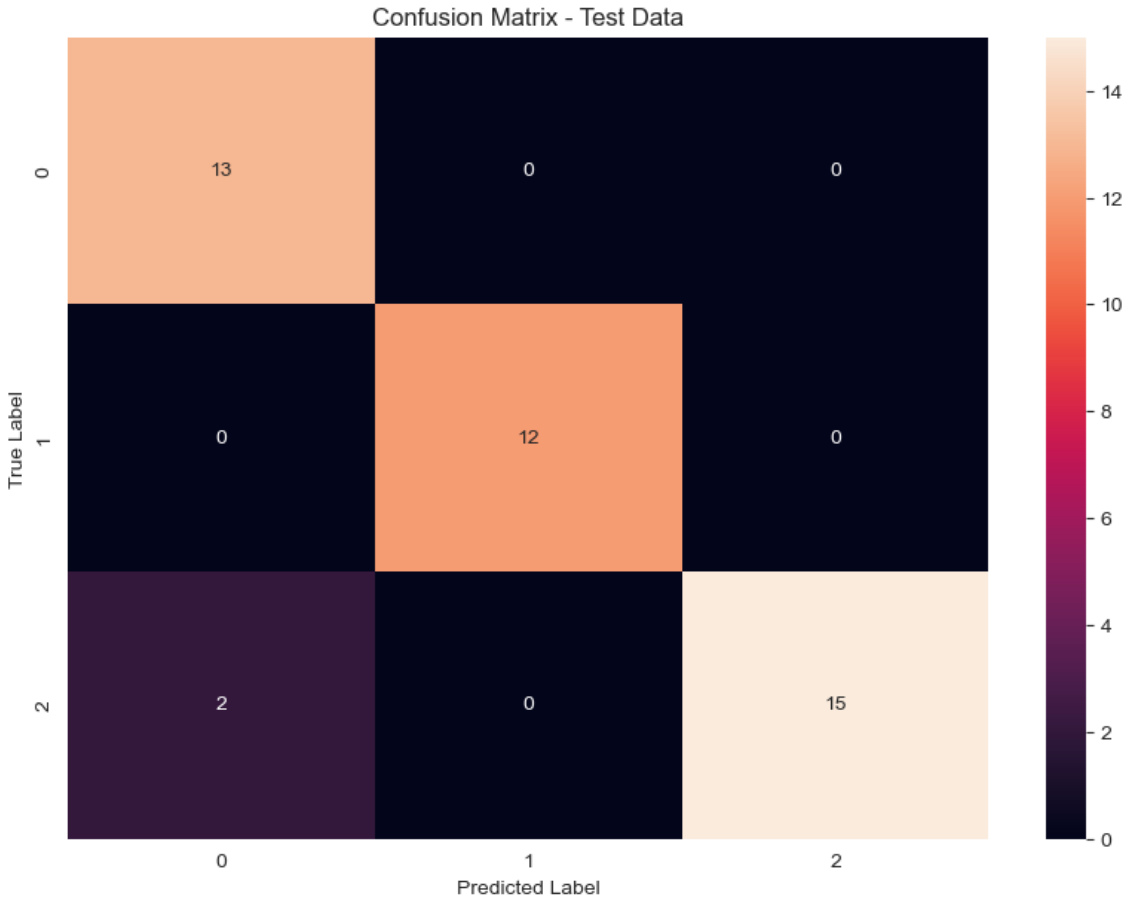
In [163]:
```
#classificarion report
print(classification_report(y_test, y_pred))  # For classification
```

```
              precision    recall  f1-score   support

         1.0       0.87      1.00      0.93        13
         2.0       1.00      1.00      1.00        12
         3.0       1.00      0.88      0.94        17

    accuracy                           0.95        42
   macro avg       0.96      0.96      0.96        42
weighted avg       0.96      0.95      0.95        42
```

In [164]:

```
#confusion Matrix
cm=confusion_matrix(y_test,y_pred)
print(cm)
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt='.3g')
plt.title('Confusion Matrix - Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
[[13  0  0]
 [ 0 12  0]
 [ 2  0 15]]
```



In [166]:
```
# Create a dictionary to store accuracy values for each model
data= {
    "Model": ["Logistic Regression", "SVM", "k-NN"],
    "Accuracy": [ accuracy_logistics,accuracy_SVM, accuracy_kNN],
    "GridSearchCV":[Glor_acc,Gsvm_acc,Gknn_acc],


}


accuracy_data = pd.DataFrame(data)
accuracy_data
```

Out[166]:

	Model	Accuracy	GridSearchCV
0	Logistic Regression	0.928571	0.952381
1	SVM	0.952381	0.928571
2	k-NN	0.952381	0.952381

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```