```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LogisticRegression
        from sklearn.preprocessing import StandardScaler
        from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.preprocessing import StandardScaler
        from sklearn.svm import SVC
```

```
In [2]: df = pd.read_csv("mobile_price_range_data.csv")
```

```
In [3]: df.head()
```

Out[3]:

	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	three_g	touch_screen	wifi
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	0	0	1
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	1	1	0
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	1	1	0
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	1	0	0
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	1	1	0

5 rows × 21 columns

```
In [4]: df.tail()
```

Out[4]:

	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	three_g	touch_screen	wifi
1995	794	1	0.5	1	0	1	2	0.8	106	6	...	1222	1890	668	13	4	19	1	1	0
1996	1965	1	2.6	1	0	0	39	0.2	187	4	...	915	1965	2032	11	10	16	1	1	1
1997	1911	0	0.9	1	1	1	36	0.7	108	8	...	868	1632	3057	9	1	5	1	1	0
1998	1512	0	0.9	0	4	1	46	0.1	145	5	...	336	670	869	18	10	19	1	1	1
1999	510	1	2.0	1	5	1	45	0.9	168	6	...	483	754	3919	19	4	2	1	1	1

5 rows × 21 columns

```
In [5]: df.shape

Out[5]: (2000, 21)
```

```
In [6]: df.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
```

```
In [7]: df.describe

Out[7]: <bound method NDFrame.describe of       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]>
```

Data Preprocessing

```
In [8]: df.isnull().sum()

Out[8]: 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
```

```
In [9]: df.duplicated().sum()

Out[9]: 0
```

```
In [10]: df.dtypes

Out[10]: 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
```

Set X(Independent) and y(Dependent) values.

```
In [11]: X = df.drop('price_range', axis=1)
         y = df['price_range']
```

Split data into Train and Test data.

```
In [12]: X_train, X_test, y_train, y_test = train_test_split(X , y , test_size=0.20 , random_state=100)
```

```
In [13]: X_train.shape

Out[13]: (1600, 20)
```

```
In [14]: X_test.shape

Out[14]: (400, 20)
```

```
In [15]: y_train.shape

Out[15]: (1600,)
```

```
In [16]: y_test.shape

Out[16]: (400,)
```

Model Building.

```
In [17]: st_x= StandardScaler()
         X_train= st_x.fit_transform(X_train)
         X_test= st_x.transform(X_test)
```

1) Using Logistic Regression

```
In [18]: lr = LogisticRegression()
         lr.fit(X_train , y_train)

Out[18]: LogisticRegression()
```

```
In [19]: print("Train Score = ",lr.score(X_train,y_train))
         print("Test Score = ",lr.score(X_test,y_test))

Train Score =  0.978125
Test Score =  0.9525
```

```
In [20]: y_pred_lr = lr.predict(X_test)
         y_pred_lr

Out[20]: array([0, 2, 1, 3, 2, 3, 3, 2, 3, 0, 0, 2, 3, 3, 0, 2, 2, 3, 2, 0, 0, 3,
       1, 0, 0, 1, 3, 0, 2, 2, 0, 3, 0, 1, 0, 3, 3, 2, 1, 3, 3, 3, 0, 0,
       3, 3, 2, 0, 1, 2, 3, 2, 3, 2, 1, 3, 1, 3, 1, 0, 3, 2, 3, 0, 2, 0,
       0, 3, 3, 2, 0, 0, 2, 1, 2, 3, 0, 3, 2, 3, 1, 0, 2, 0, 2, 2, 1, 2,
       3, 3, 2, 3, 3, 2, 0, 2, 3, 1, 1, 1, 0, 0, 3, 3, 2, 0, 0, 1, 0, 1,
       1, 1, 3, 2, 0, 3, 1, 3, 1, 1, 3, 1, 0, 1, 3, 2, 3, 2, 0, 3, 2, 0,
       1, 2, 3, 1, 3, 2, 0, 1, 3, 0, 2, 3, 1, 2, 2, 2, 0, 1, 0, 3, 2, 2,
       1, 0, 2, 3, 1, 1, 3, 2, 3, 0, 0, 3, 0, 2, 3, 0, 2, 0, 2, 1, 1, 2,
       2, 1, 3, 1, 0, 2, 1, 0, 0, 3, 2, 2, 3, 2, 2, 1, 2, 1, 2, 2, 3, 1,
       1, 1, 3, 1, 2, 1, 3, 0, 2, 3, 3, 2, 0, 1, 2, 0, 1, 2, 3, 0, 1, 3,
       0, 3, 0, 0, 1, 2, 2, 3, 2, 0, 2, 2, 0, 1, 1, 3, 2, 3, 2, 0, 1, 3,
       1, 1, 0, 2, 3, 1, 1, 1, 1, 2, 3, 1, 3, 2, 1, 1, 2, 3, 1, 3, 2, 2,
       2, 2, 3, 1, 2, 1, 1, 3, 2, 0, 2, 1, 1, 3, 2, 2, 2, 1, 3, 1, 0, 2,
       3, 0, 0, 1, 3, 2, 1, 2, 2, 0, 2, 3, 0, 3, 3, 3, 0, 0, 3, 0, 3, 0,
       0, 0, 3, 0, 1, 1, 0, 1, 1, 3, 1, 1, 3, 3, 0, 0, 0, 0, 2, 1, 1,
       3, 1, 0, 0, 1, 2, 3, 1, 2, 1, 0, 2, 0, 0, 3, 2, 1, 1, 0, 2, 0, 3,
       3, 0, 2, 3, 0, 0, 1, 2, 3, 3, 1, 3, 0, 3, 1, 2, 1, 0, 2, 0, 0, 2,
       2, 3, 0, 1, 1, 3, 0, 3, 0, 1, 0, 1, 0, 2, 2, 1, 1, 3, 0, 1, 0, 3,
       0, 3, 0, 0], dtype=int64)
```

Confusion Matrix

```
In [21]: cm = confusion_matrix(y_test , y_pred_lr)
         print(cm)

[[ 99   1   0   0]
 [  2  89   8   0]
 [  0   5  90   1]
 [  0   0   2 103]]
```

Classification Report

```
In [22]: cr = classification_report(y_test , y_pred_lr)
         print(cr)

              precision    recall  f1-score   support

           0       0.98      0.99      0.99       100
           1       0.94      0.90      0.92        99
           2       0.90      0.94      0.92        96
           3       0.99      0.98      0.99       105

    accuracy                           0.95       400
   macro avg       0.95      0.95      0.95       400
weighted avg       0.95      0.95      0.95       400
```

```
In [ ]:
```

```
In [ ]:
```

2) Using KNN Classification

```
In [23]: KNN = KNeighborsClassifier(n_neighbors = 21)
         KNN.fit(X_train , y_train)

Out[23]: KNeighborsClassifier(n_neighbors=21)
```

```
In [24]: print('Train Score = ',KNN.score(X_train , y_train))
         print('Test Score = ',KNN.score(X_test , y_test))

Train Score =  0.7025
Test Score =  0.58
```

```
In [25]: y_pred_knn = KNN.predict(X_test)
         y_pred_knn

Out[25]: array([1, 2, 1, 1, 2, 1, 3, 3, 3, 0, 0, 2, 1, 3, 0, 0, 3, 2, 3, 1, 0, 3,
       1, 1, 0, 1, 3, 1, 2, 3, 0, 1, 0, 1, 0, 3, 3, 0, 0, 3, 3, 3, 0, 0,
       3, 3, 2, 0, 0, 3, 3, 3, 2, 1, 1, 3, 1, 3, 1, 0, 2, 0, 2, 0, 2, 0,
       0, 3, 3, 0, 0, 0, 1, 2, 2, 2, 0, 3, 2, 3, 1, 2, 2, 0, 2, 2, 2, 1,
       3, 3, 2, 2, 2, 2, 0, 2, 3, 0, 1, 0, 1, 1, 3, 3, 2, 1, 0, 0, 1, 1,
       1, 1, 3, 2, 0, 2, 1, 3, 0, 1, 2, 0, 0, 1, 3, 3, 2, 3, 0, 2, 1, 1,
       1, 3, 3, 0, 3, 2, 1, 0, 3, 0, 3, 3, 0, 3, 3, 1, 0, 1, 0, 2, 1, 2,
       1, 0, 2, 2, 1, 1, 2, 1, 3, 0, 2, 2, 0, 3, 3, 0, 2, 1, 2, 1, 1, 1,
       2, 1, 3, 0, 0, 1, 2, 0, 0, 3, 3, 3, 3, 0, 2, 0, 1, 1, 2, 3, 3, 0,
       1, 1, 3, 1, 2, 1, 3, 0, 2, 3, 2, 1, 0, 2, 2, 1, 1, 2, 2, 0, 2, 3,
       0, 2, 0, 0, 2, 2, 1, 3, 2, 0, 2, 2, 0, 1, 1, 3, 2, 3, 2, 0, 0, 3,
       0, 1, 0, 2, 2, 0, 2, 1, 1, 1, 3, 0, 3, 2, 2, 1, 2, 1, 1, 2, 2, 3,
       1, 1, 2, 0, 3, 2, 1, 2, 0, 0, 1, 1, 0, 3, 1, 2, 3, 1, 3, 1, 0, 2,
       2, 1, 0, 0, 1, 1, 2, 2, 2, 0, 2, 3, 1, 3, 3, 2, 0, 0, 3, 0, 2, 1,
       0, 0, 2, 0, 3, 2, 0, 0, 1, 3, 1, 3, 2, 3, 0, 0, 0, 0, 1, 0, 1, 0,
       3, 0, 0, 0, 0, 2, 2, 2, 2, 2, 0, 3, 0, 1, 3, 2, 3, 0, 0, 2, 0, 2,
       2, 1, 3, 2, 1, 0, 0, 1, 3, 2, 0, 3, 0, 2, 2, 2, 1, 0, 1, 0, 1, 3,
       1, 3, 2, 1, 1, 1, 0, 3, 0, 2, 0, 2, 0, 2, 3, 1, 0, 3, 0, 0, 0, 2,
       0, 3, 1, 0], dtype=int64)
```

Confusion Matrix

```
In [26]: cm = confusion_matrix(y_test , y_pred_knn)
         print(cm)

[[76 20  4  0]
 [30 47 19  3]
 [ 7 22 45 22]
 [ 0  7 34 64]]
```

Classification Report

```
In [27]: cr = classification_report(y_test , y_pred_knn)
         print(cr)

              precision    recall  f1-score   support

           0       0.67      0.76      0.71       100
           1       0.49      0.47      0.48        99
           2       0.44      0.47      0.45        96
           3       0.72      0.61      0.66       105

    accuracy                           0.58       400
   macro avg       0.58      0.58      0.58       400
weighted avg       0.58      0.58      0.58       400
```

```
In [ ]:
```

```
In [ ]:
```

3) Using SVM Classifier with linear and rbf kernel

SVM Classifier with linear kernel

```
In [28]: svm1 = SVC(kernel = 'linear' , C =1)
         svm1.fit(X_train , y_train)

Out[28]: SVC(C=1, kernel='linear')
```

```
In [29]: print('Train Score = ',svm1.score(X_train,y_train))
         print('Test Score = ',svm1.score(X_test,y_test))

Train Score =  0.975
Test Score =  0.9575
```

```
In [30]: y_pred_SVM_L = svm1.predict(X_test)
         y_pred_SVM_L

Out[30]: array([0, 2, 1, 3, 2, 3, 3, 2, 3, 0, 0, 2, 3, 3, 0, 2, 2, 3, 2, 0, 0, 3,
       1, 0, 0, 1, 3, 0, 2, 2, 0, 3, 0, 1, 0, 3, 3, 2, 1, 3, 3, 3, 0, 0,
       3, 3, 2, 0, 1, 2, 3, 2, 3, 2, 1, 3, 1, 3, 1, 0, 3, 2, 3, 0, 2, 0,
       0, 3, 3, 2, 0, 0, 2, 1, 2, 3, 0, 3, 2, 3, 1, 0, 2, 0, 2, 2, 1, 2,
       3, 3, 2, 3, 3, 2, 0, 2, 3, 1, 2, 1, 0, 0, 3, 3, 2, 0, 0, 1, 0, 1,
       1, 1, 3, 2, 0, 3, 1, 3, 1, 1, 3, 1, 0, 1, 3, 2, 3, 2, 0, 3, 2, 0,
       1, 2, 3, 1, 3, 2, 0, 1, 3, 0, 2, 3, 1, 2, 2, 2, 0, 1, 0, 3, 1, 2,
       1, 0, 2, 3, 1, 1, 3, 2, 3, 0, 1, 3, 0, 2, 3, 0, 2, 0, 2, 1, 1, 2,
       2, 1, 3, 1, 0, 2, 1, 0, 0, 3, 2, 2, 3, 2, 2, 1, 2, 1, 2, 2, 3, 1,
       1, 1, 3, 1, 2, 1, 3, 0, 1, 3, 3, 2, 0, 1, 2, 0, 1, 2, 3, 0, 2, 3,
       0, 3, 0, 0, 1, 2, 2, 3, 2, 0, 2, 2, 0, 1, 1, 3, 2, 3, 2, 0, 1, 3,
       1, 1, 0, 2, 3, 1, 1, 1, 1, 2, 3, 1, 3, 2, 2, 1, 2, 3, 1, 3, 2, 2,
       2, 2, 2, 1, 2, 1, 1, 3, 2, 0, 2, 1, 1, 3, 2, 2, 2, 1, 3, 1, 0, 2,
       3, 0, 0, 1, 3, 2, 1, 2, 3, 0, 2, 3, 0, 3, 3, 3, 0, 0, 3, 0, 3, 0,
       0, 0, 3, 0, 1, 1, 0, 1, 1, 3, 1, 1, 3, 3, 0, 0, 0, 0, 0, 2, 1, 1,
       3, 1, 0, 0, 1, 2, 3, 1, 2, 1, 0, 2, 0, 0, 3, 2, 1, 1, 0, 2, 0, 3,
       3, 0, 2, 3, 0, 0, 1, 2, 3, 3, 1, 3, 0, 3, 1, 2, 1, 0, 2, 0, 0, 2,
       2, 3, 0, 1, 1, 3, 0, 3, 0, 1, 0, 1, 0, 2, 2, 1, 1, 3, 0, 1, 0, 3,
       0, 3, 0, 0], dtype=int64)
```

Confusion Matrix

```
In [31]: cm = confusion_matrix(y_pred_SVM_L , y_test)
         print(cm)

[[ 98   2   0   0]
 [  2  90   3   0]
 [  0   7  92   2]
 [  0   0   1 103]]
```

Classification Matrix

```
In [32]: cr = classification_report(y_pred_SVM_L , y_test)
         print(cr)

              precision    recall  f1-score   support

           0       0.98      0.98      0.98       100
           1       0.91      0.95      0.93        95
           2       0.96      0.91      0.93       101
           3       0.98      0.99      0.99       104

    accuracy                           0.96       400
   macro avg       0.96      0.96      0.96       400
weighted avg       0.96      0.96      0.96       400
```

```
In [ ]:
```

SVM Classifier with rbf kernel

```
In [33]: svm2 = SVC(kernel = 'rbf' , C =1)
         svm2.fit(X_train , y_train)

Out[33]: SVC(C=1)
```

```
In [34]: print('Train Score = ',svm2.score(X_train,y_train))
         print('Test Score = ',svm2.score(X_test,y_test))

Train Score =  0.985
Test Score =  0.86
```

```
In [35]: y_pred_SVC_R = svm2.predict(X_test)
         y_pred_SVC_R

Out[35]: array([0, 1, 1, 3, 3, 3, 3, 2, 3, 0, 0, 2, 3, 3, 0, 2, 2, 3, 2, 0, 0, 3,
       1, 1, 0, 1, 3, 0, 1, 2, 0, 3, 0, 1, 0, 3, 3, 2, 1, 3, 3, 3, 0, 0,
       3, 3, 2, 0, 1, 2, 3, 2, 3, 1, 1, 3, 1, 3, 1, 0, 3, 2, 2, 0, 2, 0,
       0, 3, 3, 2, 0, 1, 2, 1, 2, 3, 0, 3, 2, 3, 1, 0, 2, 0, 2, 2, 2, 2,
       3, 3, 2, 3, 3, 2, 0, 2, 3, 0, 1, 1, 0, 0, 2, 3, 2, 1, 0, 1, 1, 1,
       1, 1, 3, 2, 0, 3, 1, 2, 0, 1, 2, 1, 0, 2, 3, 2, 3, 2, 0, 2, 1, 0,
       1, 2, 3, 0, 3, 2, 0, 0, 3, 0, 2, 3, 2, 2, 2, 2, 0, 1, 0, 3, 2, 2,
       1, 0, 1, 3, 1, 1, 3, 2, 3, 0, 0, 3, 0, 2, 3, 0, 2, 1, 2, 1, 1, 1,
       2, 1, 3, 1, 0, 2, 1, 1, 0, 3, 2, 2, 3, 2, 2, 1, 2, 1, 2, 3, 3, 1,
       2, 1, 3, 2, 2, 1, 3, 0, 2, 3, 3, 2, 0, 1, 2, 0, 1, 2, 3, 0, 1, 3,
       0, 3, 0, 0, 1, 2, 2, 3, 2, 0, 1, 2, 0, 1, 1, 3, 2, 3, 2, 0, 0, 3,
       1, 1, 0, 2, 3, 0, 2, 1, 1, 2, 3, 1, 3, 2, 2, 1, 2, 3, 1, 3, 2, 2,
       2, 2, 2, 1, 2, 1, 1, 3, 2, 0, 2, 1, 1, 3, 1, 1, 2, 1, 3, 1, 0, 2,
       3, 0, 0, 0, 2, 2, 1, 2, 2, 0, 2, 3, 0, 3, 3, 2, 0, 0, 3, 0, 3, 0,
       0, 0, 3, 0, 1, 2, 0, 1, 1, 3, 1, 1, 3, 3, 0, 0, 0, 0, 0, 2, 1, 1,
       3, 1, 0, 0, 1, 2, 3, 2, 2, 1, 0, 3, 0, 1, 3, 2, 1, 2, 0, 2, 0, 3,
       2, 0, 2, 3, 0, 0, 1, 2, 3, 3, 2, 3, 0, 3, 1, 2, 0, 0, 2, 0, 0, 2,
       2, 3, 0, 1, 1, 2, 0, 3, 0, 2, 0, 1, 0, 2, 3, 1, 1, 3, 0, 1, 0, 3,
       1, 3, 0, 0], dtype=int64)
```

Confusion Matrix

```
In [36]: cm = confusion_matrix(y_pred_SVC_R , y_test)
         print(cm)

[[92  9  0  0]
 [ 8 75  9  0]
 [ 0 15 83 11]
 [ 0  0  4 94]]
```

Classification Matrix

```
In [37]: cr = classification_report(y_pred_SVC_R , y_test)
         print(cr)

              precision    recall  f1-score   support

           0       0.92      0.91      0.92       101
           1       0.76      0.82      0.79        92
           2       0.86      0.76      0.81       109
           3       0.90      0.96      0.93        98

    accuracy                           0.86       400
   macro avg       0.86      0.86      0.86       400
weighted avg       0.86      0.86      0.86       400
```

```
In [ ]:
```

```
In [ ]:
```

4)Decision Tree Classifier

```
In [38]: dtc = DecisionTreeClassifier()
         dtc.fit(X_train , y_train)

Out[38]: DecisionTreeClassifier()
```

```
In [39]: y_pred_DTC = dtc.predict(X_test)
         y_pred_DTC

Out[39]: array([0, 2, 1, 3, 2, 3, 3, 3, 3, 1, 0, 1, 3, 3, 0, 1, 2, 2, 2, 0, 0, 2,
       1, 1, 0, 1, 3, 0, 1, 2, 0, 3, 0, 1, 0, 3, 2, 1, 1, 3, 3, 2, 0, 0,
       3, 3, 2, 0, 1, 2, 3, 3, 3, 2, 1, 3, 1, 3, 1, 1, 3, 2, 2, 0, 2, 0,
       0, 3, 3, 2, 1, 1, 2, 1, 1, 3, 0, 3, 2, 3, 1, 0, 2, 0, 2, 2, 1, 2,
       3, 3, 3, 3, 3, 2, 0, 2, 3, 1, 2, 1, 1, 0, 2, 3, 2, 0, 0, 1, 0, 1,
       0, 1, 3, 2, 0, 3, 1, 3, 0, 1, 3, 1, 0, 2, 3, 2, 3, 2, 0, 2, 1, 0,
       0, 2, 3, 0, 3, 2, 0, 0, 2, 1, 3, 3, 1, 2, 2, 2, 0, 2, 0, 3, 1, 2,
       1, 0, 2, 3, 1, 1, 3, 2, 3, 0, 1, 3, 0, 2, 3, 0, 3, 1, 1, 1, 2, 1,
       3, 1, 3, 0, 0, 2, 1, 0, 0, 3, 2, 2, 3, 2, 2, 1, 1, 1, 3, 2, 3, 2,
       2, 1, 3, 1, 2, 1, 3, 0, 2, 3, 3, 2, 0, 1, 2, 0, 0, 2, 3, 0, 1, 3,
       0, 3, 0, 0, 2, 1, 2, 2, 2, 1, 2, 2, 0, 1, 1, 3, 3, 2, 2, 0, 1, 3,
       1, 1, 0, 2, 3, 0, 2, 1, 1, 3, 3, 1, 2, 3, 1, 1, 2, 2, 1, 3, 3, 2,
       2, 2, 2, 1, 2, 1, 0, 2, 2, 0, 2, 1, 1, 3, 2, 2, 2, 1, 3, 1, 0, 1,
       3, 0, 0, 1, 2, 2, 1, 2, 3, 0, 2, 3, 0, 3, 3, 2, 0, 0, 3, 0, 2, 0,
       0, 0, 3, 0, 1, 2, 0, 1, 1, 3, 2, 1, 3, 3, 0, 0, 0, 0, 0, 2, 1, 1,
       3, 1, 0, 0, 1, 2, 3, 1, 2, 1, 0, 3, 0, 0, 3, 2, 2, 1, 0, 2, 0, 3,
       3, 1, 2, 3, 0, 0, 1, 2, 2, 3, 1, 3, 0, 3, 1, 2, 1, 0, 2, 0, 0, 2,
       2, 3, 1, 1, 1, 2, 0, 3, 0, 1, 0, 1, 0, 2, 2, 0, 1, 3, 0, 1, 0, 3,
       1, 3, 0, 0], dtype=int64)
```

Confusion Matrix

```
In [40]: cm = confusion_matrix(y_pred_DTC , y_test)
         print(cm)

[[87 11  0  0]
 [13 72 14  0]
 [ 0 16 69 20]
 [ 0  0 13 85]]
```

Classification Matrix

```
In [41]: cr = classification_report(y_pred_DTC , y_test)
         print(cr)

              precision    recall  f1-score   support

           0       0.87      0.89      0.88        98
           1       0.73      0.73      0.73        99
           2       0.72      0.66      0.69       105
           3       0.81      0.87      0.84        98

    accuracy                           0.78       400
   macro avg       0.78      0.78      0.78       400
weighted avg       0.78      0.78      0.78       400
```

```
In [ ]:
```

```
In [ ]:
```

5) Random Forest Classifier

```
In [42]: RF= RandomForestClassifier(n_estimators= 10, criterion="entropy")
         RF.fit(X_train, y_train)

Out[42]: RandomForestClassifier(criterion='entropy', n_estimators=10)
```

```
In [43]: print('Train Score = ',RF.score(X_train,y_train))
         print('Test Score = ',RF.score(X_test,y_test))

Train Score =  0.998125
Test Score =  0.81
```

```
In [44]: y_pred_RF = RF.predict(X_test)
         y_pred_RF

Out[44]: array([0, 2, 1, 3, 2, 3, 3, 3, 3, 0, 0, 1, 3, 3, 0, 2, 2, 3, 2, 0, 0, 2,
       1, 0, 0, 1, 2, 0, 1, 2, 0, 3, 0, 1, 0, 3, 3, 2, 1, 3, 3, 3, 0, 0,
       3, 3, 2, 0, 1, 2, 3, 2, 3, 2, 1, 3, 1, 3, 1, 0, 2, 2, 3, 0, 3, 0,
       1, 3, 3, 2, 0, 1, 2, 1, 2, 2, 0, 3, 2, 3, 2, 0, 2, 0, 2, 2, 1, 2,
       3, 3, 3, 3, 3, 2, 0, 2, 3, 1, 1, 1, 0, 0, 2, 3, 2, 0, 0, 0, 1, 1,
       0, 1, 3, 2, 0, 2, 1, 3, 1, 1, 3, 1, 1, 2, 3, 2, 2, 2, 0, 3, 1, 0,
       1, 2, 3, 1, 3, 2, 0, 1, 3, 0, 3, 3, 1, 2, 2, 2, 0, 1, 0, 2, 2, 2,
       1, 0, 2, 3, 1, 1, 3, 2, 3, 0, 1, 2, 0, 3, 3, 0, 3, 1, 2, 1, 1, 1,
       2, 1, 3, 1, 0, 2, 1, 0, 0, 3, 2, 2, 3, 2, 1, 1, 1, 1, 2, 3, 3, 1,
       2, 1, 3, 1, 2, 1, 3, 0, 2, 3, 3, 2, 0, 1, 2, 0, 0, 2, 3, 0, 1, 3,
       0, 3, 0, 0, 1, 1, 2, 3, 2, 0, 1, 2, 0, 1, 1, 3, 3, 3, 2, 0, 1, 3,
       2, 1, 0, 2, 2, 1, 2, 1, 1, 2, 3, 1, 2, 2, 1, 1, 1, 2, 1, 3, 2, 2,
       2, 1, 3, 0, 1, 1, 0, 2, 2, 0, 2, 1, 1, 3, 2, 2, 2, 1, 3, 1, 0, 2,
       2, 0, 0, 0, 3, 2, 1, 2, 2, 0, 2, 3, 0, 2, 3, 2, 0, 0, 3, 0, 1, 0,
       0, 0, 3, 0, 1, 2, 0, 1, 1, 3, 1, 2, 3, 3, 0, 0, 0, 0, 0, 2, 1, 1,
       3, 1, 0, 1, 1, 2, 3, 1, 3, 1, 0, 2, 0, 0, 3, 2, 2, 1, 0, 1, 0, 3,
       2, 0, 2, 3, 1, 0, 1, 2, 2, 3, 1, 3, 0, 3, 1, 2, 0, 0, 1, 0, 0, 2,
       2, 3, 0, 1, 1, 3, 0, 2, 0, 2, 0, 1, 0, 2, 2, 1, 1, 3, 0, 1, 0, 3,
       1, 3, 0, 0], dtype=int64)
```

Confusion Matrix

```
In [45]: cm = confusion_matrix(y_pred_RF , y_test)
         print(cm)

[[91  8  0  0]
 [ 9 77 15  1]
 [ 0 14 72 20]
 [ 0  0  9 84]]
```

Classification Matrix

```
In [46]: cr = classification_report(y_pred_RF , y_test)
         print(cr)

              precision    recall  f1-score   support

           0       0.91      0.92      0.91        99
           1       0.78      0.75      0.77       102
           2       0.75      0.68      0.71       106
           3       0.80      0.90      0.85        93

    accuracy                           0.81       400
   macro avg       0.81      0.81      0.81       400
weighted avg       0.81      0.81      0.81       400
```

```
In [ ]:
```

Result

=> Model With Best Accuracy is "SVM Classifier with linear kernel" with 96% accuracy.

```
In [ ]:
```