In [109]:
```
#imports
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
from sklearn.neighbors import NearestCentroid
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold
```

In [110]:
```
class data_loader():
  def __init__(self, data_path):
    self.data_path=data_path

  def load_data(self):
    df= pd.read_csv(self.data_path)
    return df
```

In [124]:
```
train_data_path="/content/drive/MyDrive/ML-1_Project/dry_bean_classifi
cation_train.csv"
data_loader_child=data_loader(train_data_path)
df_train=data_loader_child.load_data()

df_train.describe()
```

Out[124]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentri
count	10889.000000	10889.000000	10889.000000	10889.000000	10889.000000	10889.000
mean	53037.428965	855.112602	319.933861	202.383737	1.581485	0.750
std	29305.978236	213.927855	85.515726	44.974352	0.246503	0.092
min	20420.000000	524.736000	183.601165	122.512653	1.024868	0.218
25%	36325.000000	703.597000	253.331461	175.931143	1.429944	0.714
50%	44665.000000	795.057000	296.875317	192.626917	1.549898	0.764
75%	61377.000000	976.350000	375.980183	217.434281	1.704556	0.809
max	254616.000000	1985.370000	738.860154	450.926187	2.430306	0.911

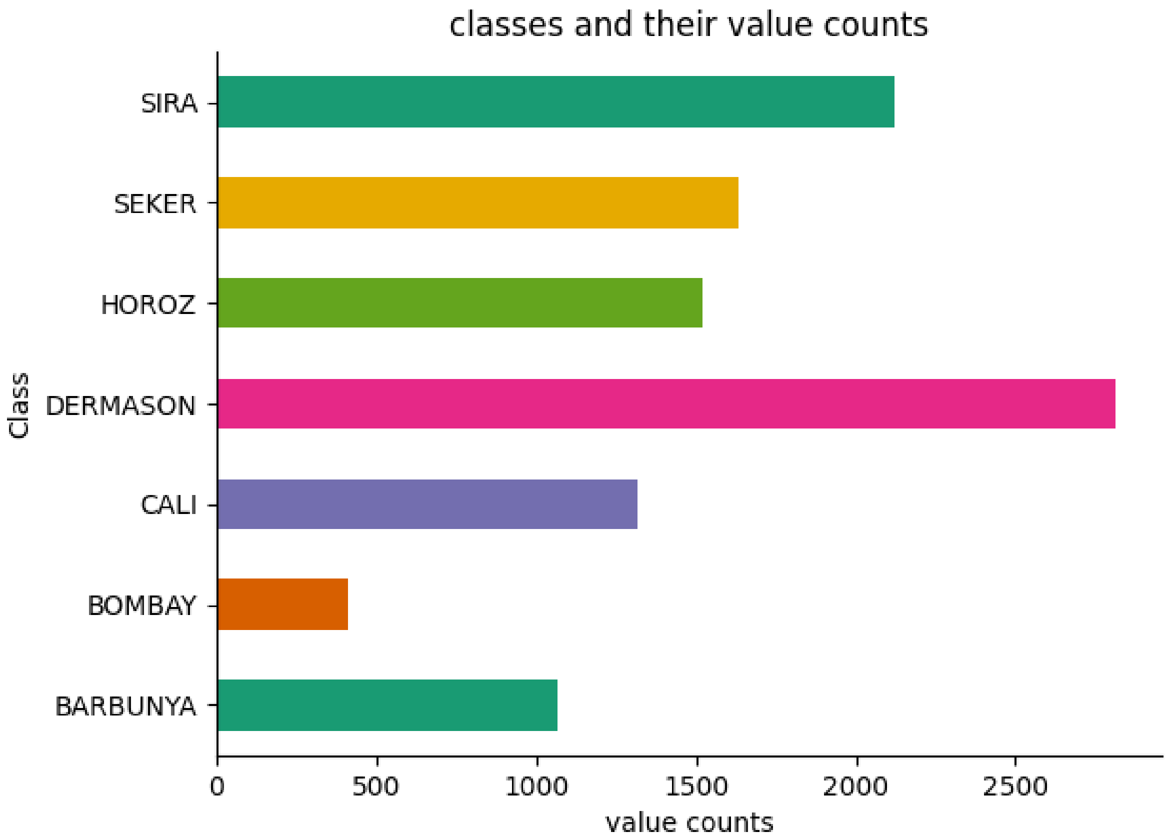
In [122]:
```
class data_visualization():
  def __init__(self, df):
    self.df=df

  def class_valuecounts(self):

    plt.title("classes and their value counts")
    self.df.groupby('Class').size().plot(xlabel="value counts",kind='b
arh', color=sns.palettes.mpl_palette('Dark2'))
    plt.gca().spines[['top', 'right',]].set_visible(False)
    print("===========================================================
===============================================")

  def feature_comparision(self):
    # MinorAxisLength vs AspectRation
    self.df.plot(kind='scatter', x='MinorAxisLength', y='AspectRatio
n', s=32, alpha=.8)
    plt.gca().spines[['top', 'right',]].set_visible(False)
    plt.title("MinorAxisLength vs AspectRation")

    print("===========================================================
===============================================")

    #MajorAxisLength vs MinorAxisLength
    self.df.plot(kind='scatter', x='MajorAxisLength', y='MinorAxisLeng
th', s=32, alpha=.8)
    plt.gca().spines[['top', 'right',]].set_visible(False)
    plt.title("MajorAxisLength vs MinorAxisLength")
    print("===========================================================
===============================================")

    # Perimeter vs MajorAxisLength
    self.df.plot(kind='scatter', x='Perimeter', y='MajorAxisLength', s
=32, alpha=.8)
    plt.gca().spines[['top', 'right',]].set_visible(False)
    plt.title("Perimeter vs MajorAxisLength")
    print("===========================================================
===============================================")

    # Class vs Area
    figsize = (5, len(self.df['Class'].unique()))
    plt.figure(figsize=figsize)
    sns.violinplot(self.df, x='Area', y='Class', inner='box', palette
='Dark2')
    sns.despine(top=True, right=True, bottom=True, left=True)
    plt.title("Class vs Area")
    print("===========================================================
===============================================")

    #Class vs Perimeter
    figsize = (5,5)
    plt.figure(figsize=figsize)
    sns.violinplot(self.df, x='Perimeter', y='Class', inner='box', pal
ette='Dark2')
    sns.despine(top=True, right=True, bottom=True, left=True)
    plt.title("Class vs Perimeter  ")
    print("===========================================================
```

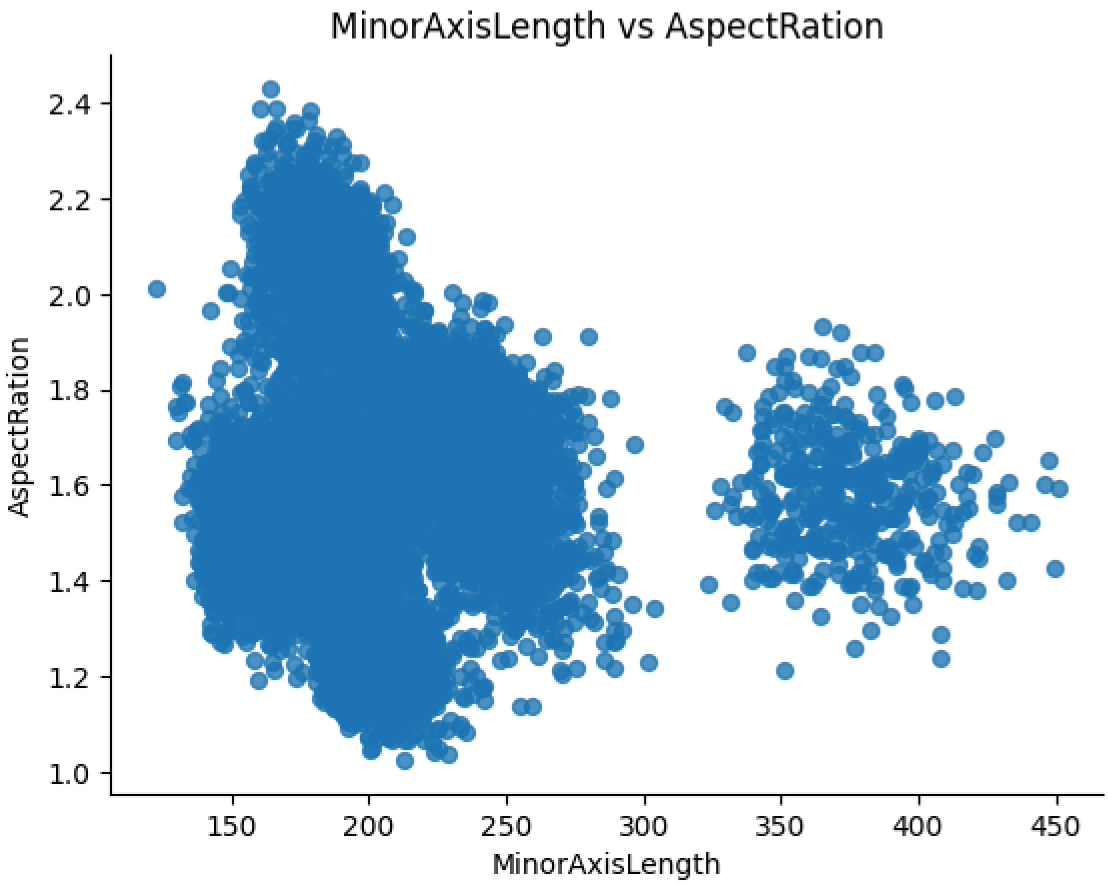
```
================================================")

    # Class vs MajorAxisLength
    figsize = (5,5)
    plt.figure(figsize=figsize)
    sns.violinplot(self.df, x='MajorAxisLength', y='Class', inner='bo
x', palette='Dark2')
    sns.despine(top=True, right=True, bottom=True, left=True)
    plt.title(" Class vs MajorAxisLength")
    print("============================================================
================================================")

    # Class vs MinorAxisLength
    #figsize = (12, 1.2 * len(df['Class'].unique()))
    figsize=(5,5)
    plt.figure(figsize=figsize)
    sns.violinplot(self.df, x='MinorAxisLength', y='Class', inner='bo
x', palette='Dark2')
    sns.despine(top=True, right=True, bottom=True, left=True)
    plt.title("Class vs MinorAxisLength")
    print("============================================================
================================================")

    #parameter influence on dataset
    self.df.hist(figsize=(10,10))
    print("============================================================
================================================")

  def corr_matrix(self):
    #correlation matrix
    df_numbersonly= self.df.select_dtypes([float,int])
    correlation_matrix= df_numbersonly.corr()

    plt.figure(figsize=(10,10))
    sns.heatmap(correlation_matrix, annot=True, fmt='.2f')
    plt.title("CORRELATION MATRIX")
    print("============================================================
================================================")
```

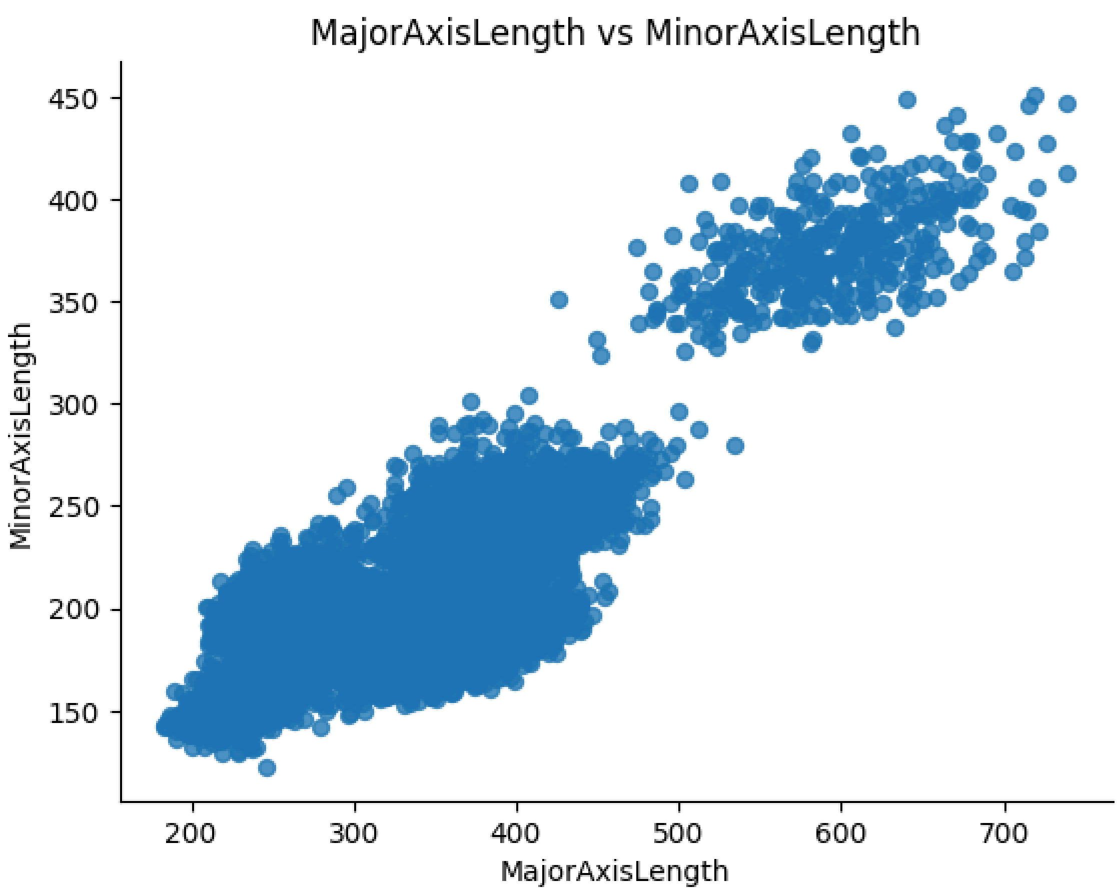
In [123]:
```
data_visualization_train_child= data_visualization(df_train)
data_visualization_train_child.class_valuecounts()
data_visualization_train_child.feature_comparision()
data_visualization_train_child.corr_matrix()
```

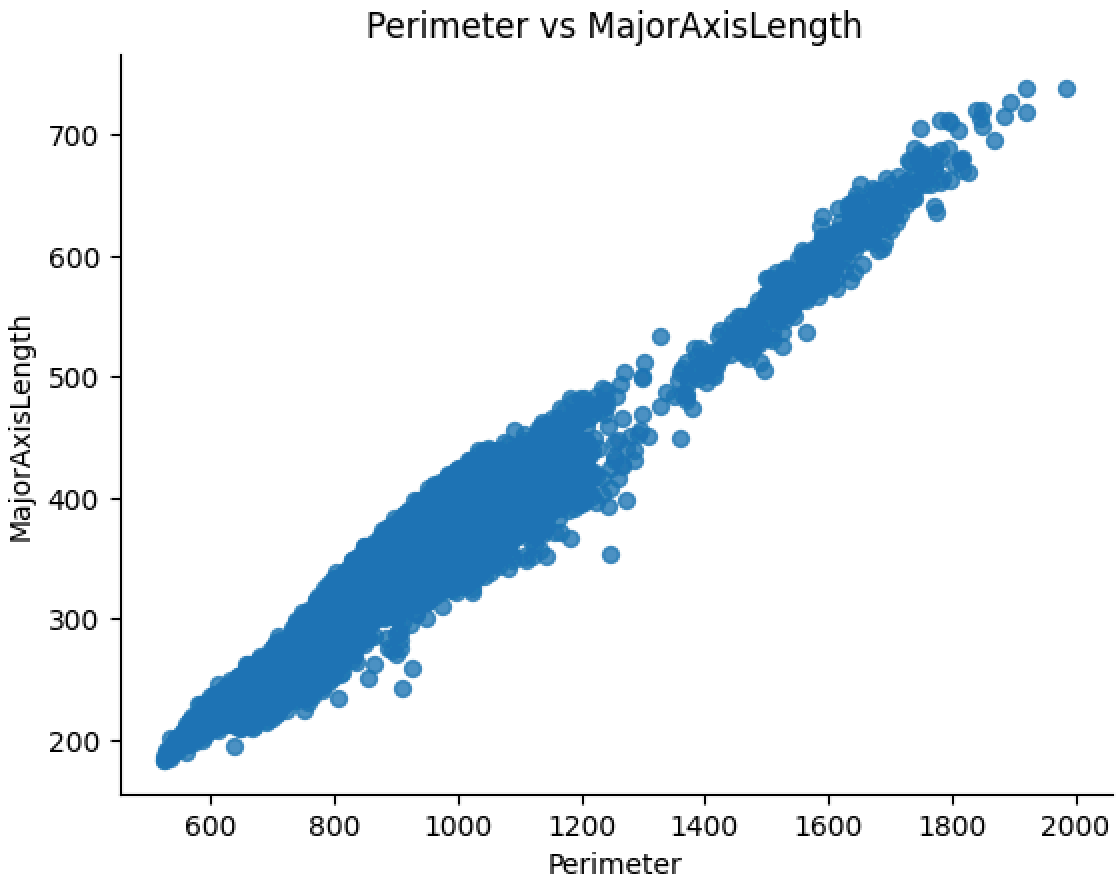
```
======================================================================
====================================
======================================================================
====================================
======================================================================
====================================
======================================================================
====================================

<ipython-input-122-d17b6975091c>:35: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be re
moved in v0.14.0. Assign the `y` variable to `hue` and set `legend=Fal
se` for the same effect.

  sns.violinplot(self.df, x='Area', y='Class', inner='box', palette='D
ark2')

======================================================================
====================================

<ipython-input-122-d17b6975091c>:43: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be re
moved in v0.14.0. Assign the `y` variable to `hue` and set `legend=Fal
se` for the same effect.

  sns.violinplot(self.df, x='Perimeter', y='Class', inner='box', palet
te='Dark2')

======================================================================
====================================

<ipython-input-122-d17b6975091c>:51: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be re
moved in v0.14.0. Assign the `y` variable to `hue` and set `legend=Fal
se` for the same effect.

  sns.violinplot(self.df, x='MajorAxisLength', y='Class', inner='box',
palette='Dark2')

======================================================================
====================================

<ipython-input-122-d17b6975091c>:60: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be re
moved in v0.14.0. Assign the `y` variable to `hue` and set `legend=Fal
se` for the same effect.

  sns.violinplot(self.df, x='MinorAxisLength', y='Class', inner='box',
palette='Dark2')
```

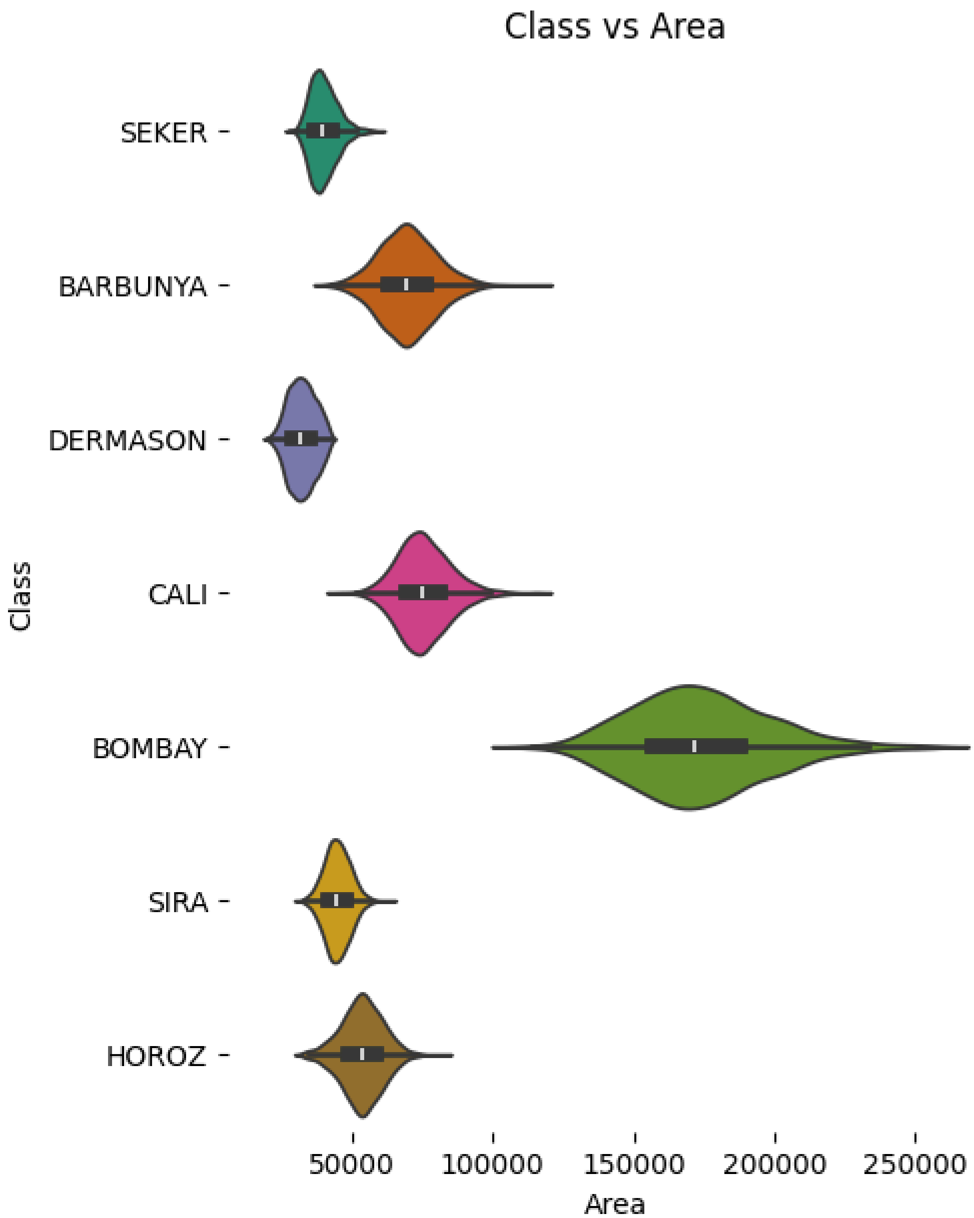
```
========================================================================
=====================================
========================================================================
=====================================
========================================================================
=====================================
```

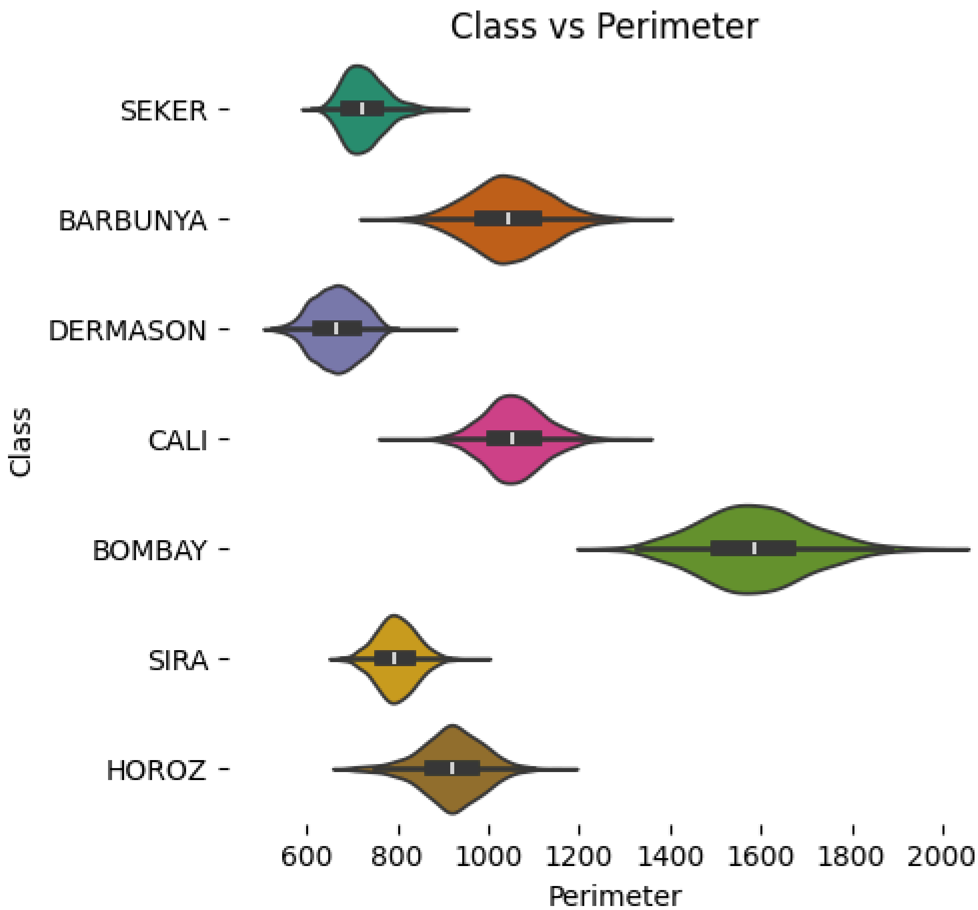

classes and their value counts
SIRA
SEKER
HOROZ
DERMASON
CALI
BOMBAY
BARBUNYA
Class
0
500
1000
1500
2000
2500
value counts

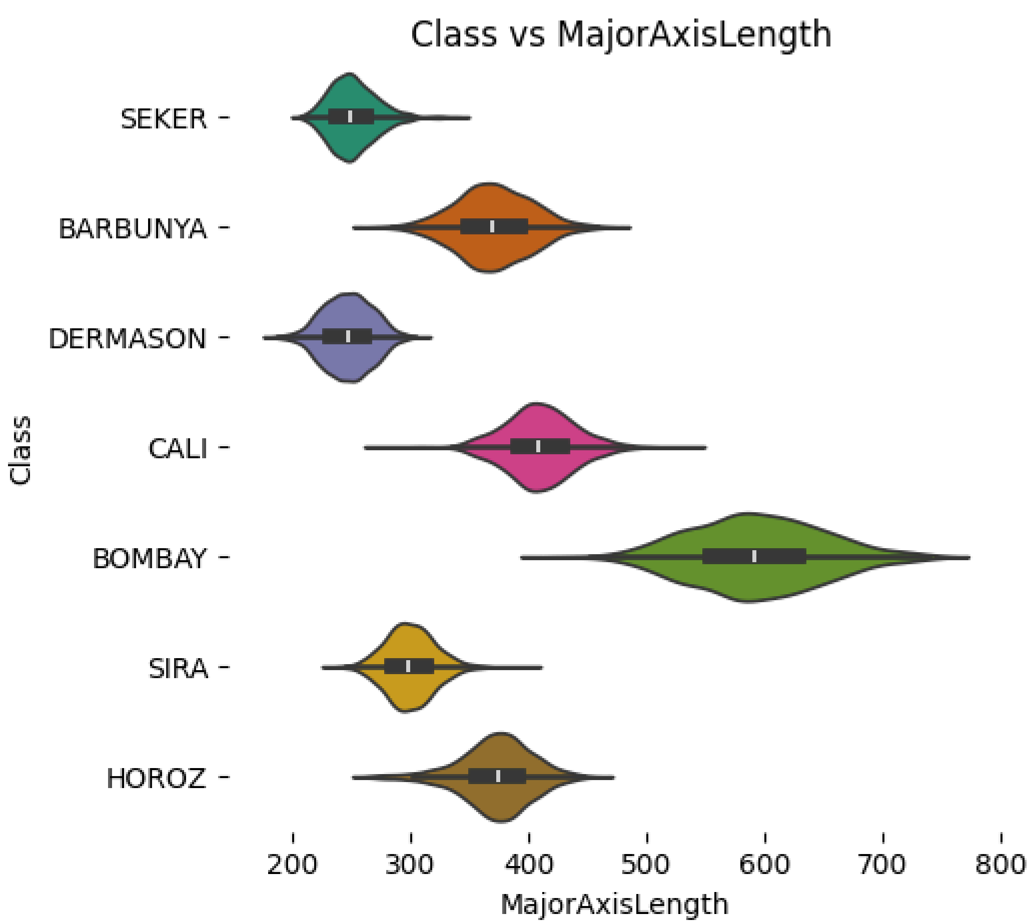

MinorAxisLength vs AspectRation
AspectRation
MinorAxisLength
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
150
200
250
300
350
400
450

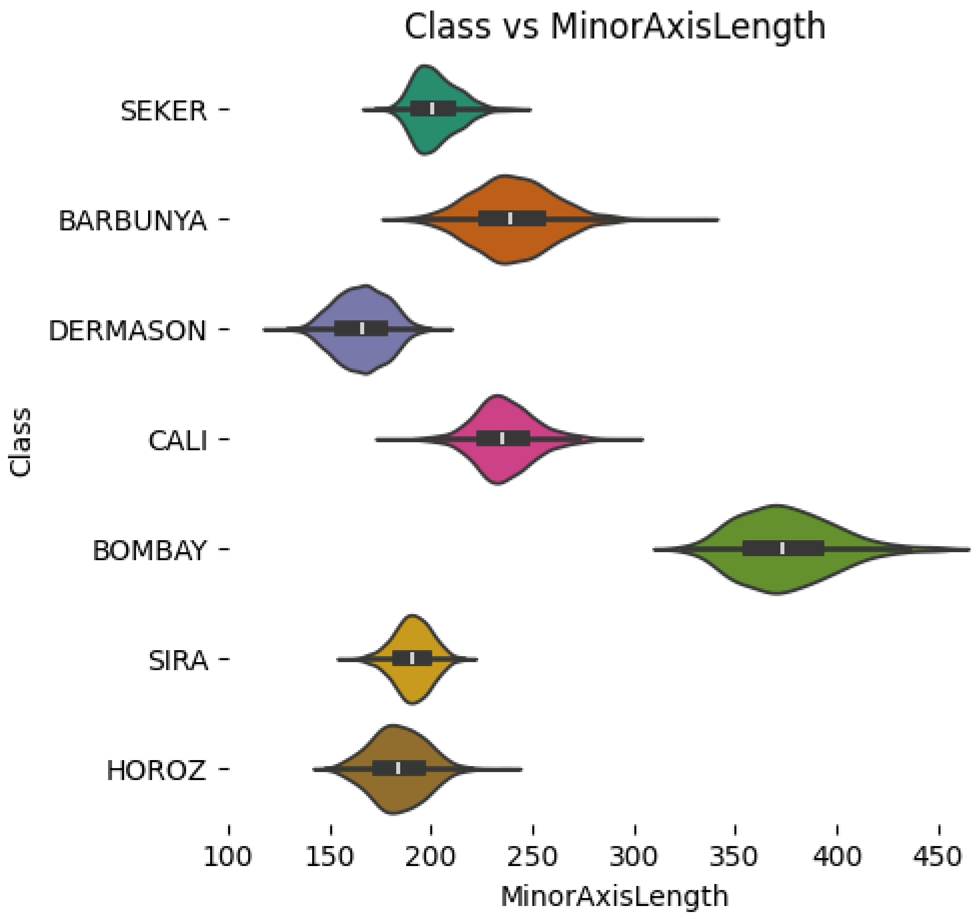

MajorAxisLength vs MinorAxisLength
MinorAxisLength
MajorAxisLength
450
400
350
300
250
200
150
200
300
400
500
600
700

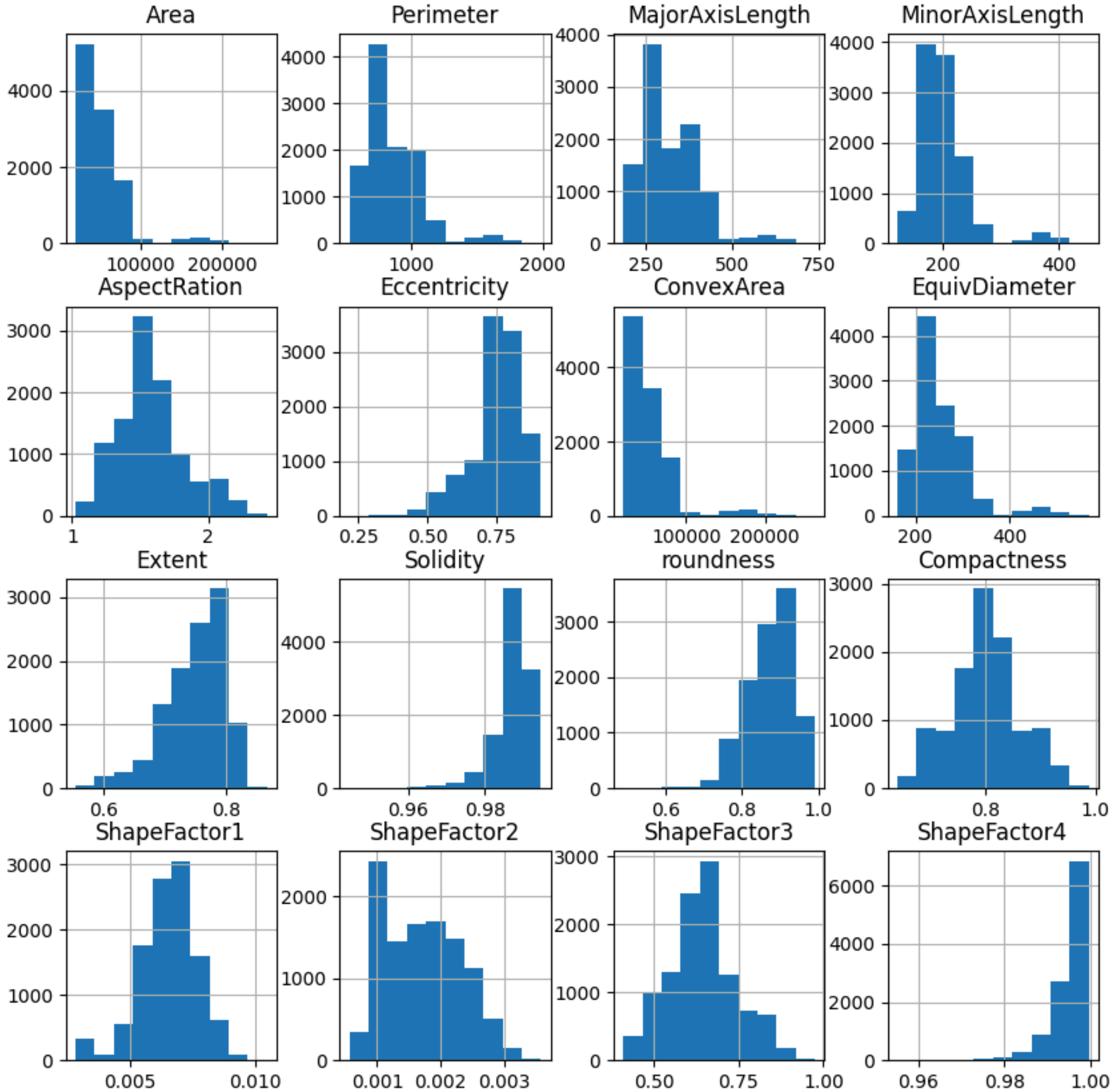

Perimeter vs MajorAxisLength
MajorAxisLength
Perimeter
200
300
400
500
600
700
600
800
1000
1200
1400
1600
1800
2000

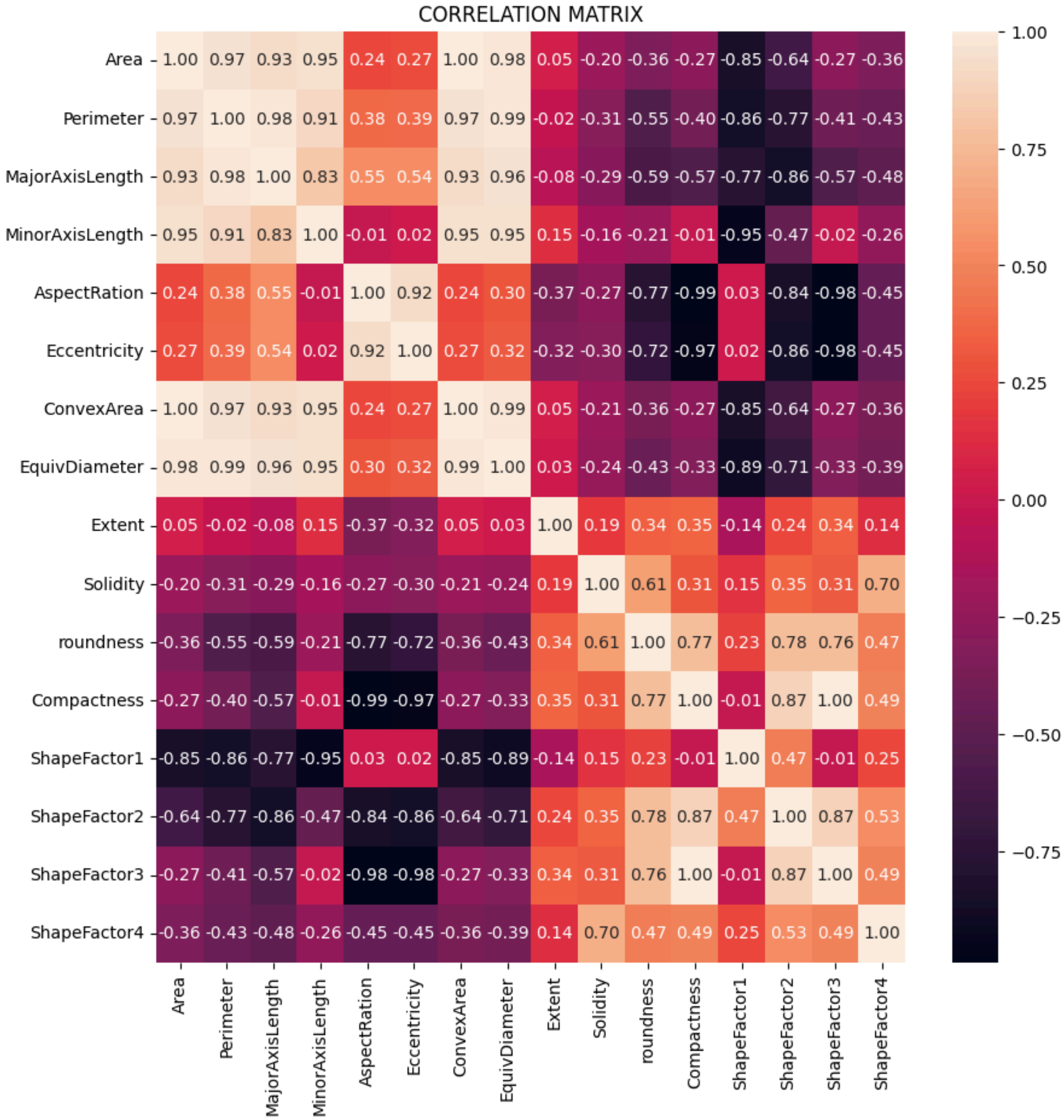

Class vs Area
SEKER
BARBUNYA
DERMASON
CALI
BOMBAY
SIRA
HOROZ
Class
50000
100000
150000
200000
250000
Area


Class vs Perimeter
SEKER
BARBUNYA
DERMASON
CALI
BOMBAY
SIRA
HOROZ
Class
600
800
1000
1200
1400
1600
1800
2000
Perimeter


Class vs MajorAxisLength
SEKER
BARBUNYA
DERMASON
CALI
BOMBAY
SIRA
HOROZ
Class
200
300
400
500
600
700
800
MajorAxisLength


Class vs MinorAxisLength
SEKER
BARBUNYA
DERMASON
CALI
BOMBAY
SIRA
HOROZ
Class
100
150
200
250
300
350
400
450
MinorAxisLength


Area
Perimeter
MajorAxisLength
MinorAxisLength
AspectRation
Eccentricity
ConvexArea
EquivDiameter
Extent
Solidity
roundness
Compactness
ShapeFactor1
ShapeFactor2
ShapeFactor3
ShapeFactor4


CORRELATION MATRIX
Area
Perimeter
MajorAxisLength
MinorAxisLength
AspectRation
Eccentricity
ConvexArea
EquivDiameter
Extent
Solidity
roundness
Compactness
ShapeFactor1
ShapeFactor2
ShapeFactor3
ShapeFactor4
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

In [95]:
```
class data_preprocessor():
  def __init__(self,df):
    self.df=df

  def check_missing_values(self):
    missing_values= self.df.isnull().sum()
    print(missing_values)

  def handle_missing_values(self, strategy='mean'):
    if strategy == 'mean':
        self.df.fillna(self.df.mean(), inplace=True)
    elif strategy == 'median':
        self.df.fillna(self.df.median(), inplace=True)
    elif strategy == 'mode': #puts most freq value in place of the mis
sing values
        mode_values = self.df.mode().iloc[0]
        self.df.fillna(mode_values, inplace=True)
    elif strategy == 'constant':
        constant_value = 0
        self.df.fillna(constant_value, inplace=True)
    else:
        raise ValueError(f"Unsupported strategy: {strategy}")
    print(f"Missing values have been handled using the '{strategy}' st
rategy.")

  def duplicate_removal(self):
    duplicate_rows= self.df.duplicated() #check for duplicates
    duplicate_row_values= self.df[duplicate_rows]
    print("duplicate row values: ")
    print(duplicate_row_values)
    df_cleaned= self.df.drop_duplicates() #removes dupliactes

  def encoder(self):
    #label encoding
    from sklearn.preprocessing import LabelEncoder
    column_name = 'Class'
    label_encoder = LabelEncoder()
    self.df[column_name] = label_encoder.fit_transform(self.df[column_
name])
    print("DF after label encoding: ")
    print(self.df)

  def scaler(self):
    #using min max scaler to enclose all values btw 0 and 1
    scaler = MinMaxScaler()
    columns_to_scale = ['Area', 'Perimeter', 'MajorAxisLength', 'Minor
AxisLength', 'AspectRation',
                        'Eccentricity', 'ConvexArea', 'EquivDiameter',
'Extent', 'Solidity',
                        'roundness', 'Compactness', 'ShapeFactor1', 'S
hapeFactor2', 'ShapeFactor3', 'ShapeFactor4']
    self.df[columns_to_scale] = scaler.fit_transform(self.df[columns_t
o_scale])

  def smote_processor(self):
    df_x= self.df.drop(columns=['Class'])
    df_y= self.df['Class']
```

```
    smote = SMOTE(random_state=42)
    self.x_cleaned, self.y_cleaned = smote.fit_resample(df_x,df_y)

  # def noise_manager(self):

  #   noise_level = 0.01
  #   noise = np.random.normal(0, x_cleaned.std() * noise_level, x_cle
aned.shape)
  #   x_cleaned = smote_processor(self)
  #   df_x= x_cleaned+ df_x

  def noise_manager(self):
    smote = SMOTE(random_state=42) #using smote
    df_x = self.df.drop(columns=['Class'])
    df_y = self.df['Class']
    x_cleaned, y_cleaned = smote.fit_resample(df_x, df_y)
    # adding gaussian noise to clean data
    noise_level = 0.01
    noise = np.random.normal(0, x_cleaned.std() * noise_level, x_clean
ed.shape)
    # Add noise to the cleaned data
    x_cleaned_noisy = x_cleaned + noise
    self.df_x_noisy = x_cleaned_noisy
    print("Noise added to cleaned data successfully.")

  def feature_engineer(self,k=11):
    columns = ['Area','Perimeter','MajorAxisLength','MinorAxisLengt
h','AspectRation','Eccentricity','ConvexArea','EquivDiameter','Exten
t','Solidity','roundness','Compactness','ShapeFactor1','ShapeFactor
2','ShapeFactor3','ShapeFactor4']
    X_df = pd.DataFrame(self.x_cleaned, columns = columns)
    y_df = pd.DataFrame(self.y_cleaned, columns = ['Class'])
    final_df = pd.concat([X_df, y_df], axis = 1)

    #dimensionality reduction
    corr_matrix = final_df.corr()
    cmr_class = abs(corr_matrix['Class'])

    selected_features = cmr_class.nlargest(k).index.tolist()
    print("features that affect class prediction in ascending order:")
    print(selected_features)
    self.selected_features= selected_features

    final_df= final_df[selected_features]
    print("the refined DF now looks like: ")
    print(final_df.head())
    return selected_features,final_df

  # def feature_selection(self):
  #   final_df= self.df[self.selected_features]
  #   print(final_df.head())
  #   return final_df
```

In [125]:
```
train_data_preprocessor_child= data_preprocessor(df_train)
train_data_preprocessor_child.check_missing_values()
train_data_preprocessor_child.handle_missing_values(strategy='mode')
train_data_preprocessor_child.duplicate_removal()
train_data_preprocessor_child.encoder()
train_data_preprocessor_child.scaler()
train_data_preprocessor_child.smote_processor()
train_data_preprocessor_child.noise_manager()
selected_features,df_train= train_data_preprocessor_child.feature_engi
neer(k=11)

df_train.describe()
```

```
Area               0
Perimeter          0
MajorAxisLength    0
MinorAxisLength    0
AspectRation       0
Eccentricity       0
ConvexArea         0
EquivDiameter      0
Extent             0
Solidity           0
roundness          0
Compactness        0
ShapeFactor1       0
ShapeFactor2       0
ShapeFactor3       0
ShapeFactor4       0
Class              0
dtype: int64
Missing values have been handled using the 'mode' strategy.
duplicate row values:
        Area  Perimeter  MajorAxisLength  MinorAxisLength  AspectRatio
n  \
2038   51894    901.802       367.354766       182.162461      2.01663
3
3298   54860    911.589       369.730500       189.842876      1.94756
1
3398   60134    984.152       385.504338       199.989509      1.92762
3
3525   38891    791.343       319.499996       156.869619      2.03672
3
3648   52313    896.732       352.482089       189.300951      1.86202
0
4115   51142    898.882       357.519998       182.737640      1.95646
6
4250   46863    867.433       347.442755       172.128791      2.01850
5
4380   47134    870.142       360.775000       167.488454      2.15402
9
4419   62764   1003.767       409.207082       198.330199      2.06326
2
4461   44614    850.425       351.268986       162.308089      2.16421
1
4816   56413    929.481       377.880838       191.039673      1.97802
3
4982   57029    953.664       388.137971       187.927020      2.06536
5
5097   53978    924.848       366.147740       189.438667      1.93280
4
5164   53434    918.473       367.629058       185.545588      1.98134
1
5261   55145    930.030       365.745053       198.056410      1.84667
1
5749   65781   1039.257       409.713859       204.992832      1.99867
4
6444   46777    845.203       349.187769       171.167126      2.04004
0
6504   44710    832.120       332.068530       171.958309      1.93109
```

9

6570 2	43844	826.338	338.904539	165.758175	2.04457
6934 2	61505	982.866	395.761514	203.370492	1.94601
7034 2	50181	880.940	365.298096	175.486384	2.08163
7227 1	61911	985.026	408.790035	194.117337	2.10589
7276 8	42450	828.116	347.951525	156.469366	2.22376
7352 7	52462	903.657	364.976965	183.830683	1.98539
8038 6	40804	790.802	323.475648	163.287717	1.98101
8159 9	63882	1004.206	411.263403	198.765453	2.06908
8288 2	33518	702.956	277.571399	154.305581	1.79884
8294 4	60942	1020.605	413.970269	187.850317	2.20372
8381 1	61076	982.932	417.474629	187.277110	2.22918
8774 1	33954	716.750	277.368480	156.356326	1.77395
8922 5	59105	957.363	372.981361	205.545240	1.81459
9197 6	51515	896.279	375.980183	174.907355	2.14959
9364 3	45478	838.042	331.843270	176.276767	1.88251
9448 7	59664	982.497	410.284948	185.624314	2.21029
9496 5	63408	1005.966	412.551649	196.337705	2.10123
9507 6	43746	836.693	339.352567	165.411442	2.05156
9615 2	55689	925.555	371.517829	192.522981	1.92973
9697 5	48396	861.023	341.815489	181.429303	1.88401
9835 8	47180	850.406	348.615238	172.904633	2.01622
9848 5	60516	970.091	391.907258	199.308475	1.96633
9884 5	49730	879.912	365.825690	173.569194	2.10766
10279 8	46696	862.883	354.981027	168.054453	2.11229
10318 2	54689	911.217	372.146327	187.816831	1.98143
10437 1	56539	943.147	370.374563	196.675975	1.88317
10445 2	53679	949.603	354.397129	195.980161	1.80833
10495 9	54160	929.492	349.459450	202.934731	1.72202

```
10709  51903    908.476       366.544714       181.189268      2.02299
4

       Eccentricity  ConvexArea  EquivDiameter    Extent  Solidity  ro
undness  \
2038       0.868393       52605     257.047647  0.653059  0.986484
0.801871
3298       0.858112       55468     264.291357  0.691838  0.989039
0.829598
3398       0.854912       61079     276.703789  0.681745  0.984528
0.780199
3525       0.871168       39651     222.525412  0.650025  0.980833
0.780422
3648       0.843550       53085     258.083282  0.785198  0.985457
0.817512
4115       0.859506       51746     255.178402  0.783629  0.988328
0.795394
4250       0.868656       47538     244.269983  0.700703  0.985801
0.782651
4380       0.885706       47862     244.975249  0.623688  0.984790
0.782283
4419       0.874697       64158     282.689948  0.703995  0.978272
0.782807
4461       0.886848       45243     238.336546  0.633956  0.986097
0.775191
4816       0.862794       57007     268.006087  0.661240  0.989580
0.820556
4982       0.874971       57971     269.465356  0.784065  0.983750
0.787979
5097       0.855754       54949     262.158204  0.627272  0.982329
0.793023
5164       0.863290       54145     260.833820  0.815736  0.986869
0.795966
5261       0.840691       56953     264.976970  0.785016  0.968255
0.801165
5749       0.865834       66762     289.404510  0.642549  0.985306
0.765358
6444       0.871618       47247     244.045746  0.821413  0.990052
0.822849
6504       0.855478       45234     238.592833  0.694104  0.988416
0.811414
6570       0.872228       44434     236.270850  0.822620  0.986722
0.806872
6934       0.857867       63731     279.840308  0.653474  0.965072
0.800077
7034       0.877054       50724     252.769527  0.743973  0.989295
0.812562
7227       0.880062       62612     280.762415  0.787712  0.988804
0.801830
7276       0.893186       42820     232.484448  0.609388  0.991359
0.777867
7352       0.863892       53123     258.450562  0.809624  0.987557
0.807323
8038       0.863241       41636     227.932592  0.787570  0.980017
0.819931
8159       0.875452       64663     285.196579  0.754705  0.987922
0.796054
```

```
8288        0.831240       34023     206.582775  0.808383  0.985157
0.852377
8294        0.891115       61764     278.556573  0.678400  0.986691
0.735210
8381        0.893735       61597     278.862652  0.824071  0.991542
0.794390
8774        0.825970       34420     207.922042  0.799482  0.986461
0.830549
8922        0.834448       60667     274.326126  0.757630  0.974253
0.810365
9197        0.885204       52000     256.107273  0.804294  0.990673
0.805855
9364        0.847243       46206     240.633306  0.608011  0.984244
0.813729
9448        0.891801       60327     275.620326  0.623898  0.989010
0.776712
9496        0.879494       64200     284.136540  0.798791  0.987664
0.787385
9507        0.873161       44442     236.006646  0.713778  0.984339
0.785264
9615        0.855255       56406     266.280749  0.725921  0.987289
0.816911
9697        0.847509       48942     248.233159  0.704413  0.988844
0.820332
9835        0.868336       47729     245.094761  0.710040  0.988498
0.819814
9848        0.861026       61355     277.581275  0.760758  0.986325
0.808081
9884        0.880278       50263     251.631084  0.789365  0.989396
0.807142
10279       0.880838       47259     243.834357  0.778228  0.988087
0.788108
10318       0.863303       55206     263.879134  0.726861  0.990635
0.827687
10437       0.847360       57599     268.305219  0.695728  0.981597
0.798729
10445       0.833184       54804     261.431110  0.715386  0.979472
0.748049
10495       0.814110       55623     262.599798  0.749505  0.973698
0.787766
10709       0.869282       52611     257.069936  0.708825  0.986543
0.790270

       Compactness  ShapeFactor1  ShapeFactor2  ShapeFactor3  ShapeFac
tor4  \
2038      0.699726      0.007079      0.001047      0.489616      0.98
7376
3298      0.714822      0.006740      0.001085      0.510970      0.99
5145
3398      0.717771      0.006411      0.001050      0.515195      0.99
3102
3525      0.696480      0.008215      0.001192      0.485085      0.98
7983
3648      0.732188      0.006738      0.001195      0.536100      0.99
8228
4115      0.713746      0.006991      0.001119      0.509433      0.99
6689
```

4250 7708	0.703051	0.007414	0.001117	0.494281	0.99
4380 3169	0.679025	0.007654	0.001004	0.461075	0.99
4419 4666	0.690824	0.006520	0.000916	0.477237	0.98
4461 6326	0.678502	0.007874	0.001029	0.460364	0.99
4816 4972	0.709234	0.006698	0.001045	0.503013	0.99
4982 5475	0.694251	0.006806	0.000975	0.481985	0.99
5097 0836	0.715990	0.006783	0.001100	0.512642	0.99
5164 7395	0.709503	0.006880	0.001075	0.503394	0.99
5261 9279	0.724485	0.006632	0.001127	0.524879	0.96
5749 7221	0.706358	0.006228	0.000956	0.498941	0.99
6444 6467	0.698895	0.007465	0.001099	0.488455	0.99
6504 6928	0.718505	0.007427	0.001221	0.516249	0.99
6570 3729	0.697160	0.007730	0.001126	0.486033	0.99
6934 2969	0.707093	0.006435	0.000992	0.499981	0.97
7034 6687	0.691954	0.007280	0.001029	0.478801	0.99
7227 3375	0.686813	0.006603	0.000906	0.471712	0.99
7276 2750	0.668152	0.008197	0.001008	0.446427	0.99
7352 5569	0.708128	0.006957	0.001079	0.501446	0.99
8038 3598	0.704636	0.007928	0.001206	0.496512	0.98
8159 5010	0.693465	0.006438	0.000918	0.480893	0.99
8288 6396	0.744251	0.008281	0.001567	0.553909	0.99
8294 7805	0.672890	0.006793	0.000859	0.452781	0.99
8381 4640	0.667975	0.006835	0.000839	0.446191	0.99
8774 6847	0.749624	0.008169	0.001591	0.561936	0.99
8922 1612	0.735496	0.006310	0.001139	0.540954	0.98
9197 7403	0.681172	0.007298	0.000969	0.463996	0.99
9364 9882	0.725141	0.007297	0.001245	0.525830	0.98
9448 7475	0.671778	0.006877	0.000864	0.451285	0.99
9496	0.688730	0.006506	0.000903	0.474348	0.99

```
6718
9507     0.695462     0.007757     0.001119     0.483667     0.99
2274
9615     0.716737     0.006671     0.001086     0.513713     0.99
1328
9697     0.726220     0.007063     0.001212     0.527395     0.99
3620
9835     0.703052     0.007389     0.001114     0.494283     0.99
6587
9848     0.708283     0.006476     0.001005     0.501665     0.98
6441
9884     0.687844     0.007356     0.001016     0.473130     0.99
7199
10279    0.686894     0.007602     0.001044     0.471823     0.99
6632
10318    0.709074     0.006805     0.001061     0.502785     0.99
6235
10437    0.724416     0.006551     0.001113     0.524778     0.98
8248
10445    0.737678     0.006602     0.001206     0.544169     0.98
4039
10495    0.751446     0.006452     0.001269     0.564671     0.97
2379
10709    0.701333     0.007062     0.001054     0.491868     0.99
5046

       Class
2038   HOROZ
3298   HOROZ
3398   HOROZ
3525   HOROZ
3648   HOROZ
4115   HOROZ
4250   HOROZ
4380   HOROZ
4419   HOROZ
4461   HOROZ
4816   HOROZ
4982   HOROZ
5097   HOROZ
5164   HOROZ
5261   HOROZ
5749   HOROZ
6444   HOROZ
6504   HOROZ
6570   HOROZ
6934   HOROZ
7034   HOROZ
7227   HOROZ
7276   HOROZ
7352   HOROZ
8038   HOROZ
8159   HOROZ
8288   HOROZ
8294   HOROZ
8381   HOROZ
8774   HOROZ
```

```
8922   HOROZ
9197   HOROZ
9364   HOROZ
9448   HOROZ
9496   HOROZ
9507   HOROZ
9615   HOROZ
9697   HOROZ
9835   HOROZ
9848   HOROZ
9884   HOROZ
10279  HOROZ
10318  HOROZ
10437  HOROZ
10445  HOROZ
10495  HOROZ
10709  HOROZ
DF after label encoding:
        Area  Perimeter  MajorAxisLength  MinorAxisLength  AspectRatio
n  \
0      42339    741.226       260.199330       207.306394      1.25514
4
1      68247   1088.754       370.368146       237.863792      1.55706
0
2      37856    708.716       248.430330       194.360324      1.27819
5
3      33143    648.385       222.526309       189.737379      1.17281
2
4      29925    647.570       237.714031       161.004849      1.47644
0
...      ...        ...              ...              ...
...
10884  34948    697.453       264.444305       168.744463      1.56712
9
10885  45550    797.549       296.932493       196.391957      1.51193
8
10886  27409    641.327       232.735907       150.457291      1.54685
7
10887  38487    732.235       281.095672       174.608613      1.60986
1
10888  40465    743.838       273.311060       189.088375      1.44541
4

       Eccentricity  ConvexArea  EquivDiameter    Extent  Solidity  ro
undness  \
0          0.604347       42676     232.180294  0.771202  0.992103
0.968387
1          0.766507       70172     294.779204  0.767683  0.972567
0.723492
2          0.622835       38232     219.544429  0.744640  0.990165
0.947109
3          0.522480       33377     205.423899  0.769980  0.992989
0.990685
4          0.735703       30321     195.196551  0.785309  0.986940
0.896748
...             ...         ...            ...       ...       ...
...
```

```
10884      0.769946      35262     210.943536  0.700417  0.991095
0.902822
10885      0.750031      46093     240.823714  0.775662  0.988219
0.899878
10886      0.762938      27842     186.810660  0.794464  0.984448
0.837421
10887      0.783675      38860     221.366597  0.741389  0.990401
0.902034
10888      0.722048      40916     226.983784  0.791321  0.988977
0.919036

       Compactness  ShapeFactor1  ShapeFactor2  ShapeFactor3  ShapeFac
tor4  \
0         0.892317      0.006146      0.002403      0.796230      0.99
9383
1         0.795909      0.005427      0.001343      0.633471      0.98
6352
2         0.883726      0.006563      0.002469      0.780972      0.99
8235
3         0.923144      0.006714      0.003008      0.852195      0.99
9465
4         0.821140      0.007944      0.002228      0.674271      0.99
5521
...            ...           ...           ...           ...       
...
10884     0.797686      0.007567      0.001890      0.636303      0.99
7169
10885     0.811039      0.006519      0.001740      0.657784      0.99
4528
10886     0.802672      0.008491      0.002174      0.644283      0.99
6613
10887     0.787513      0.007304      0.001733      0.620177      0.99
8400
10888     0.830496      0.006754      0.001982      0.689724      0.99
6937

       Class
0          5
1          0
2          5
3          5
4          3
...      ...
10884      3
10885      6
10886      3
10887      3
10888      3

[10889 rows x 17 columns]
Noise added to cleaned data successfully.
features that affect class prediction in ascending order:
['Class', 'Perimeter', 'EquivDiameter', 'ShapeFactor1', 'MinorAxisLeng
th', 'MajorAxisLength', 'ConvexArea', 'Area', 'ShapeFactor2', 'roundne
ss', 'Solidity']
the refined DF now looks like:
   Class  Perimeter  EquivDiameter  ShapeFactor1  MinorAxisLength  \
```

```
0      5   0.148216       0.173808      0.433141         0.258192
1      0   0.386146       0.327188      0.338513         0.351237
2      5   0.125959       0.142848      0.488028         0.218772
3      5   0.084654       0.108250      0.507990         0.204695
4      3   0.084096       0.083191      0.669867         0.117206

   MajorAxisLength  ConvexArea      Area  ShapeFactor2  roundness  Sol
idity
0         0.137950    0.090660  0.093593      0.611221   0.955499  0.9
48628
1         0.336360    0.204009  0.204218      0.258937   0.466752  0.5
58764
2         0.116755    0.072340  0.074450      0.633029   0.913033  0.9
09953
3         0.070103    0.052326  0.054326      0.812085   1.000000  0.9
66307
4         0.097455    0.039728  0.040586      0.552861   0.812526  0.8
45583
```

Out[125]:

	Class	Perimeter	EquivDiameter	ShapeFactor1	MinorAxisLength	MajorAxisLen
count	19705.000000	19705.000000	19705.000000	19705.000000	19705.000000	19705.000
mean	3.000000	0.305797	0.305950	0.414685	0.321027	0.323
std	2.000051	0.203421	0.209135	0.180775	0.204379	0.207
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	1.000000	0.149545	0.160237	0.318919	0.189968	0.153
50%	3.000000	0.264853	0.244126	0.450857	0.246699	0.303
75%	5.000000	0.376190	0.363272	0.534742	0.366792	0.402
max	6.000000	1.000000	1.000000	1.000000	1.000000	1.000

In [126]:

```
# a=df_train.columns.tolist()
# print(a)
```

In [127]:

```
class data_splitter():
  def __init__(self,df):
    self.df=df

  def train_val_split(self):
    x=self.df.drop(columns=['Class'])
    y=self.df['Class']
    x_train, x_val, y_train, y_val = train_test_split(x,y, test_size=
0.2, random_state=42)
    return x_train, x_val, y_train, y_val

  def test_x_y_split(self):
    x_test=self.df.drop(columns=['Class'])
    y_test=self.df['Class']
    return x_test,y_test
```

In [128]:
```
data_splitter_child=data_splitter(df_train)
x_train, x_val, y_train, y_val= data_splitter_child.train_val_split()
```

In [100]:
```
class Models:
  def __init__(self, x_train, x_test, y_train, y_test):
      self.x_train = x_train
      self.y_train = y_train
      self.x_test = x_test
      self.y_test = y_test

  def neural_network(self):
      # Function to create the Keras model
      def create_model():
          model = Sequential()
          model.add(Dense(32, input_shape=(10,), activation='relu'))
          model.add(Dense(32, activation='relu'))
          model.add(Dense(7, activation='softmax'))
          model.compile(optimizer='adam', loss='categorical_crossentro
py', metrics=['accuracy'])
          return model

      # Convert the labels to categorical format
      y_train_cat = to_categorical(self.y_train, num_classes=7)
      y_test_cat = to_categorical(self.y_test, num_classes=7)

      # Train the model
      model = create_model()
      history = model.fit(self.x_train, y_train_cat, epochs=50, batch_
size=32, validation_split=0.1)

      # Evaluate the model
      neural_network_loss, neural_network_accuracy = model.evaluate(se
lf.x_test, y_test_cat)
      print(f'Test loss: {neural_network_loss}, Test accuracy: {neural
_network_accuracy}')

      # Predict on the test set
      predictions = model.predict(self.x_test)
      predictions_classes = predictions.argmax(axis=1)

      # Calculate F1 scores
      f1_macro = f1_score(self.y_test, predictions_classes, average='m
acro')
      f1_micro = f1_score(self.y_test, predictions_classes, average='m
icro')
      print("Neural Network Classifier Macro-averaged F1 Score:", f1_m
acro)
      print("Neural Network Classifier Micro-averaged F1 Score:", f1_m
icro)

      # Generate confusion matrix
      conf_matrix = confusion_matrix(self.y_test, predictions_classes)
      print("Neural Network Classifier Confusion Matrix:")
      print(conf_matrix)

      # Perform cross-validation
      cross_val_scores = []
      for i in range(5):
          # Create a new model for each fold
          model = create_model()
```

```
            model.fit(self.x_train, y_train_cat, epochs=10, batch_size=3
2)
            # Evaluate the model
            score = model.evaluate(self.x_test, y_test_cat)
            cross_val_scores.append(score[1])

        print(f'Neural Network Classifier cross-validation scores: {cros
s_val_scores}')
        print(f'Neural Network Classifier average cross-validation scor
e: {np.mean(cross_val_scores)}')

        # Return the model, predictions, and history
        return model, predictions, history

    def nearest_means(self):
        model = NearestCentroid()
        model.fit(self.x_train, self.y_train)
        predictions = model.predict(self.x_test)
        nearest_means_accuracy = accuracy_score(self.y_test, prediction
s)
        print(f"Nearest Means Classifier accuracy: {nearest_means_accura
cy}")

        # Calculate the macro and micro F1 scores
        f1_macro = f1_score(self.y_test, predictions, average='macro')
        f1_micro = f1_score(self.y_test, predictions, average='micro')
        print("Nearest Means Classifier Macro-averaged F1 Score:", f1_ma
cro)
        print("Nearest Means Classifier Micro-averaged F1 Score:", f1_mi
cro)

        # Generate a confusion matrix
        conf_matrix = confusion_matrix(self.y_test, predictions)
        print("Nearest Means Classifier Confusion Matrix:")
        print(conf_matrix)

        # Perform cross-validation
        cv_scores = cross_val_score(model, self.x_train, self.y_train, c
v=5, scoring='accuracy')
        print(f'Nearest Means Classifier cross-validation scores: {cv_sc
ores}')
        print(f'Nearest Means Classifier average cross-validation score:
{np.mean(cv_scores)}')

        return model, predictions

    def gradient_boosting_classifier(self):
        # Initialize a Gradient Boosting Classifier
        gbc = GradientBoostingClassifier(random_state=0)

        # Fit the model
        gbc.fit(self.x_train, self.y_train)

        # Predict on the test set
        predictions = gbc.predict(self.x_test)

        # Calculate the accuracy
```

```
        gbc_accuracy = accuracy_score(self.y_test, predictions)
        print(f'Gradient Boosting Classifier accuracy: {gbc_accuracy}')

        # Calculate the macro and micro F1 scores
        f1_macro = f1_score(self.y_test, predictions, average='macro')
        f1_micro = f1_score(self.y_test, predictions, average='micro')
        print("Gradient Boosting Classifier Macro-averaged F1 Score:", f1_macro)
        print("Gradient Boosting Classifier Micro-averaged F1 Score:", f1_micro)

        # Generate a confusion matrix
        conf_matrix = confusion_matrix(self.y_test, predictions)
        print("Gradient Boosting Classifier Confusion Matrix:")
        print(conf_matrix)

        # Perform cross-validation
        cv_scores = cross_val_score(gbc, self.x_train, self.y_train, cv=5, scoring='accuracy')
        print(f'Gradient Boosting Classifier cross-validation scores: {cv_scores}')
        print(f'Gradient Boosting Classifier average cross-validation score: {np.mean(cv_scores)}')

        return gbc, predictions

    def xgb_classifier(self):
        xgb = XGBClassifier(random_state=0, use_label_encoder=False, eval_metric='mlogloss')
        xgb.fit(self.x_train, self.y_train)
        predictions = xgb.predict(self.x_test)
        xgb_accuracy = accuracy_score(self.y_test, predictions)
        print(f'XGBoost Classifier accuracy: {xgb_accuracy}')
        f1_macro = f1_score(self.y_test, predictions, average='macro')
        f1_micro = f1_score(self.y_test, predictions, average='micro')
        print("XGBoost Classifier Macro-averaged F1 Score:", f1_macro)
        print("XGBoost Classifier Micro-averaged F1 Score:", f1_micro)

        # confusion matrix
        conf_matrix = confusion_matrix(self.y_test, predictions)
        print("XGBoost Classifier Confusion Matrix:")
        print(conf_matrix)

        # cross-validation
        cv_scores = cross_val_score(xgb, self.x_train, self.y_train, cv=5, scoring='accuracy')
        print(f'XGBoost Classifier cross-validation scores: {cv_scores}')
        print(f'XGBoost Classifier average cross-validation score: {np.mean(cv_scores)}')

        return xgb, predictions

    def trivial_solution(self):
      # class probabilities based on training data
      class_counts = np.bincount(self.y_train)
      class_probabilities = class_counts / len(self.y_train)
```

```
    predicted_labels = np.random.choice(np.arange(len(class_counts)),
size=len(self.y_test), p=class_probabilities)
    trivial_accuracy = accuracy_score(self.y_test, predicted_labels)
    print("Trivial Solution Accuracy:", trivial_accuracy)

    # Calculate F1-score
    f1_macro = f1_score(self.y_test, predicted_labels, average='macr
o')
    f1_micro = f1_score(self.y_test, predicted_labels, average='micr
o')
    print("Trivial Solution Macro-averaged F1 Score:", f1_macro)
    print("Trivial Solution Micro-averaged F1 Score:", f1_micro)

    # confusion matrix
    conf_matrix = confusion_matrix(self.y_test, predicted_labels)
    print("Trivial Solution Confusion Matrix:")
    print(conf_matrix)

    def trivial_scoring_function(X, y):
        predicted_labels = np.random.choice(np.arange(len(class_count
s)), size=len(y), p=class_probabilities)
        return accuracy_score(y, predicted_labels)
```

In [101]:
```
classifier_model_child= Models(x_train, x_val, y_train, y_val)
print("===============================================================
================")
print("Model 1 : Trivial Solution - probability based classifier: ")
accuray = classifier_model_child.trivial_solution()
print("===============================================================
================")

print("===============================================================
================")
print("Model 2 : Baseline Solution - Nearest means classifier: ")
model, predictions = classifier_model_child.nearest_means()
print("===============================================================
================")

print("===============================================================
================")
print("Model 3 : Neural network classifier: ")
model, predictions, history = classifier_model_child.neural_network()
print("===============================================================
================")

print("===============================================================
================")
print("Model 4 : Gradient Boosting Classifier: ")
gbc,predictions= classifier_model_child.gradient_boosting_classifier()
print("===============================================================
================")

print("===============================================================
================")
print("Model 5 : XGB Classifier: ")
xgb, predictions= classifier_model_child.xgb_classifier()
print("===============================================================
================")
```

```
======================================================================
=========
Model 1 : Trivial Solution - probability based classifier:
Trivial Solution Accuracy: 0.14209591474245115
Trivial Solution Macro-averaged F1 Score: 0.14198560293579368
Trivial Solution Micro-averaged F1 Score: 0.14209591474245115
Trivial Solution Confusion Matrix:
[[79 85 72 73 78 70 77]
 [83 96 76 86 69 86 92]
 [84 78 82 90 80 76 71]
 [78 78 81 76 70 90 83]
 [89 88 84 99 73 72 90]
 [82 69 82 80 75 80 84]
 [92 87 75 79 71 77 74]]
======================================================================
=========
======================================================================
=========
Model 2 : Baseline Solution - Nearest means classifier:
Nearest Means Classifier accuracy: 0.8774422735346359
Nearest Means Classifier Macro-averaged F1 Score: 0.8756412918380011
Nearest Means Classifier Micro-averaged F1 Score: 0.8774422735346359
Nearest Means Classifier Confusion Matrix:
[[385   0 107   0  15   0  27]
 [  0 587   1   0   0   0   0]
 [ 61   0 477   0  15   0   8]
 [  0   0   0 470   1  21  64]
 [ 13   0   7   7 548   0  20]
 [  1   0   0   5   0 509  37]
 [  2   0   0  37  27   7 482]]
Nearest Means Classifier cross-validation scores: [0.88740882 0.892483
35 0.88233428 0.89406914 0.88864213]
Nearest Means Classifier average cross-validation score: 0.88898754469
59709
======================================================================
=========
======================================================================
=========
Model 3 : Neural network classifier:
Epoch 1/50
444/444 [==============================] - 3s 3ms/step - loss: 1.1277
- accuracy: 0.5598 - val_loss: 0.5648 - val_accuracy: 0.8167
Epoch 2/50
444/444 [==============================] - 1s 2ms/step - loss: 0.3870
- accuracy: 0.8819 - val_loss: 0.3267 - val_accuracy: 0.8782
Epoch 3/50
444/444 [==============================] - 1s 2ms/step - loss: 0.2874
- accuracy: 0.8987 - val_loss: 0.2810 - val_accuracy: 0.8960
Epoch 4/50
444/444 [==============================] - 1s 2ms/step - loss: 0.2594
- accuracy: 0.9082 - val_loss: 0.2643 - val_accuracy: 0.9087
Epoch 5/50
444/444 [==============================] - 1s 2ms/step - loss: 0.2434
- accuracy: 0.9115 - val_loss: 0.2524 - val_accuracy: 0.9150
Epoch 6/50
444/444 [==============================] - 1s 2ms/step - loss: 0.2328
- accuracy: 0.9172 - val_loss: 0.2412 - val_accuracy: 0.9131
```

```
Epoch 7/50
444/444 [==============================] - 1s 2ms/step - loss: 0.2252
- accuracy: 0.9206 - val_loss: 0.2308 - val_accuracy: 0.9214
Epoch 8/50
444/444 [==============================] - 1s 2ms/step - loss: 0.2179
- accuracy: 0.9212 - val_loss: 0.2276 - val_accuracy: 0.9220
Epoch 9/50
444/444 [==============================] - 1s 2ms/step - loss: 0.2125
- accuracy: 0.9230 - val_loss: 0.2168 - val_accuracy: 0.9271
Epoch 10/50
444/444 [==============================] - 1s 3ms/step - loss: 0.2098
- accuracy: 0.9236 - val_loss: 0.2116 - val_accuracy: 0.9258
Epoch 11/50
444/444 [==============================] - 2s 4ms/step - loss: 0.2034
- accuracy: 0.9262 - val_loss: 0.2170 - val_accuracy: 0.9252
Epoch 12/50
444/444 [==============================] - 2s 4ms/step - loss: 0.2026
- accuracy: 0.9263 - val_loss: 0.2132 - val_accuracy: 0.9252
Epoch 13/50
444/444 [==============================] - 2s 4ms/step - loss: 0.2015
- accuracy: 0.9265 - val_loss: 0.2184 - val_accuracy: 0.9226
Epoch 14/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1992
- accuracy: 0.9277 - val_loss: 0.2043 - val_accuracy: 0.9264
Epoch 15/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1969
- accuracy: 0.9280 - val_loss: 0.2007 - val_accuracy: 0.9283
Epoch 16/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1948
- accuracy: 0.9290 - val_loss: 0.2022 - val_accuracy: 0.9309
Epoch 17/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1941
- accuracy: 0.9287 - val_loss: 0.2000 - val_accuracy: 0.9309
Epoch 18/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1947
- accuracy: 0.9288 - val_loss: 0.2038 - val_accuracy: 0.9264
Epoch 19/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1936
- accuracy: 0.9297 - val_loss: 0.2009 - val_accuracy: 0.9290
Epoch 20/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1936
- accuracy: 0.9289 - val_loss: 0.2074 - val_accuracy: 0.9271
Epoch 21/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1933
- accuracy: 0.9300 - val_loss: 0.2030 - val_accuracy: 0.9290
Epoch 22/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1926
- accuracy: 0.9292 - val_loss: 0.1992 - val_accuracy: 0.9309
Epoch 23/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1921
- accuracy: 0.9298 - val_loss: 0.1991 - val_accuracy: 0.9296
Epoch 24/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1911
- accuracy: 0.9310 - val_loss: 0.1942 - val_accuracy: 0.9302
Epoch 25/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1911
- accuracy: 0.9298 - val_loss: 0.1994 - val_accuracy: 0.9302
```

```
Epoch 26/50
444/444 [==============================] - 2s 3ms/step - loss: 0.1924
- accuracy: 0.9281 - val_loss: 0.1954 - val_accuracy: 0.9315
Epoch 27/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1916
- accuracy: 0.9311 - val_loss: 0.1984 - val_accuracy: 0.9277
Epoch 28/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1917
- accuracy: 0.9297 - val_loss: 0.1959 - val_accuracy: 0.9283
Epoch 29/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1894
- accuracy: 0.9311 - val_loss: 0.1951 - val_accuracy: 0.9290
Epoch 30/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1906
- accuracy: 0.9308 - val_loss: 0.1952 - val_accuracy: 0.9296
Epoch 31/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1895
- accuracy: 0.9314 - val_loss: 0.2012 - val_accuracy: 0.9252
Epoch 32/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1885
- accuracy: 0.9301 - val_loss: 0.1968 - val_accuracy: 0.9290
Epoch 33/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1885
- accuracy: 0.9322 - val_loss: 0.1973 - val_accuracy: 0.9334
Epoch 34/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1884
- accuracy: 0.9306 - val_loss: 0.1996 - val_accuracy: 0.9309
Epoch 35/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1885
- accuracy: 0.9301 - val_loss: 0.1984 - val_accuracy: 0.9258
Epoch 36/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1864
- accuracy: 0.9309 - val_loss: 0.1984 - val_accuracy: 0.9283
Epoch 37/50
444/444 [==============================] - 2s 3ms/step - loss: 0.1854
- accuracy: 0.9309 - val_loss: 0.1937 - val_accuracy: 0.9290
Epoch 38/50
444/444 [==============================] - 2s 4ms/step - loss: 0.1865
- accuracy: 0.9313 - val_loss: 0.1961 - val_accuracy: 0.9328
Epoch 39/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1864
- accuracy: 0.9321 - val_loss: 0.1898 - val_accuracy: 0.9353
Epoch 40/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1870
- accuracy: 0.9320 - val_loss: 0.1944 - val_accuracy: 0.9328
Epoch 41/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1842
- accuracy: 0.9325 - val_loss: 0.1957 - val_accuracy: 0.9321
Epoch 42/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1847
- accuracy: 0.9306 - val_loss: 0.1912 - val_accuracy: 0.9334
Epoch 43/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1838
- accuracy: 0.9342 - val_loss: 0.1932 - val_accuracy: 0.9309
Epoch 44/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1839
- accuracy: 0.9325 - val_loss: 0.1929 - val_accuracy: 0.9296
```

```
Epoch 45/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1850
- accuracy: 0.9306 - val_loss: 0.1905 - val_accuracy: 0.9321
Epoch 46/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1835
- accuracy: 0.9328 - val_loss: 0.1875 - val_accuracy: 0.9334
Epoch 47/50
444/444 [==============================] - 2s 3ms/step - loss: 0.1850
- accuracy: 0.9341 - val_loss: 0.1953 - val_accuracy: 0.9283
Epoch 48/50
444/444 [==============================] - 2s 4ms/step - loss: 0.1839
- accuracy: 0.9331 - val_loss: 0.1889 - val_accuracy: 0.9309
Epoch 49/50
444/444 [==============================] - 2s 4ms/step - loss: 0.1834
- accuracy: 0.9321 - val_loss: 0.1874 - val_accuracy: 0.9321
Epoch 50/50
444/444 [==============================] - 2s 4ms/step - loss: 0.1825
- accuracy: 0.9325 - val_loss: 0.1931 - val_accuracy: 0.9315
124/124 [==============================] - 0s 2ms/step - loss: 0.2000
- accuracy: 0.9272
Test loss: 0.20001496374607086, Test accuracy: 0.9271758198738098
124/124 [==============================] - 0s 2ms/step
Neural Network Classifier Macro-averaged F1 Score: 0.9264966324209135
Neural Network Classifier Micro-averaged F1 Score: 0.9271758436944938
Neural Network Classifier Confusion Matrix:
[[491   0  30   1   5   3   4]
 [  0 588   0   0   0   0   0]
 [ 31   0 507   0  19   0   4]
 [  1   0   0 515   0   6  34]
 [  2   0   4   9 572   0   8]
 [  2   0   0  16   0 508  26]
 [  3   0   1  48  25   5 473]]
Epoch 1/10
493/493 [==============================] - 2s 2ms/step - loss: 1.0425
- accuracy: 0.7260
Epoch 2/10
493/493 [==============================] - 1s 2ms/step - loss: 0.3884
- accuracy: 0.8733
Epoch 3/10
493/493 [==============================] - 1s 2ms/step - loss: 0.3033
- accuracy: 0.8941
Epoch 4/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2650
- accuracy: 0.9071
Epoch 5/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2414
- accuracy: 0.9156
Epoch 6/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2271
- accuracy: 0.9210
Epoch 7/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2161
- accuracy: 0.9237
Epoch 8/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2117
- accuracy: 0.9220
Epoch 9/10
```

```
493/493 [==============================] - 1s 2ms/step - loss: 0.2059
- accuracy: 0.9269
Epoch 10/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2042
- accuracy: 0.9267
124/124 [==============================] - 0s 3ms/step - loss: 0.2145
- accuracy: 0.9211
Epoch 1/10
493/493 [==============================] - 2s 3ms/step - loss: 1.0754
- accuracy: 0.6597
Epoch 2/10
493/493 [==============================] - 1s 2ms/step - loss: 0.3891
- accuracy: 0.8777
Epoch 3/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2922
- accuracy: 0.8956
Epoch 4/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2591
- accuracy: 0.9072
Epoch 5/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2385
- accuracy: 0.9156
Epoch 6/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2250
- accuracy: 0.9201
Epoch 7/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2172
- accuracy: 0.9239
Epoch 8/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2099
- accuracy: 0.9256
Epoch 9/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2086
- accuracy: 0.9239
Epoch 10/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2041
- accuracy: 0.9274
124/124 [==============================] - 0s 2ms/step - loss: 0.2156
- accuracy: 0.9226
Epoch 1/10
493/493 [==============================] - 3s 4ms/step - loss: 1.1800
- accuracy: 0.6326
Epoch 2/10
493/493 [==============================] - 2s 3ms/step - loss: 0.4093
- accuracy: 0.8799
Epoch 3/10
493/493 [==============================] - 1s 3ms/step - loss: 0.3036
- accuracy: 0.8933
Epoch 4/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2737
- accuracy: 0.9007
Epoch 5/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2568
- accuracy: 0.9074
Epoch 6/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2456
- accuracy: 0.9121
```

```
Epoch 7/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2333
- accuracy: 0.9182
Epoch 8/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2247
- accuracy: 0.9214
Epoch 9/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2186
- accuracy: 0.9225
Epoch 10/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2127
- accuracy: 0.9239
124/124 [==============================] - 0s 2ms/step - loss: 0.2577
- accuracy: 0.9071
Epoch 1/10
493/493 [==============================] - 2s 2ms/step - loss: 1.0703
- accuracy: 0.6382
Epoch 2/10
493/493 [==============================] - 1s 2ms/step - loss: 0.4115
- accuracy: 0.8736
Epoch 3/10
493/493 [==============================] - 1s 2ms/step - loss: 0.3033
- accuracy: 0.8956
Epoch 4/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2649
- accuracy: 0.9085
Epoch 5/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2446
- accuracy: 0.9159
Epoch 6/10
493/493 [==============================] - 1s 3ms/step - loss: 0.2305
- accuracy: 0.9212
Epoch 7/10
493/493 [==============================] - 1s 3ms/step - loss: 0.2224
- accuracy: 0.9237
Epoch 8/10
493/493 [==============================] - 1s 3ms/step - loss: 0.2169
- accuracy: 0.9238
Epoch 9/10
493/493 [==============================] - 1s 3ms/step - loss: 0.2122
- accuracy: 0.9232
Epoch 10/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2085
- accuracy: 0.9255
124/124 [==============================] - 0s 2ms/step - loss: 0.2135
- accuracy: 0.9218
Epoch 1/10
493/493 [==============================] - 2s 2ms/step - loss: 1.0056
- accuracy: 0.7056
Epoch 2/10
493/493 [==============================] - 1s 2ms/step - loss: 0.3768
- accuracy: 0.8684
Epoch 3/10
493/493 [==============================] - 1s 2ms/step - loss: 0.3106
- accuracy: 0.8826
Epoch 4/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2870
```

```
- accuracy: 0.8932
Epoch 5/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2714
- accuracy: 0.8973
Epoch 6/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2605
- accuracy: 0.9029
Epoch 7/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2501
- accuracy: 0.9076
Epoch 8/10
493/493 [==============================] - 1s 3ms/step - loss: 0.2423
- accuracy: 0.9111
Epoch 9/10
493/493 [==============================] - 1s 3ms/step - loss: 0.2345
- accuracy: 0.9162
Epoch 10/10
493/493 [==============================] - 1s 3ms/step - loss: 0.2293
- accuracy: 0.9168
124/124 [==============================] - 0s 2ms/step - loss: 0.2417
- accuracy: 0.9112
Neural Network Classifier cross-validation scores: [0.921086013317108
2, 0.922608494758606, 0.9071301817893982, 0.9218472242355347, 0.911190
0329589844]
Neural Network Classifier average cross-validation score: 0.9167723894
119263
======================================================================
=========
======================================================================
=========
Model 4 : Gradient Boosting Classifier:
Gradient Boosting Classifier accuracy: 0.9370718091854859
Gradient Boosting Classifier Macro-averaged F1 Score: 0.93635170297628
32
Gradient Boosting Classifier Micro-averaged F1 Score: 0.93707180918548
59
Gradient Boosting Classifier Confusion Matrix:
[[490   0  34   0   3   1   6]
 [  0 588   0   0   0   0   0]
 [ 25   0 526   0   8   0   2]
 [  0   0   0 504   0  18  34]
 [  2   0  10   4 567   0  12]
 [  1   0   0  11   0 530  10]
 [  1   0   4  44  11   7 488]]
Gradient Boosting Classifier cross-validation scores: [0.93973993 0.93
46654  0.93751982 0.94037425 0.93876904]
Gradient Boosting Classifier average cross-validation score: 0.9382136
865864595
======================================================================
=========
======================================================================
=========
Model 5 : XGB Classifier:
XGBoost Classifier accuracy: 0.9408779497589445
XGBoost Classifier Macro-averaged F1 Score: 0.9404292308757503
XGBoost Classifier Micro-averaged F1 Score: 0.9408779497589445
XGBoost Classifier Confusion Matrix:
```

```
[[502   1  24   0   2   0   5]
 [  0 588   0   0   0   0   0]
 [ 17   0 530   0  11   0   3]
 [  0   0   0 505   1  14  36]
 [  4   0  10   4 565   0  12]
 [  0   0   0  15   0 526  11]
 [  1   0   3  43  10   6 492]]
XGBoost Classifier cross-validation scores: [0.94766889 0.94259435 0.9
4354583 0.94576594 0.94321066]
XGBoost Classifier average cross-validation score: 0.9445571335654867
======================================================================
=========
```

TESTING THE MODELS

In [129]:
```
test_data_path="/content/drive/MyDrive/ML-1_Project/dry_bean_classific
ation_test.csv"
data_loader_test_child=data_loader(test_data_path)
df_test=data_loader_test_child.load_data()

df_test.describe()
```

Out[129]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity
count	2722.000000	2722.000000	2722.000000	2722.000000	2722.000000	2722.00000
mean	53091.710874	855.966948	320.973968	201.818580	1.590271	0.75372
std	29401.815156	215.769276	86.415207	44.958459	0.247299	0.08992
min	22205.000000	553.600000	190.282632	132.298336	1.060798	0.33368
25%	36355.000000	703.252250	252.896832	175.369379	1.437686	0.71846
50%	44560.000000	793.932500	297.389957	191.736938	1.556550	0.76633
75%	61140.750000	978.567750	378.596290	215.125930	1.714538	0.81229
max	231066.000000	1847.940000	722.494068	460.198497	2.334680	0.90362

In [103]:
```
test_data_preprocessor_child= data_preprocessor(df_test)
test_data_preprocessor_child.encoder()
test_data_preprocessor_child.scaler()

df_test= df_test[selected_features]
```

```
DF after label encoding:
       Area  Perimeter  MajorAxisLength  MinorAxisLength  AspectRation
\
0     28734    638.018       200.524796       182.734419      1.097356
1     30140    620.134       201.847882       190.279279      1.060798
2     30279    634.927       212.560556       181.510182      1.171067
3     30834    631.934       217.227813       180.897469      1.200834
4     31091    638.558       210.486255       188.326848      1.117665
...     ...        ...              ...              ...           ...
2717  41966    746.121       273.508678       195.449153      1.399385
2718  41995    765.763       284.073178       188.591957      1.506285
2719  42008    759.454       280.332717       191.218136      1.466036
2720  42008    773.158       294.492203       181.847359      1.619447
2721  42139    759.321       281.539928       191.187979      1.472582

      Eccentricity  ConvexArea  EquivDiameter    Extent  Solidity  rou
ndness  \
0         0.411785       29172     191.272751  0.783968  0.984986   0.
887034
1         0.333680       30417     195.896503  0.773098  0.990893   0.
984877
2         0.520401       30600     196.347702  0.775688  0.989510   0.
943852
3         0.553642       31120     198.139012  0.783683  0.990810   0.
970278
4         0.446622       31458     198.963039  0.786377  0.988334   0.
958173
...            ...         ...            ...       ...       ...
...
2717      0.699534       42250     231.155296  0.751998  0.993278   0.
947303
2718      0.747835       42477     231.235150  0.732514  0.988653   0.
899951
2719      0.731248       42419     231.270938  0.711710  0.990311   0.
915248
2720      0.786575       42547     231.270938  0.692230  0.987332   0.
883091
2721      0.734065       42569     231.631261  0.729932  0.989899   0.
918424

      Compactness  ShapeFactor1  ShapeFactor2  ShapeFactor3  ShapeFact
or4  \
0        0.953861      0.006979      0.003564      0.909851       0.998
430
1        0.970516      0.006697      0.003665      0.941900       0.999
166
2        0.923726      0.007020      0.003153      0.853270       0.999
236
3        0.912125      0.007045      0.003008      0.831973       0.999
061
4        0.945254      0.006770      0.003334      0.893506       0.998
640
...           ...           ...           ...           ...
...
2717     0.845148      0.006517      0.002051      0.714275       0.999
546
2718     0.813999      0.006764      0.001832      0.662594       0.998
```

```
055
2719    0.824987    0.006673    0.001907    0.680604    0.997
790
2720    0.785321    0.007010    0.001645    0.616729    0.998
760
2721    0.822730    0.006681    0.001888    0.676884    0.996
767

      Class
0         5
1         5
2         5
3         5
4         5
...     ...
2717      3
2718      3
2719      3
2720      3
2721      3

[2722 rows x 17 columns]
```

In [104]:

```
data_splitter_test_child=data_splitter(df_test)
x_test, y_test= data_splitter_test_child.test_x_y_split()
```

In [ ]:

```
classifier_model_child= Models(x_train, x_test, y_train, y_test)
print("===============================================================================")
print("Model 1 : Trivial Solution - probability based classifier: ")
accuray = classifier_model_child.trivial_solution()
print("===============================================================================")

print("===============================================================================")
print("Model 2 : Baseline Solution - Nearest means classifier: ")
model, predictions = classifier_model_child.nearest_means()
print("===============================================================================")

print("===============================================================================")
print("Model 3 : Neural network classifier: ")
model, predictions, history = classifier_model_child.neural_network()
print("===============================================================================")

print("===============================================================================")
print("Model 4 : Gradient Boosting Classifier: ")
gbc,predictions= classifier_model_child.gradient_boosting_classifier()
print("===============================================================================")

print("===============================================================================")
print("Model 5 : XGB Classifier: ")
xgb, predictions= classifier_model_child.xgb_classifier()
print("===============================================================================")
```

```
================================================================================
Model 1 : Trivial Solution - probability based classifier:
Trivial Solution Accuracy: 0.1473181484202792
Trivial Solution Macro-averaged F1 Score: 0.13955255173336373
Trivial Solution Micro-averaged F1 Score: 0.1473181484202792
Trivial Solution Confusion Matrix:
[[ 37  33  41  37  36  39  31]
 [ 14  19  10  14  14  20  17]
 [ 41  44  43  50  38  43  55]
 [106  89 114 108 105 110  99]
 [ 58  48  75  48  64  45  68]
 [ 61  53  51  52  55  64  60]
 [ 69  70  65  88  82  73  66]]
================================================================================
================================================================================
Model 2 : Baseline Solution - Nearest means classifier:
Nearest Means Classifier accuracy: 0.8692138133725202
Nearest Means Classifier Macro-averaged F1 Score: 0.8772917033116944
Nearest Means Classifier Micro-averaged F1 Score: 0.8692138133725202
Nearest Means Classifier Confusion Matrix:
[[168   0  49   0  19   0  18]
 [  0 108   0   0   0   0   0]
 [  8   0 286   0  18   0   2]
 [  0   0   0 661   2   3  65]
 [  1   0   7   2 391   0   5]
 [  0   0   0  26   1 327  42]
 [  0   0   0  54  32   2 425]]
Nearest Means Classifier cross-validation scores: [0.88740882 0.892483
35 0.88233428 0.89406914 0.88864213]
Nearest Means Classifier average cross-validation score: 0.88898754469
59709
================================================================================
================================================================================
Model 3 : Neural network classifier:
Epoch 1/50
444/444 [==============================] - 3s 4ms/step - loss: 1.1008
- accuracy: 0.6515 - val_loss: 0.5403 - val_accuracy: 0.8542
Epoch 2/50
444/444 [==============================] - 2s 3ms/step - loss: 0.3864
- accuracy: 0.8826 - val_loss: 0.3304 - val_accuracy: 0.8852
Epoch 3/50
444/444 [==============================] - 1s 3ms/step - loss: 0.2866
- accuracy: 0.9006 - val_loss: 0.2864 - val_accuracy: 0.8960
Epoch 4/50
444/444 [==============================] - 1s 2ms/step - loss: 0.2536
- accuracy: 0.9106 - val_loss: 0.2655 - val_accuracy: 0.9188
Epoch 5/50
444/444 [==============================] - 1s 2ms/step - loss: 0.2362
- accuracy: 0.9173 - val_loss: 0.2431 - val_accuracy: 0.9131
Epoch 6/50
444/444 [==============================] - 1s 3ms/step - loss: 0.2248
- accuracy: 0.9206 - val_loss: 0.2323 - val_accuracy: 0.9264
```

```
Epoch 7/50
444/444 [==============================] - 1s 3ms/step - loss: 0.2190
- accuracy: 0.9199 - val_loss: 0.2237 - val_accuracy: 0.9252
Epoch 8/50
444/444 [==============================] - 1s 3ms/step - loss: 0.2137
- accuracy: 0.9239 - val_loss: 0.2333 - val_accuracy: 0.9169
Epoch 9/50
444/444 [==============================] - 1s 3ms/step - loss: 0.2097
- accuracy: 0.9234 - val_loss: 0.2179 - val_accuracy: 0.9233
Epoch 10/50
444/444 [==============================] - 1s 2ms/step - loss: 0.2052
- accuracy: 0.9256 - val_loss: 0.2365 - val_accuracy: 0.9207
Epoch 11/50
444/444 [==============================] - 1s 2ms/step - loss: 0.2026
- accuracy: 0.9271 - val_loss: 0.2123 - val_accuracy: 0.9315
Epoch 12/50
444/444 [==============================] - 2s 3ms/step - loss: 0.2033
- accuracy: 0.9265 - val_loss: 0.2084 - val_accuracy: 0.9290
Epoch 13/50
444/444 [==============================] - 2s 4ms/step - loss: 0.2027
- accuracy: 0.9258 - val_loss: 0.2070 - val_accuracy: 0.9290
Epoch 14/50
444/444 [==============================] - 2s 3ms/step - loss: 0.1957
- accuracy: 0.9288 - val_loss: 0.2168 - val_accuracy: 0.9252
Epoch 15/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1958
- accuracy: 0.9280 - val_loss: 0.2132 - val_accuracy: 0.9233
Epoch 16/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1947
- accuracy: 0.9297 - val_loss: 0.2143 - val_accuracy: 0.9296
Epoch 17/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1938
- accuracy: 0.9302 - val_loss: 0.2005 - val_accuracy: 0.9302
Epoch 18/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1936
- accuracy: 0.9286 - val_loss: 0.1992 - val_accuracy: 0.9328
Epoch 19/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1910
- accuracy: 0.9306 - val_loss: 0.2001 - val_accuracy: 0.9296
Epoch 20/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1897
- accuracy: 0.9298 - val_loss: 0.1974 - val_accuracy: 0.9321
Epoch 21/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1917
- accuracy: 0.9291 - val_loss: 0.1974 - val_accuracy: 0.9302
Epoch 22/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1887
- accuracy: 0.9301 - val_loss: 0.2010 - val_accuracy: 0.9302
Epoch 23/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1878
- accuracy: 0.9315 - val_loss: 0.1937 - val_accuracy: 0.9341
Epoch 24/50
444/444 [==============================] - 2s 4ms/step - loss: 0.1901
- accuracy: 0.9302 - val_loss: 0.1944 - val_accuracy: 0.9321
Epoch 25/50
444/444 [==============================] - 2s 4ms/step - loss: 0.1863
- accuracy: 0.9332 - val_loss: 0.1939 - val_accuracy: 0.9315
```

```
Epoch 26/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1855
- accuracy: 0.9328 - val_loss: 0.1941 - val_accuracy: 0.9302
Epoch 27/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1846
- accuracy: 0.9333 - val_loss: 0.1954 - val_accuracy: 0.9321
Epoch 28/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1853
- accuracy: 0.9325 - val_loss: 0.1933 - val_accuracy: 0.9315
Epoch 29/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1841
- accuracy: 0.9337 - val_loss: 0.1934 - val_accuracy: 0.9328
Epoch 30/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1839
- accuracy: 0.9319 - val_loss: 0.2086 - val_accuracy: 0.9283
Epoch 31/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1842
- accuracy: 0.9327 - val_loss: 0.1909 - val_accuracy: 0.9321
Epoch 32/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1833
- accuracy: 0.9320 - val_loss: 0.1983 - val_accuracy: 0.9290
Epoch 33/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1830
- accuracy: 0.9333 - val_loss: 0.1885 - val_accuracy: 0.9347
Epoch 34/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1838
- accuracy: 0.9317 - val_loss: 0.2013 - val_accuracy: 0.9252
Epoch 35/50
444/444 [==============================] - 2s 4ms/step - loss: 0.1821
- accuracy: 0.9330 - val_loss: 0.1881 - val_accuracy: 0.9347
Epoch 36/50
444/444 [==============================] - 2s 4ms/step - loss: 0.1823
- accuracy: 0.9335 - val_loss: 0.1896 - val_accuracy: 0.9334
Epoch 37/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1808
- accuracy: 0.9343 - val_loss: 0.1865 - val_accuracy: 0.9353
Epoch 38/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1795
- accuracy: 0.9338 - val_loss: 0.1903 - val_accuracy: 0.9321
Epoch 39/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1807
- accuracy: 0.9330 - val_loss: 0.1961 - val_accuracy: 0.9321
Epoch 40/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1812
- accuracy: 0.9341 - val_loss: 0.1907 - val_accuracy: 0.9321
Epoch 41/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1800
- accuracy: 0.9351 - val_loss: 0.1879 - val_accuracy: 0.9302
Epoch 42/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1811
- accuracy: 0.9347 - val_loss: 0.1876 - val_accuracy: 0.9302
Epoch 43/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1808
- accuracy: 0.9344 - val_loss: 0.1957 - val_accuracy: 0.9239
Epoch 44/50
444/444 [==============================] - 1s 3ms/step - loss: 0.1794
- accuracy: 0.9355 - val_loss: 0.1913 - val_accuracy: 0.9334
```

```
Epoch 45/50
444/444 [==============================] - 1s 2ms/step - loss: 0.1779
- accuracy: 0.9347 - val_loss: 0.1986 - val_accuracy: 0.9290
Epoch 46/50
444/444 [==============================] - 2s 4ms/step - loss: 0.1802
- accuracy: 0.9335 - val_loss: 0.1986 - val_accuracy: 0.9258
Epoch 47/50
444/444 [==============================] - 2s 4ms/step - loss: 0.1792
- accuracy: 0.9331 - val_loss: 0.1877 - val_accuracy: 0.9321
Epoch 48/50
444/444 [==============================] - 2s 4ms/step - loss: 0.1784
- accuracy: 0.9356 - val_loss: 0.1853 - val_accuracy: 0.9347
Epoch 49/50
444/444 [==============================] - 2s 4ms/step - loss: 0.1783
- accuracy: 0.9341 - val_loss: 0.1958 - val_accuracy: 0.9277
Epoch 50/50
444/444 [==============================] - 2s 4ms/step - loss: 0.1780
- accuracy: 0.9345 - val_loss: 0.1854 - val_accuracy: 0.9341
86/86 [==============================] - 0s 2ms/step - loss: 0.3734 -
accuracy: 0.8692
Test loss: 0.37342727184295654, Test accuracy: 0.8692138195037842
86/86 [==============================] - 0s 2ms/step
Neural Network Classifier Macro-averaged F1 Score: 0.8787727797994064
Neural Network Classifier Micro-averaged F1 Score: 0.8692138133725202
Neural Network Classifier Confusion Matrix:
[[235   0   7   0   3   2   7]
 [  0 108   0   0   0   0   0]
 [ 54   0 228   0  28   0   4]
 [  0   0   0 705  10   2  14]
 [  3   0   0   2 399   0   2]
 [  0   0   0  69   0 304  23]
 [  4   0   0  83  36   3 387]]
Epoch 1/10
493/493 [==============================] - 2s 2ms/step - loss: 1.0169
- accuracy: 0.6911
Epoch 2/10
493/493 [==============================] - 1s 2ms/step - loss: 0.3586
- accuracy: 0.8759
Epoch 3/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2883
- accuracy: 0.8929
Epoch 4/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2616
- accuracy: 0.9039
Epoch 5/10
493/493 [==============================] - 2s 3ms/step - loss: 0.2439
- accuracy: 0.9141
Epoch 6/10
493/493 [==============================] - 2s 3ms/step - loss: 0.2298
- accuracy: 0.9187
Epoch 7/10
493/493 [==============================] - 1s 3ms/step - loss: 0.2214
- accuracy: 0.9215
Epoch 8/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2138
- accuracy: 0.9243
Epoch 9/10
```

```
493/493 [==============================] - 1s 2ms/step - loss: 0.2092
- accuracy: 0.9262
Epoch 10/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2057
- accuracy: 0.9264
86/86 [==============================] - 0s 2ms/step - loss: 0.3325 -
accuracy: 0.8755
Epoch 1/10
493/493 [==============================] - 2s 2ms/step - loss: 1.0189
- accuracy: 0.6907
Epoch 2/10
493/493 [==============================] - 1s 2ms/step - loss: 0.3837
- accuracy: 0.8820
Epoch 3/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2947
- accuracy: 0.8976
Epoch 4/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2685
- accuracy: 0.9037
Epoch 5/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2494
- accuracy: 0.9126
Epoch 6/10
493/493 [==============================] - 2s 3ms/step - loss: 0.2372
- accuracy: 0.9175
Epoch 7/10
493/493 [==============================] - 2s 3ms/step - loss: 0.2269
- accuracy: 0.9210
Epoch 8/10
493/493 [==============================] - 1s 3ms/step - loss: 0.2203
- accuracy: 0.9213
Epoch 9/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2137
- accuracy: 0.9248
Epoch 10/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2105
- accuracy: 0.9271
86/86 [==============================] - 0s 2ms/step - loss: 0.4079 -
accuracy: 0.8516
Epoch 1/10
493/493 [==============================] - 2s 2ms/step - loss: 1.0422
- accuracy: 0.7161
Epoch 2/10
493/493 [==============================] - 1s 2ms/step - loss: 0.3536
- accuracy: 0.8917
Epoch 3/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2659
- accuracy: 0.9121
Epoch 4/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2348
- accuracy: 0.9218
Epoch 5/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2208
- accuracy: 0.9232
Epoch 6/10
493/493 [==============================] - 2s 4ms/step - loss: 0.2147
- accuracy: 0.9238
```

```
Epoch 7/10
493/493 [==============================] - 2s 4ms/step - loss: 0.2073
- accuracy: 0.9281
Epoch 8/10
493/493 [==============================] - 1s 3ms/step - loss: 0.2033
- accuracy: 0.9290
Epoch 9/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2021
- accuracy: 0.9279
Epoch 10/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2006
- accuracy: 0.9271
86/86 [==============================] - 0s 2ms/step - loss: 0.3529 -
accuracy: 0.8696
Epoch 1/10
493/493 [==============================] - 2s 2ms/step - loss: 1.0881
- accuracy: 0.6115
Epoch 2/10
493/493 [==============================] - 1s 2ms/step - loss: 0.4029
- accuracy: 0.8743
Epoch 3/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2903
- accuracy: 0.8974
Epoch 4/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2559
- accuracy: 0.9080
Epoch 5/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2388
- accuracy: 0.9133
Epoch 6/10
493/493 [==============================] - 1s 3ms/step - loss: 0.2266
- accuracy: 0.9205
Epoch 7/10
493/493 [==============================] - 2s 3ms/step - loss: 0.2164
- accuracy: 0.9230
Epoch 8/10
493/493 [==============================] - 2s 4ms/step - loss: 0.2097
- accuracy: 0.9234
Epoch 9/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2072
- accuracy: 0.9245
Epoch 10/10
493/493 [==============================] - 1s 2ms/step - loss: 0.2029
- accuracy: 0.9263
```

In [ ]: