1. GROUP MEMBERS(Name AND Roll number)

Ashutosh(20/49052)

Monu Kumar(20/49022)

Shambhu(20/49073)

2. MOBILE PRICE CLASSIFICATION

> **SOURCE**:-https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification?select=test.csv

> **DESCRIPTION**:- IN the given dataset **Mobile Price Classification** There are 20 attributes(Columns) and 1 classification attributes having 4 classes((low cost=0), (medium cost=1), (high cost=2) and (very high cost=4)) and 2000 records.

According to the attributes we predict the class of the mobile phone.

> Attributes(Columns):-
>
> 1. **battery_power**(Total energy a battery can store in one time measured in mAh).
> 2. **blue**(Has bluetooth or not).
> 3. **clock_speed**(speed at which microprocessor executes instructions)
> 4. **dual_sim**(Has dual sim support or not).
> 5. **fc**(Front Camera mega pixels).
> 6. **four_g**(Has 4G or not).
> 7. **int_memory**(Internal Memory in Gigabytes)
> 8. **m_dep**(Mobile Depth in cm).
> 9. **mobile_wt**(Weight of mobile phone).
> 10. **n_cores**(Number of cores of processor).
> 11. **pc**(Primary Camera mega pixels).
> 12. **px_height**(Pixel Resolution Height).
> 13. **px_width**(Pixel Resolution Width).
> 14. **ram**(Random Access Memory in Mega Bytes)
> 15. **sc_h**(Screen Height of mobile in cm).
> 16. **sc_w**(Screen Width of mobile in cm).
> 17. **talk_time**(longest time that a single battery charge will last when you are).
> 18. **three_g**(Has 3G or not).
> 19. **touch_screen**(Has touch screen or not).
> 20. **wifi**(Has wifi or not).
> 21. **price_range(This is the target variable with value of 0(low cost), 1(medium cost), 2(high cost) and 3(very high cost).)**

```
# LIBRARIES
# !pip install fabulous

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from fabulous import text as ft


import pprint as pprint
import seaborn as sns
from fabulous.color import *


#READ CSV FILE
df_Original=pd.read_csv("train.csv")
print(bold(yellow(df_Original)))
```

```
      battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  \
0               842     0          2.2         0   1       0           7
1              1021     1          0.5         1   0       1          53
2               563     1          0.5         1   2       1          41
3               615     1          2.5         0   0       0          10
4              1821     1          1.2         0  13       1          44
...             ...   ...          ...       ...  ..     ...         ...
1995            794     1          0.5         1   0       1           2
```

```
1996          1965     1          2.6         1   0       0          39
1997          1911     0          0.9         1   1       1          36
1998          1512     0          0.9         0   4       1          46
1999           510     1          2.0         1   5       1          45

      m_dep  mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w  \
0       0.6        188        2  ...         20       756  2549     9     7
1       0.7        136        3  ...        905      1988  2631    17     3
2       0.9        145        5  ...       1263      1716  2603    11     2
3       0.8        131        6  ...       1216      1786  2769    16     8
4       0.6        141        2  ...       1208      1212  1411     8     2
...     ...        ...      ...  ...        ...       ...   ...   ...   ...
1995    0.8        106        6  ...       1222      1890   668    13     4
1996    0.2        187        4  ...        915      1965  2032    11    10
1997    0.7        108        8  ...        868      1632  3057     9     1
1998    0.1        145        5  ...        336       670   869    18    10
1999    0.9        168        6  ...        483       754  3919    19     4

      talk_time  three_g  touch_screen  wifi  price_range
0            19        0             0     1            1
1             7        1             1     0            2
2             9        1             1     0            2
3            11        1             0     0            2
4            15        1             1     0            1
...         ...      ...           ...   ...          ...
1995         19        1             1     0            0
1996         16        1             1     1            2
1997          5        1             1     0            3
1998         19        1             1     1            0
1999          2        1             1     1            3

[2000 rows x 21 columns]


#Describe
print(bold(blue(df_Original.describe())))

       battery_power       blue  clock_speed     dual_sim           fc  \
count    2000.000000  2000.0000  2000.000000  2000.000000  2000.000000
mean     1238.518500     0.4950     1.522250     0.509500     4.309500
std       439.418206     0.5001     0.816004     0.500035     4.341444
min       501.000000     0.0000     0.500000     0.000000     0.000000
25%       851.750000     0.0000     0.700000     0.000000     1.000000
50%      1226.000000     0.0000     1.500000     1.000000     3.000000
75%      1615.250000     1.0000     2.200000     1.000000     7.000000
max      1998.000000     1.0000     3.000000     1.000000    19.000000

            four_g   int_memory        m_dep    mobile_wt      n_cores  ...  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000  ...
mean      0.521500    32.046500     0.501750   140.249000     4.520500  ...
std       0.499662    18.145715     0.288416    35.399655     2.287837  ...
min       0.000000     2.000000     0.100000    80.000000     1.000000  ...
25%       0.000000    16.000000     0.200000   109.000000     3.000000  ...
50%       1.000000    32.000000     0.500000   141.000000     4.000000  ...
75%       1.000000    48.000000     0.800000   170.000000     7.000000  ...
max       1.000000    64.000000     1.000000   200.000000     8.000000  ...

         px_height     px_width          ram         sc_h         sc_w  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000
mean    645.108000  1251.515500  2124.213000    12.306500     5.767000
std     443.780811   432.199447  1084.732044     4.213245     4.356398
min       0.000000   500.000000   256.000000     5.000000     0.000000
25%     282.750000   874.750000  1207.500000     9.000000     2.000000
50%     564.000000  1247.000000  2146.500000    12.000000     5.000000
75%     947.250000  1633.000000  3064.500000    16.000000     9.000000
max    1960.000000  1998.000000  3998.000000    19.000000    18.000000

         talk_time      three_g  touch_screen         wifi  price_range
count  2000.000000  2000.000000   2000.000000  2000.000000  2000.000000
mean     11.011000     0.761500      0.503000     0.507000     1.500000
std       5.463955     0.426273      0.500116     0.500076     1.118314
min       2.000000     0.000000      0.000000     0.000000     0.000000
25%       6.000000     1.000000      0.000000     0.000000     0.750000
50%      11.000000     1.000000      1.000000     1.000000     1.500000
75%      16.000000     1.000000      1.000000     1.000000     2.250000
max      20.000000     1.000000      1.000000     1.000000     3.000000

[8 rows x 21 columns]


#INFO
print(bold(magenta(df_Original.info())))

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  2000 non-null   int64
 1   blue           2000 non-null   int64
 2   clock_speed    2000 non-null   float64
```

```
 3   dual_sim       2000 non-null   int64
 4   fc             2000 non-null   int64
 5   four_g         2000 non-null   int64
 6   int_memory     2000 non-null   int64
 7   m_dep          2000 non-null   float64
 8   mobile_wt      2000 non-null   int64
 9   n_cores        2000 non-null   int64
 10  pc             2000 non-null   int64
 11  px_height      2000 non-null   int64
 12  px_width       2000 non-null   int64
 13  ram            2000 non-null   int64
 14  sc_h           2000 non-null   int64
 15  sc_w           2000 non-null   int64
 16  talk_time      2000 non-null   int64
 17  three_g        2000 non-null   int64
 18  touch_screen   2000 non-null   int64
 19  wifi           2000 non-null   int64
 20  price_range    2000 non-null   int64
dtypes: float64(2), int64(19)
memory usage: 328.2 KB
None
```

```
#HEAD
print(bold(df_Original.head()))
```

```
   battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  m_dep  \
0            842     0          2.2         0   1       0           7    0.6
1           1021     1          0.5         1   0       1          53    0.7
2            563     1          0.5         1   2       1          41    0.9
3            615     1          2.5         0   0       0          10    0.8
4           1821     1          1.2         0  13       1          44    0.6

   mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w  talk_time  \
0        188        2  ...         20       756  2549     9     7         19
1        136        3  ...        905      1988  2631    17     3          7
2        145        5  ...       1263      1716  2603    11     2          9
3        131        6  ...       1216      1786  2769    16     8         11
4        141        2  ...       1208      1212  1411     8     2         15

   three_g  touch_screen  wifi  price_range
0        0             0     1            1
1        1             1     0            2
2        1             1     0            2
3        1             0     0            2
4        1             1     0            1

[5 rows x 21 columns]
```

```
#TAIL
print(bold(df_Original.tail()))
```

```
      battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  \
1995            794     1          0.5         1   0       1           2
1996           1965     1          2.6         1   0       0          39
1997           1911     0          0.9         1   1       1          36
1998           1512     0          0.9         0   4       1          46
1999            510     1          2.0         1   5       1          45

      m_dep  mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w  \
1995    0.8        106        6  ...       1222      1890   668    13     4
1996    0.2        187        4  ...        915      1965  2032    11    10
1997    0.7        108        8  ...        868      1632  3057     9     1
1998    0.1        145        5  ...        336       670   869    18    10
1999    0.9        168        6  ...        483       754  3919    19     4

      talk_time  three_g  touch_screen  wifi  price_range
1995         19        1             1     0            0
1996         16        1             1     1            2
1997          5        1             1     0            3
1998         19        1             1     1            0
1999          2        1             1     1            3

[5 rows x 21 columns]
```

```
#INDEX | COLUMNS
print(bold(red(df_Original.columns)))
print(blue(df_Original.index))
```

```
Index(['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
       'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
       'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
       'touch_screen', 'wifi', 'price_range'],
      dtype='object')
RangeIndex(start=0, stop=2000, step=1)
```

```
#DIMENSION
print(bold(yellow("Dimension of Orignal data : ",np.ndim(df_Original))))
print(bold(red("Shape of Orignal data : ",np.shape(df_Original))))
print(bold(cyan("Type : ",type(df_Original))))
print(bold(magenta("DataTypes --->\n",df_Original.dtypes)))
```

```
Dimension of Orignal data : 2
Shape of Orignal data : (2000, 21)
Type : <class 'pandas.core.frame.DataFrame'>
DataTypes --->
battery_power      int64
blue               int64
clock_speed      float64
dual_sim           int64
fc                 int64
four_g             int64
int_memory         int64
m_dep            float64
mobile_wt          int64
n_cores            int64
pc                 int64
px_height          int64
px_width           int64
ram                int64
sc_h               int64
sc_w               int64
talk_time          int64
three_g            int64
touch_screen       int64
wifi               int64
price_range        int64
dtype: object
```

```
#Check NULL Values
print(bold(green(df_Original.isnull().sum())))
```

```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

3. Exploratory Data Analysis

> **1.1 Data Distribution frequecy plots. (HISTOGRAM)**

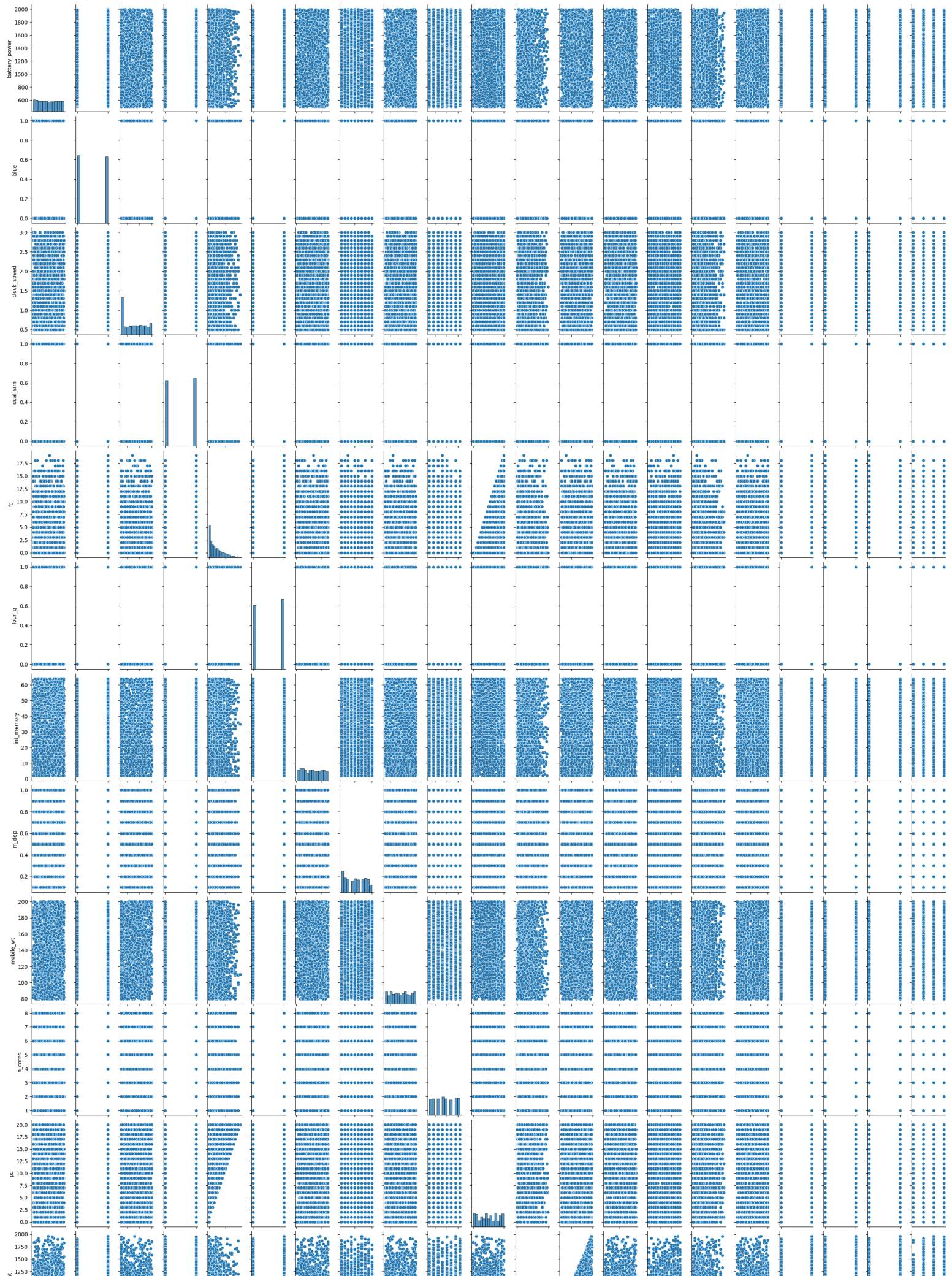
```
df_Original.hist(bins=30,figsize=(25, 15) )
```

```
array([[<Axes: title={'center': 'battery_power'}>,
        <Axes: title={'center': 'blue'}>,
        <Axes: title={'center': 'clock_speed'}>,
        <Axes: title={'center': 'dual_sim'}>,
        <Axes: title={'center': 'fc'}>],
       [<Axes: title={'center': 'four_g'}>,
        <Axes: title={'center': 'int_memory'}>,
        <Axes: title={'center': 'm_dep'}>,
        <Axes: title={'center': 'mobile_wt'}>,
        <Axes: title={'center': 'n_cores'}>],
       [<Axes: title={'center': 'pc'}>,
        <Axes: title={'center': 'px_height'}>,
        <Axes: title={'center': 'px_width'}>,
        <Axes: title={'center': 'ram'}>,
        <Axes: title={'center': 'sc_h'}>],
       [<Axes: title={'center': 'sc_w'}>,
        <Axes: title={'center': 'talk_time'}>,
        <Axes: title={'center': 'three_g'}>,
        <Axes: title={'center': 'touch_screen'}>,
        <Axes: title={'center': 'wifi'}>],
       [<Axes: title={'center': 'price_range'}>, <Axes: >, <Axes: >,
        <Axes: >, <Axes: >]], dtype=object)
```

1.2 Scatter plot

```
sns.pairplot(df_Original,height=3,aspect=0.4,)
```

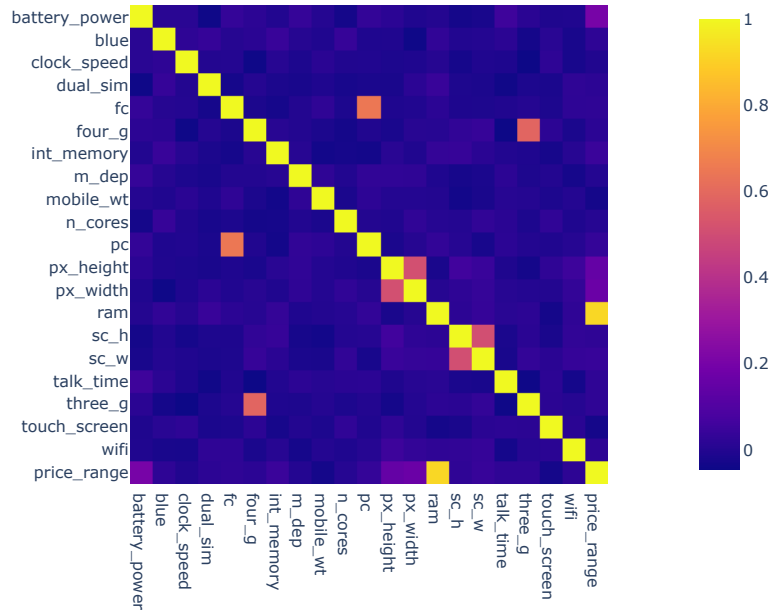
<seaborn.axisgrid.PairGrid at 0x7f1fcf8c6850>



1.3 Pearson correlation

```
# sns.heatmap(df_Original.corr(),cmap="RdYlBu")
```

```
fig = px.imshow(df_Original.corr())
fig.show()
```




```
print("Top 10 corelated Columns to the Tagated value : ")
abs(df_Original.corr()["price_range"]).nlargest(10)
```

```
Top 10 corelated Columns to the Tagated value :
price_range      1.000000
ram              0.917046
battery_power    0.200723
px_width         0.165818
px_height        0.148858
int_memory       0.044435
sc_w             0.038711
pc               0.033599
touch_screen     0.030411
mobile_wt        0.030302
Name: price_range, dtype: float64
```

```
#Attribute Classification
Independent_Attributes=df_Original[df_Original.corr()["price_range"].nlargest(10).index[1:]]
Dependent_Attribute=df_Original.iloc[:,-1]
```

```
print("Independent Attributes are :- /n")
print(Independent_Attributes.info())
```

```
Independent Attributes are :- /n
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   ram            2000 non-null   int64
 1   battery_power  2000 non-null   int64
 2   px_width       2000 non-null   int64
 3   px_height      2000 non-null   int64
 4   int_memory     2000 non-null   int64
 5   sc_w           2000 non-null   int64
 6   pc             2000 non-null   int64
 7   three_g        2000 non-null   int64
 8   sc_h           2000 non-null   int64
dtypes: int64(9)
memory usage: 140.8 KB
None
```

SCATTER PLOT OF INDEPENDENT ATTRIBUTES

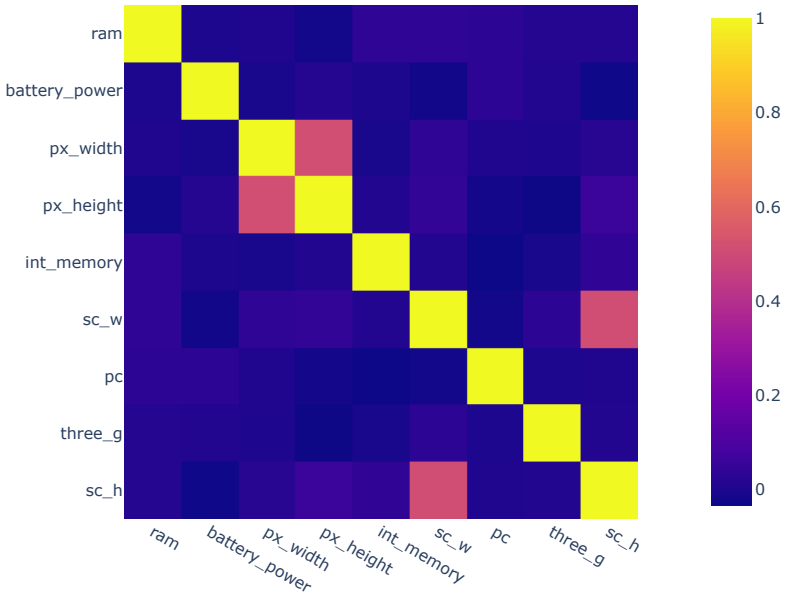
```
fig = px.scatter_matrix(Independent_Attributes, dimensions=(Independent_Attributes), height=1500, width=2500)

fig.update_traces(diagonal_visible=False)

fig.show()
```

4000

PEARSON CORELATION OF INDEPENDENT ATTRIBUTES:-

2000

```
fig = px.imshow(Independent_Attributes.corr())
fig.show()
```




```
print("Dependent Attribute is :- \n")
print(Dependent_Attribute.info())
```

```
Dependent Attribute is :-

<class 'pandas.core.series.Series'>
RangeIndex: 2000 entries, 0 to 1999
Series name: price_range
Non-Null Count  Dtype
--------------  -----
2000 non-null   int64
dtypes: int64(1)
memory usage: 15.8 KB
None
```

```
Dependent_Attribute
```

```
0       1
1       2
2       2
3       2
4       1
      ..
1995    0
1996    2
1997    3
1998    0
1999    3
Name: price_range, Length: 2000, dtype: int64
```

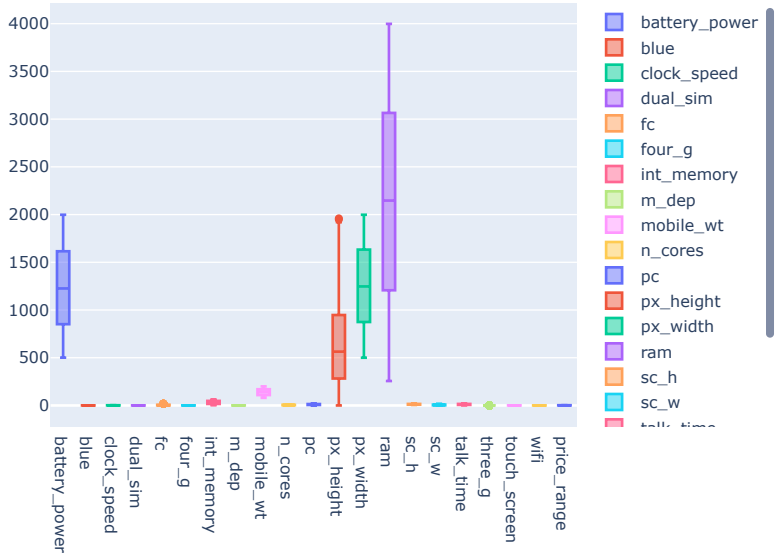
2.1 Removing Duplicate values

```
Duplicates_value=df_Original[Independent_Attributes.duplicated()]
Duplicates_value
```

battery_power blue clock_speed dual_sim fc four_g int_memory m_dep mobile_wt n_cores px_height px_width ram sc_h sc_w talk_time th

2.2 Handling Outliers

```
fig = go.Figure()

for col in df_Original:
  fig.add_trace(go.Box(y=df_Original[col].values, name=df_Original[col].name))

fig.show()
```



2.3 Handling Null values (K-nearest neighbour method)

```
print(df_Original.isna().sum(),bold(yellow("\n\n\n NO NULL VALUE PRESENT :-")))
```

```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64


 NO NULL VALUE PRESENT :-
```

3. Feature Scaling

3.1 STANDARD SCALER

```
from sklearn.preprocessing import StandardScaler
def normalize(Independent_Attributes):
        print("Mean and Standard Deviation Before")
        print(Independent_Attributes.mean(axis=0), Independent_Attributes.std(axis=0))
```

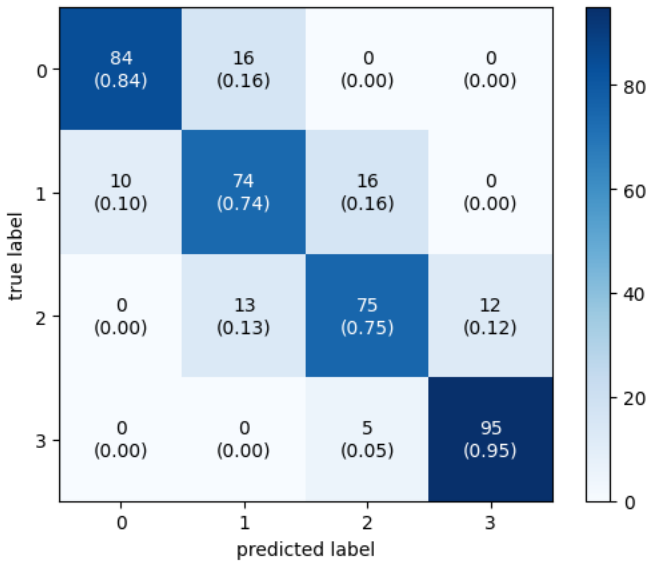
```
        sc=StandardScaler()
        XScaled = sc.fit_transform(Independent_Attributes)

        print("Mean and Standard Deviation After")
        print(XScaled.mean(axis=0).round(4), XScaled.std(axis=0))
        return XScaled

# Standard=normalize(Independent_Attributes)
```

Logistic Regression

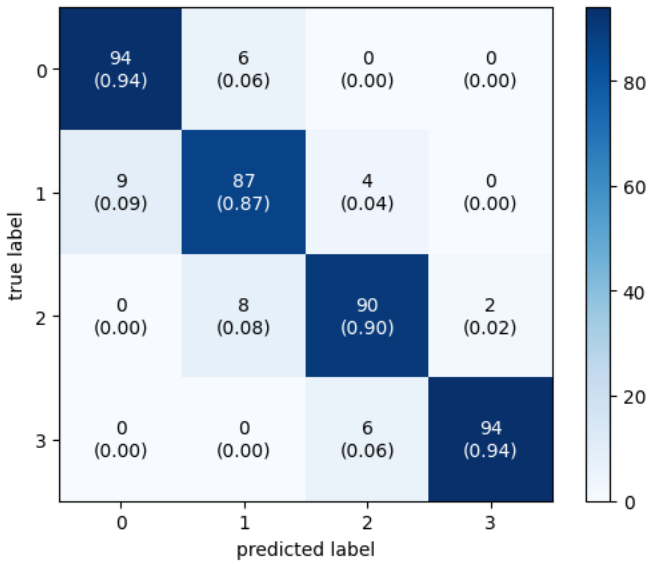
```
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.model_selection import StratifiedKFold
from sklearn .metrics import roc_auc_score
def Logistic_Regression(Independent_Attributes,Dependent_Attribute):
  X=Independent_Attributes.values
  Y=Dependent_Attribute.values
  print("*************Normalization/Standardization*************")
  XScaled = normalize(X)

  skf=StratifiedKFold(n_splits=5, shuffle=True, random_state=2)
  acc=[]

  y_ori = np.array([], dtype=int)

  y_pre= np.array([], dtype=int)
  net_mat=np.zeros((4, 4))
  roc=[]
  for train_index, test_index in skf.split(X,Y):

      X_train=X[train_index]
      X_test=X[test_index]
      Y_train=Y[train_index]
      Y_test=Y[test_index]
      print(bold(green("Shape of X_train:-   {}   ||   X_test:-   {}   || Y_train:-  {}  ||  Y_test:-   {}  ".format(X_train.shape,X_test.shape,Y_train.shap

      print(bold(magenta("Train:-   {}   ||  Test:-    {} ".format(np.bincount(Y[train_index]),np.bincount(Y[test_index])))))

      LRModel = LogisticRegression(max_iter=100000)
      LRModel.fit(X_train, Y_train)
      Y_testPred = LRModel.predict(X_test)

      y_ori=np.hstack((y_ori,Y_test))


      y_pre=np.hstack((y_pre,Y_testPred))
      testAccuracy = metrics.accuracy_score(Y_test, Y_testPred)
      print(bold(red("Test Accuracy", testAccuracy*100)))
      acc.append(testAccuracy)
      roc_score=roc_auc_score(Y_test, LRModel.predict_proba(X_test), multi_class='ovr')

      print(bold(yellow("ROC_AUC_SCORE: ",roc_score)))


      matrix1= confusion_matrix(Y_test, Y_testPred)
      #      sum of the total confusion matirx
      net_mat=net_mat+matrix1

      plot_confusion_matrix(matrix1,show_normed=True, colorbar=True, show_absolute=True)

      plt.show()
  print(bold(green(" Result Of Logsitic Regression: \n")))
  avg_accuracy=(sum(acc) / len(acc))*100
  print(bold(yellow_bg("average accuracy:  ",avg_accuracy )))

  net_mat = net_mat.astype('int')
  print(bold(cyan(net_mat)))
  plot_confusion_matrix(net_mat,show_normed=True, colorbar=True, show_absolute=True, cmap='Blues')

  plt.show()

  roc_score=roc_auc_score(Y, LRModel.predict_proba(X), multi_class='ovr')
```

```
  print(bold(yellow_bg("ROC_AUC_SCORE: ",roc_score)))



  print("Classification Report:\n")
  report=classification_report(y_ori, y_pre,output_dict=True)
  report_Df=pd.DataFrame(report)
  print(report_Df)
  sns.heatmap(report_Df.T,annot=True)
  return avg_accuracy
avg_accuracy_logistic=Logistic_Regression(Independent_Attributes,Dependent_Attribute)
```

```
************Normalization/Standardization************
Mean and Standard Deviation Before
[2.1242130e+03 1.2385185e+03 1.2515155e+03 6.4510800e+02 3.2046500e+01
 5.7670000e+00 9.9165000e+00 7.6150000e-01 1.2306500e+01] [1.08446083e+03 4.39308338e+02 4.32091384e+02
 1.81411780e+01 4.35530837e+00 6.06279867e+00 4.26166341e-01
 4.21219156e+00]
Mean and Standard Deviation After
[-0.  0.  0.  0. -0. -0.  0.  0.  0.] [1. 1. 1. 1. 1. 1. 1. 1. 1.]
Shape of X_train:-   (1600, 9)   ||   X_test:-   (400, 9)   || Y_train:-  (1600,)  ||  Y_test:-   (400,
Train:-   [400 400 400 400]   ||  Test:-    [100 100 100 100]
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of f AND g EVALUATIONS EXCEEDS LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

Test Accuracy82.0
ROC_AUC_SCORE: 0.9589083333333334
```




```
Shape of X_train:-   (1600, 9)   ||   X_test:-   (400, 9)   || Y_train:-  (1600,)  ||  Y_test:-   (400,
Train:-   [400 400 400 400]   ||  Test:-    [100 100 100 100]
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of f AND g EVALUATIONS EXCEEDS LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

Test Accuracy91.25
ROC_AUC_SCORE: 0.9921333333333333
```




```
Shape of X_train:-   (1600, 9)   ||   X_test:-   (400, 9)   || Y_train:-  (1600,)  ||  Y_test:-   (400,
Train:-   [400 400 400 400]   ||  Test:-    [100 100 100 100]
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning:

lbfgs failed to converge (status=1):
```

```
    STOP: TOTAL NO. of f AND g EVALUATIONS EXCEEDS LIMIT.
```

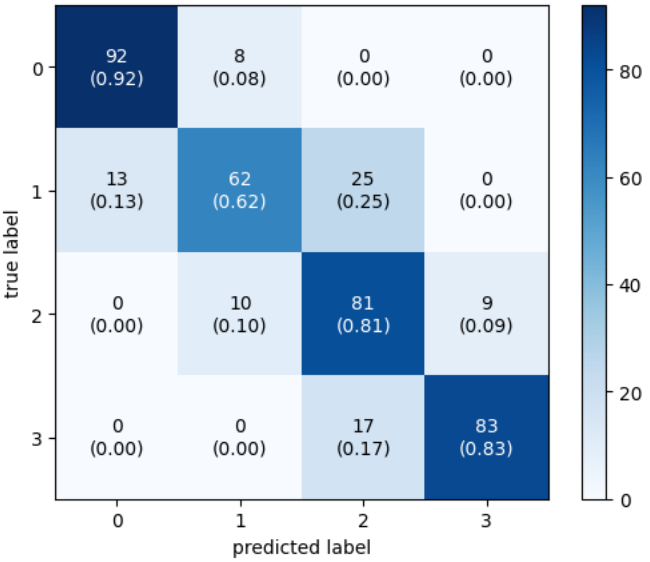
Descision Tree Classifier

```
    Please also refer to the documentation for alternative solver options:
```

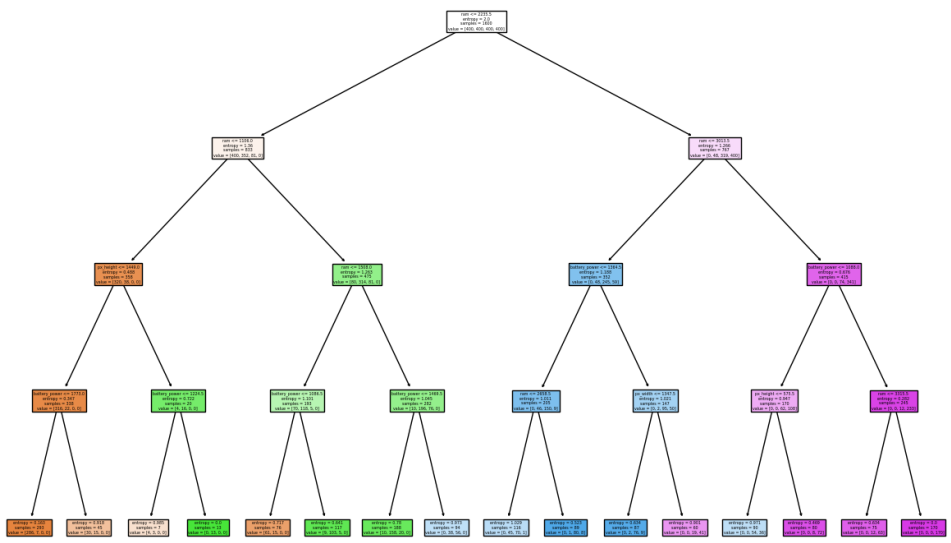
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus
def Decision_Tree(Independent_Attributes,Dependent_Attribute):
  X=Independent_Attributes.values
  Y=Dependent_Attribute.values
  print("*************Normalization/Standardization*************")
  XScaled = normalize(X)

  skf=StratifiedKFold(n_splits=5, shuffle=True, random_state=2)
  acc=[]

  y_ori = np.array([], dtype=int)

  y_pre= np.array([], dtype=int)
  net_mat=np.zeros((4, 4))

  for train_index, test_index in skf.split(X,Y):

      X_train=X[train_index]
      X_test=X[test_index]
      Y_train=Y[train_index]
      Y_test=Y[test_index]
      print(bold(green("Shape of X_train:-   {}   ||   X_test:-   {}   || Y_train:-  {}  ||  Y_test:-   {}  ".format(X_train.shape,X_test.shape,Y_train.shap

      print(bold(magenta("Train:-   {}   ||  Test:-    {} ".format(np.bincount(Y[train_index]),np.bincount(Y[test_index])))))
      treemodel=DecisionTreeClassifier(criterion='entropy',max_depth=4)
      treemodel.fit(X_train, Y_train)

      Y_testPred =treemodel.predict(X_test)

      y_ori=np.hstack((y_ori,Y_test))


      y_pre=np.hstack((y_pre,Y_testPred))
      testAccuracy = metrics.accuracy_score(Y_test, Y_testPred)
      print(bold(red("Test Accuracy", testAccuracy*100)))
      acc.append(testAccuracy)

      roc_score=roc_auc_score(Y_test, treemodel.predict_proba(X_test), multi_class='ovr')

      print(bold(yellow("ROC_AUC_SCORE: ",roc_score)))


      matrix1= confusion_matrix(Y_test, Y_testPred)
      #      sum of the total confusion matirx
      net_mat=net_mat+matrix1


      plot_confusion_matrix(matrix1,show_normed=True, colorbar=True, show_absolute=True)

      plt.show()




      plt.figure(figsize=(15,10))
      tree.plot_tree(treemodel,feature_names=Independent_Attributes.columns,filled=True)


  print(bold(green(" Result Of  Decisiom Tree: \n")))
  avg_accuracy=(sum(acc) / len(acc))*100
  print(bold(yellow_bg("average accuracy:  ",avg_accuracy )))

  net_mat = net_mat.astype('int')
  print(bold(cyan(net_mat)))
  plot_confusion_matrix(net_mat,show_normed=True, colorbar=True, show_absolute=True, cmap='Blues')

  plt.show()

  roc_score=roc_auc_score(Y, treemodel.predict_proba(X), multi_class='ovr')
  print(bold(yellow_bg("ROC_AUC_SCORE: ",roc_score)))
```

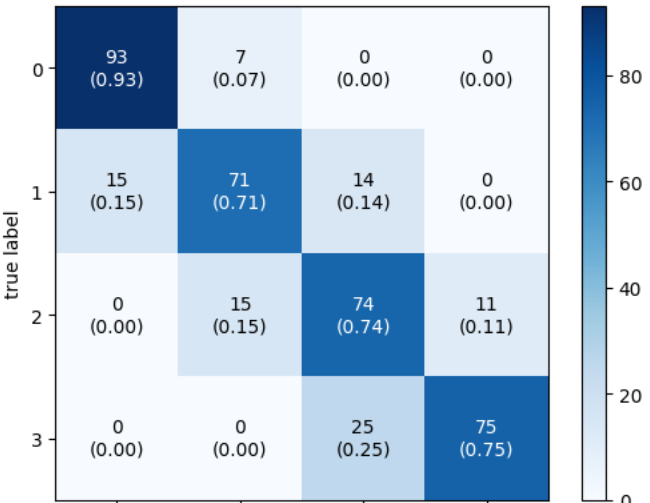
```
  print(bold("Classification Report:\n"))
  report=classification_report(y_ori, y_pre,output_dict=True)
  report_Df=pd.DataFrame(report)
  print(bold(magenta(report_Df)))
  sns.heatmap(report_Df.T,annot=True)
  return avg_accuracy


avg_accuracy_DCT=Decision_Tree(Independent_Attributes,Dependent_Attribute)
```

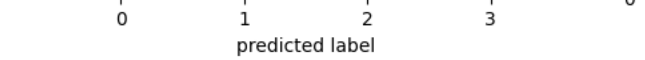
```
************Normalization/Standardization************
Mean and Standard Deviation Before
[2.1242130e+03 1.2385185e+03 1.2515155e+03 6.4510800e+02 3.2046500e+01
 5.7670000e+00 9.9165000e+00 7.6150000e-01 1.2306500e+01] [1.08446083e+03 4.39308338e+02 4.3209
 1.81411780e+01 4.35530837e+00 6.06279867e+00 4.26166341e-01
 4.21219156e+00]
Mean and Standard Deviation After
[-0.  0.  0.  0. -0. -0.  0.  0.  0.] [1. 1. 1. 1. 1. 1. 1. 1. 1.]
Shape of X_train:-   (1600, 9)   ||   X_test:-   (400, 9)   || Y_train:-  (1600,)  ||  Y_test:-
Train:-   [400 400 400 400]   ||  Test:-    [100 100 100 100]
Test Accuracy79.5
ROC_AUC_SCORE: 0.9398625
```



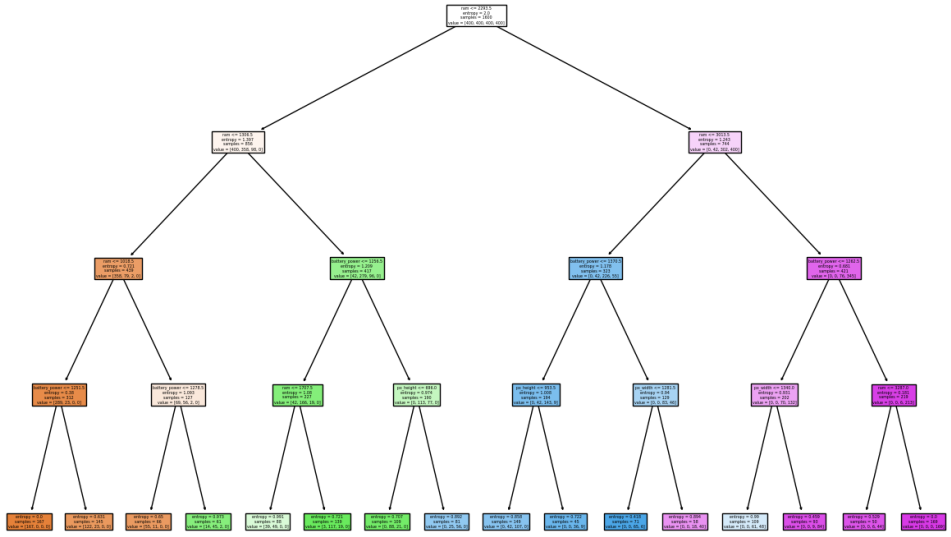

```
Shape of X_train:-   (1600, 9)   ||   X_test:-   (400, 9)   || Y_train:-  (1600,)  ||  Y_test:-
Train:-   [400 400 400 400]   ||  Test:-    [100 100 100 100]
Test Accuracy78.25
ROC_AUC_SCORE: 0.9421708333333332
```

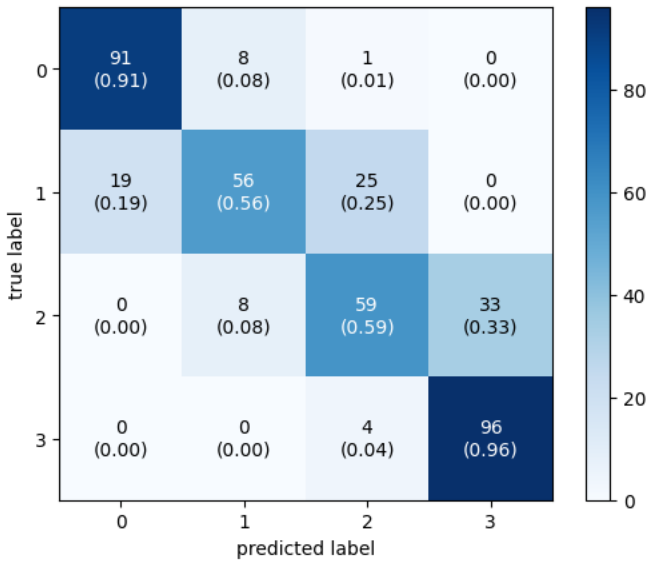




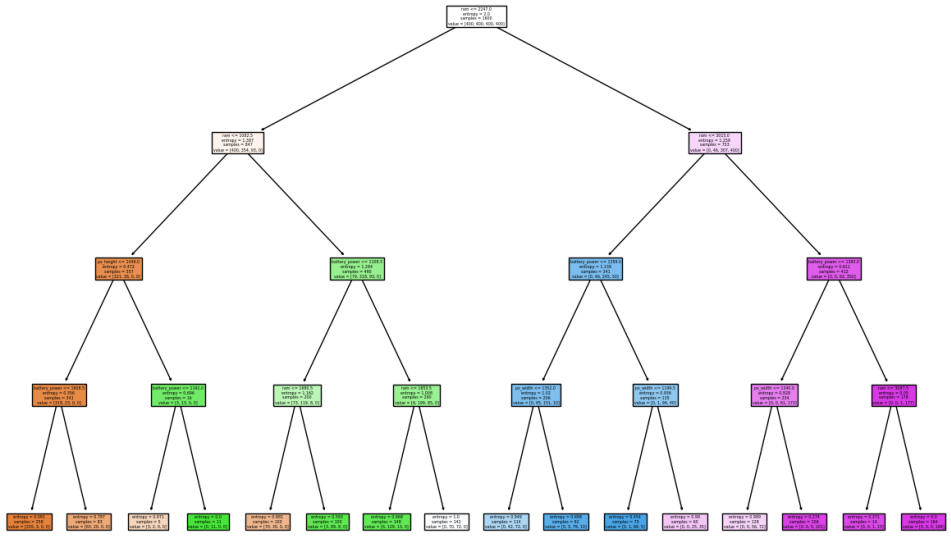


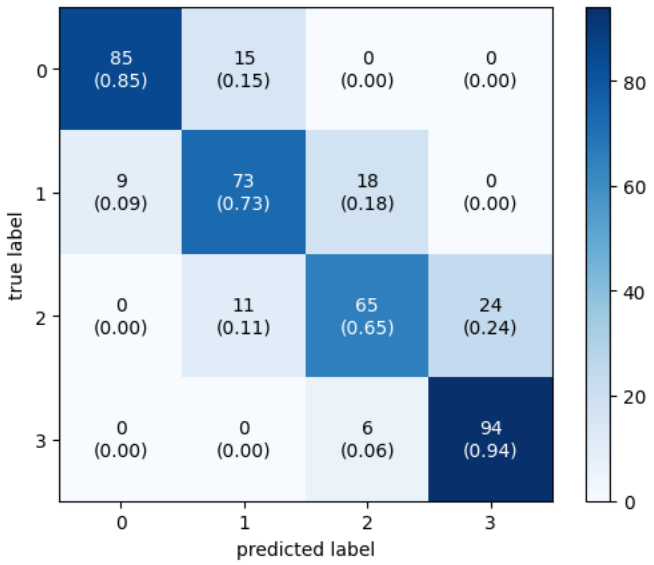
Shape of X_train:- (1600, 9) || X_test:- (400, 9) || Y_train:- (1600,) || Y_test:-
Train:- [400 400 400 400] || Test:- [100 100 100 100]
Test Accuracy75.5
ROC_AUC_SCORE: 0.9311666666666666



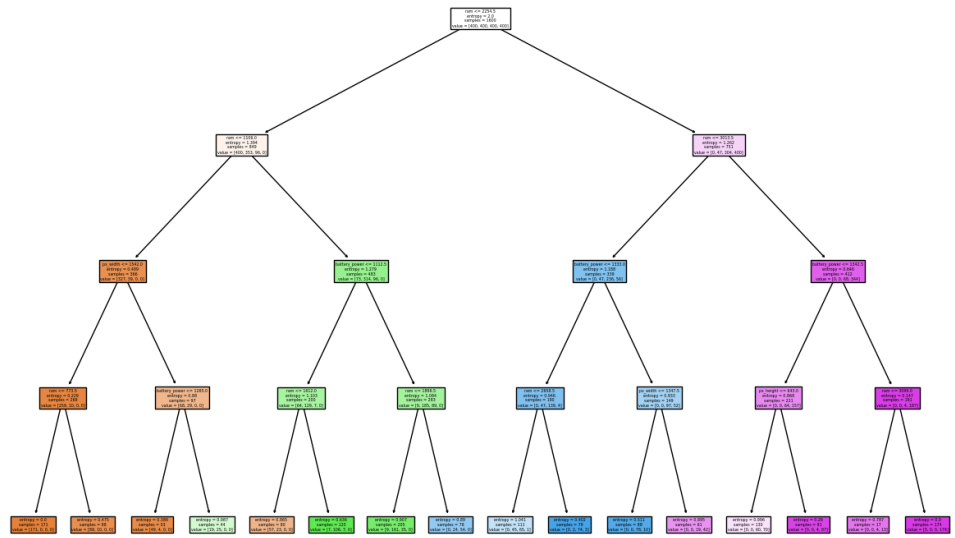


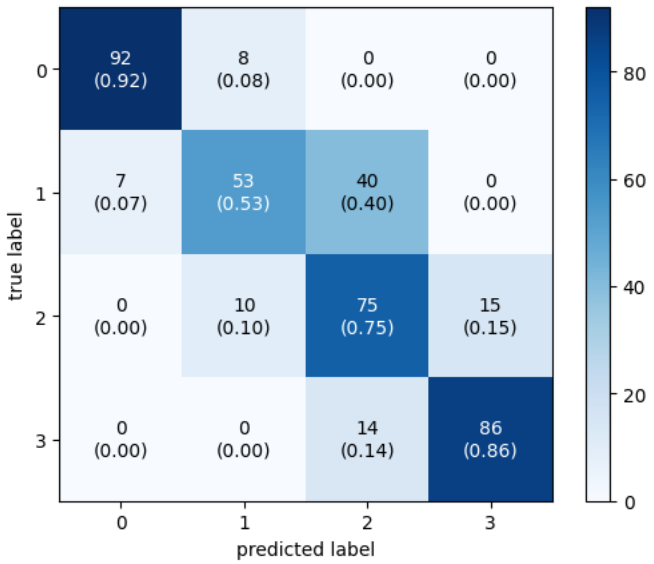
Shape of X_train:- (1600, 9) || X_test:- (400, 9) || Y_train:- (1600,) || Y_test:-
Train:- [400 400 400 400] || Test:- [100 100 100 100]
Test Accuracy79.25
ROC_AUC_SCORE: 0.9421375



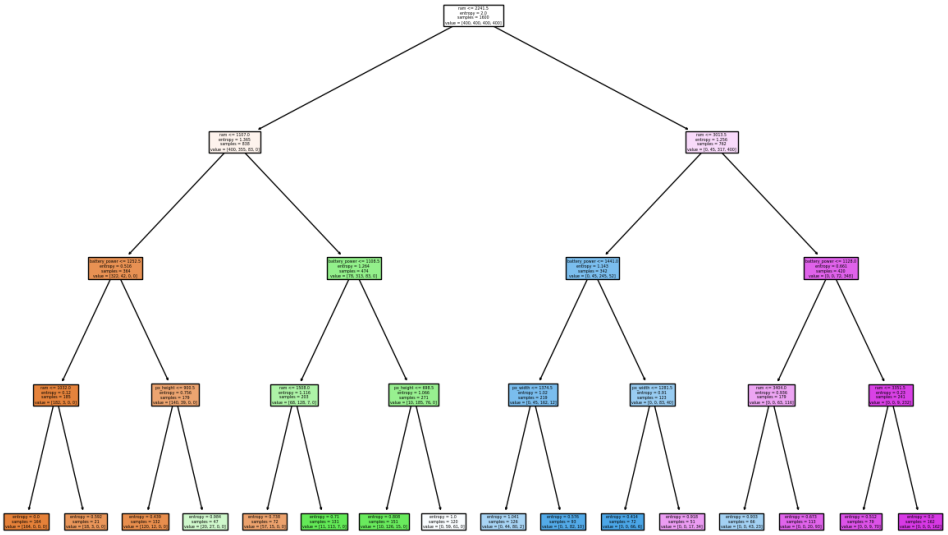


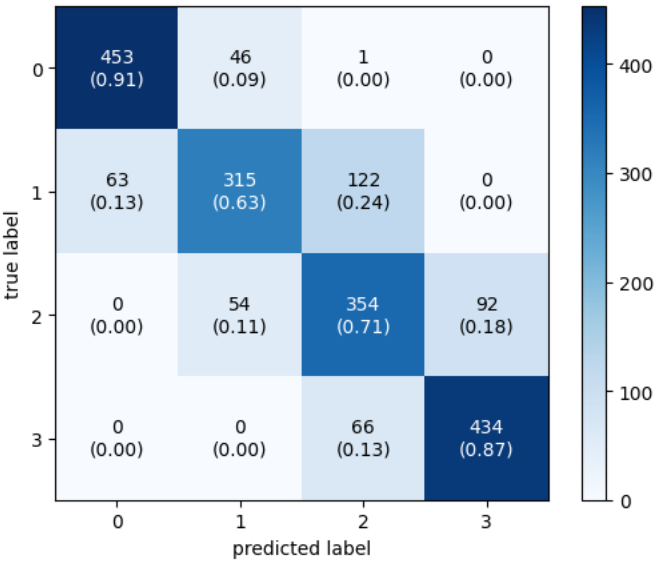

```
Shape of X_train:-   (1600, 9)   ||   X_test:-   (400, 9)   || Y_train:-  (1600,)  ||  Y_test:-
Train:-   [400 400 400 400]   ||  Test:-    [100 100 100 100]
Test Accuracy76.5
ROC_AUC_SCORE: 0.93815
```






```
 Result Of  Decisiom Tree:

average accuracy:  77.8
[[453  46   1   0]
 [ 63 315 122   0]
 [  0  54 354  92]
 [  0   0  66 434]]
```

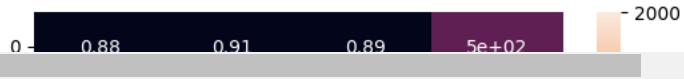




ROC_AUC_SCORE: 0.955579
Classification Report:

	0	1	2	3	accuracy	\
precision	0.877907	0.759036	0.651934	0.825095	0.778	
recall	0.906000	0.630000	0.708000	0.868000	0.778	
f1-score	0.891732	0.688525	0.678811	0.846004	0.778	
support	500.000000	500.000000	500.000000	500.000000	0.778	

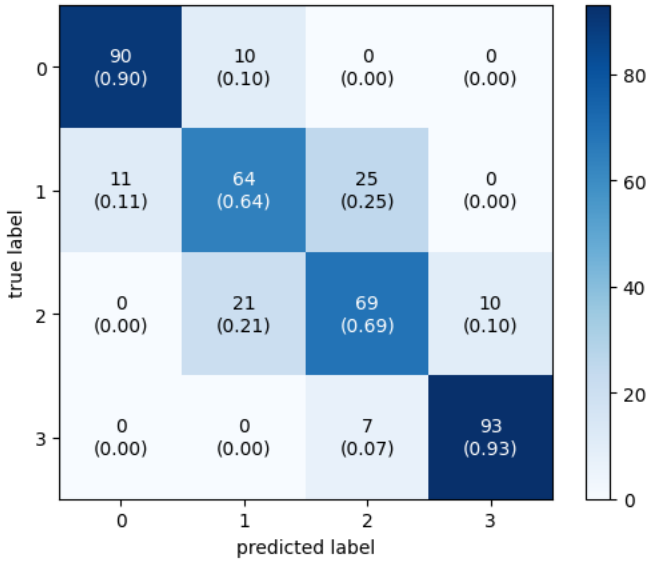
	macro avg	weighted avg
precision	0.778493	0.778493
recall	0.778000	0.778000
f1-score	0.776268	0.776268
support	2000.000000	2000.000000



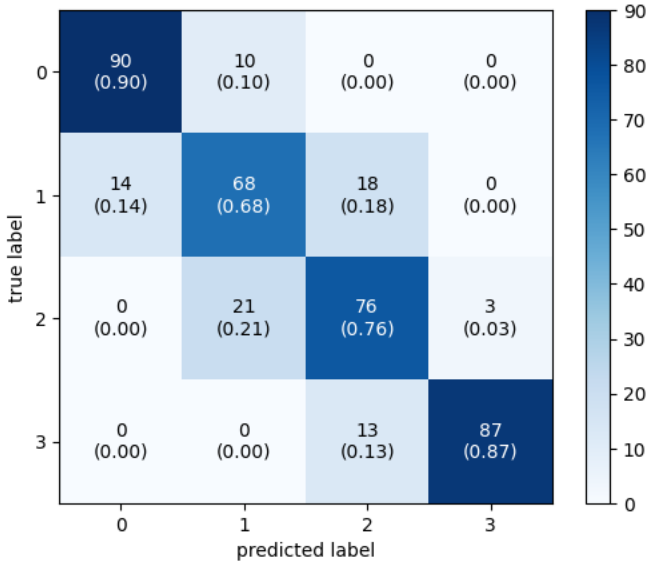
Gaussian Classification

```
from sklearn.naive_bayes import GaussianNB
def Gaussian_Naive_Bayes(Independent_Attributes,Dependent_Attribute):
  X=Independent_Attributes.values
  Y=Dependent_Attribute.values
  print("*************Normalization/Standardization*************")
  XScaled = normalize(X)

  skf=StratifiedKFold(n_splits=5, shuffle=True, random_state=2)
  acc=[]

  y_ori = np.array([], dtype=int)

  y_pre= np.array([], dtype=int)
  net_mat=np.zeros((4, 4))
  for train_index, test_index in skf.split(X,Y):

      X_train=X[train_index]
      X_test=X[test_index]
      Y_train=Y[train_index]
      Y_test=Y[test_index]
      print(bold(green("Shape of X_train:-   {}   ||   X_test:-   {}   || Y_train:-  {}  ||  Y_test:-   {}  ".format(X_train.shape,X_test.shape,Y_train.shap

      print(bold(magenta("Train:-   {}   ||  Test:-    {} ".format(np.bincount(Y[train_index]),np.bincount(Y[test_index])))))

      GN = GaussianNB()
      GN.fit(X_train, Y_train)
      Y_testPred = GN.predict(X_test)

      y_ori=np.hstack((y_ori,Y_test))


      y_pre=np.hstack((y_pre,Y_testPred))

      testAccuracy = metrics.accuracy_score(Y_test, Y_testPred)
      print(bold(red("Test Accuracy", testAccuracy*100)))
      acc.append(testAccuracy)

      roc_score=roc_auc_score(Y_test, GN.predict_proba(X_test), multi_class='ovr')

      print(bold(yellow("ROC_AUC_SCORE: ",roc_score)))

      matrix1= confusion_matrix(Y_test, Y_testPred)
      #      sum of the total confusion matirx
      net_mat=net_mat+matrix1


      plot_confusion_matrix(matrix1,show_normed=True, colorbar=True, show_absolute=True)

      plt.show()
  print(bold(green("Result Of Gaussian Naive Bayes Classifier: \n")))
  print(bold(magenta("Naive Bayes score: ",GN.score(X_test, Y_testPred))))
  avg_accuracy= (sum(acc) / len(acc))*100
  print(bold(cyan_bg("average accuracy:  ", avg_accuracy)))

  net_mat = net_mat.astype('int')
  print(bold(cyan(net_mat)))
  plot_confusion_matrix(net_mat,show_normed=True, colorbar=True, show_absolute=True, cmap='Blues')

  plt.show()

  roc_score=roc_auc_score(Y,GN.predict_proba(X), multi_class='ovr')
  print(bold(yellow_bg("ROC_AUC_SCORE: ",roc_score)))



  print("Classification Report:\n")
  report=classification_report(y_ori, y_pre,output_dict=True)
  report_Df=pd.DataFrame(report)
  print(bold(blue(report_Df)))
  sns.heatmap(report_Df.T,annot=True)
  return avg_accuracy
avg_accuracy_GNB=Gaussian_Naive_Bayes(Independent_Attributes,Dependent_Attribute)
```

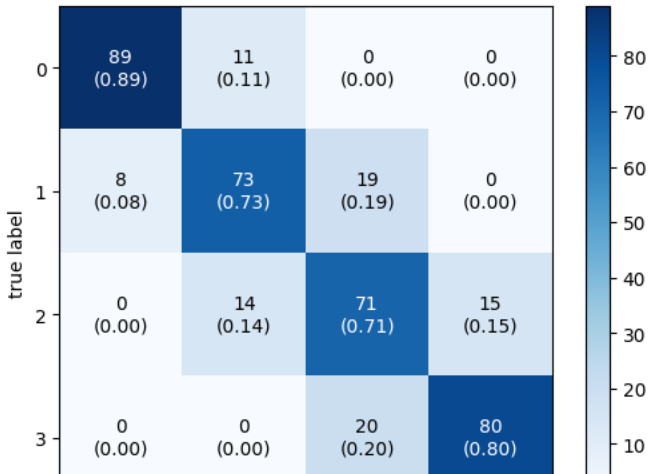
```
************Normalization/Standardization************
Mean and Standard Deviation Before
[2.1242130e+03 1.2385185e+03 1.2515155e+03 6.4510800e+02 3.2046500e+01
 5.7670000e+00 9.9165000e+00 7.6150000e-01 1.2306500e+01] [1.08446083e+03 4.39308338e+02 4.320913
 1.81411780e+01 4.35530837e+00 6.06279867e+00 4.26166341e-01
 4.21219156e+00]
Mean and Standard Deviation After
[-0.  0.  0.  0. -0. -0.  0.  0.  0.] [1. 1. 1. 1. 1. 1. 1. 1. 1.]
Shape of X_train:-   (1600, 9)   ||   X_test:-   (400, 9)   || Y_train:-  (1600,)  ||  Y_test:-
Train:-   [400 400 400 400]   ||  Test:-    [100 100 100 100]
Test Accuracy79.0
ROC_AUC_SCORE: 0.9438833333333333
```




```
Shape of X_train:-   (1600, 9)   ||   X_test:-   (400, 9)   || Y_train:-  (1600,)  ||  Y_test:-
Train:-   [400 400 400 400]   ||  Test:-    [100 100 100 100]
Test Accuracy80.25
ROC_AUC_SCORE: 0.9558500000000001
```




```
Shape of X_train:-   (1600, 9)   ||   X_test:-   (400, 9)   || Y_train:-  (1600,)  ||  Y_test:-
Train:-   [400 400 400 400]   ||  Test:-    [100 100 100 100]
Test Accuracy78.25
ROC_AUC_SCORE: 0.943
```



KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier
def KNN(Independent_Attributes,Dependent_Attribute):
  X=Independent_Attributes.values
  Y=Dependent_Attribute.values
  print("*************Normalization/Standardization************")
  XScaled = normalize(X)

  skf=StratifiedKFold(n_splits=5, shuffle=True, random_state=2)
  acc=[]

  y_ori = np.array([], dtype=int)

  y_pre= np.array([], dtype=int)
  net_mat=np.zeros((4, 4))
  for train_index, test_index in skf.split(X,Y):

      X_train=X[train_index]
      X_test=X[test_index]
      Y_train=Y[train_index]
      Y_test=Y[test_index]
      print(bold(green("Shape of X_train:-   {}   ||   X_test:-   {}   || Y_train:-  {}  ||  Y_test:-   {}  ".format(X_train.shape,X_test.shape,Y_train.shap
      print(bold(magenta("Train:-   {}   ||  Test:-    {} ".format(np.bincount(Y[train_index]),np.bincount(Y[test_index])))))

      Kn = KNeighborsClassifier(n_neighbors=10)
      Kn.fit(X_train, Y_train)
      Y_testPred = Kn.predict(X_test)

      y_ori=np.hstack((y_ori,Y_test))


      y_pre=np.hstack((y_pre,Y_testPred))

      testAccuracy = metrics.accuracy_score(Y_test, Y_testPred)


      print(bold(red("Test Accuracy", testAccuracy*100)))
      acc.append(testAccuracy)

      roc_score=roc_auc_score(Y_test, Kn.predict_proba(X_test), multi_class='ovr')

      print(bold(yellow("ROC_AUC_SCORE: ",roc_score)))


      matrix1= confusion_matrix(Y_test, Y_testPred)
      # sum of the total confusion matirx
      net_mat=net_mat+matrix1


      plot_confusion_matrix(matrix1,show_normed=True, colorbar=True, show_absolute=True)

      plt.show()
  avg_accuracy=(sum(acc) / len(acc))*100
  print(bold(yellow_bg("average accuracy:  ", avg_accuracy)))

  net_mat = net_mat.astype('int')
  print(bold(cyan(net_mat)))
  plot_confusion_matrix(net_mat,show_normed=True, colorbar=True, show_absolute=True, cmap='Blues')

  plt.show()

  roc_score=roc_auc_score(Y,Kn.predict_proba(X), multi_class='ovr')
  print(bold(yellow_bg("ROC_AUC_SCORE: ",roc_score)))



  print(bold("Classification Report:\n"))
  report=classification_report(y_ori, y_pre,output_dict=True)
  report_Df=pd.DataFrame(report)
  print(bold(blue(report_Df)))
  sns.heatmap(report_Df.T,annot=True)
  return avg_accuracy
avg_accuracy_knn=KNN(Independent_Attributes,Dependent_Attribute)
```