PRCP-1009-CellphonePrice

Problem Statement

- Task 1:-Prepare a complete data analysis report on the given data.
- Task 2:-On the basis of the mobile Specification like Battery power, 3G enabled , wifi ,Bluetooth, Ram etc predict the Price range of the mobile.
- Task 3:- Prepare the analysis report stating how model will help expanding the business by stating several factors including feature importance.

Important Library

In [2]:
```
#Important Library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import confusion_matrix,accuracy_score,recall_score,f1_score,precision_score,classificatio
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

Data Collection

In [3]:
```
#Import Data
data=pd.read_csv("datasets_11167_15520_train.csv")
pd.set_option("display.max_rows",None)
```

Rename Columns

In [4]:
```
#Change table name
data=data.rename(columns={"battery_power":"Battery_Power","blue":"Bluetooth","clock_speed":"Clock_Speed","dual_
```

Basic Checks

In [109…
```
a=data.copy()
```

In [110…
```
data.head(5)#Top 5
```

Out[110]:

	Battery_Power	Bluetooth	Clock_Speed	Dual_Sim	Selfi_Camera	4G	Internal_Memory	Mobile_Depth	Mobile_Width	Number_Of_Cores	.
0	842	0	2.2	0	1	0	7	0.6	188	2	.
1	1021	1	0.5	1	0	1	53	0.7	136	3	.
2	563	1	0.5	1	2	1	41	0.9	145	5	.
3	615	1	2.5	0	0	0	10	0.8	131	6	.
4	1821	1	1.2	0	13	1	44	0.6	141	2	.

5 rows × 21 columns

In [111…
```
data.tail(5)#Last 5
```

Out[111]:

	Battery_Power	Bluetooth	Clock_Speed	Dual_Sim	Selfi_Camera	4G	Internal_Memory	Mobile_Depth	Mobile_Width	Number_Of_Cores
1995	794	1	0.5	1	0	1	2	0.8	106	6
1996	1965	1	2.6	1	0	0	39	0.2	187	4
1997	1911	0	0.9	1	1	1	36	0.7	108	8
1998	1512	0	0.9	0	4	1	46	0.1	145	5
1999	510	1	2.0	1	5	1	45	0.9	168	6

5 rows × 21 columns

In [112…
```
data["RAM"].unique()
```

Out[112]:
```
array([2549, 2631, 2603, ..., 2032, 3057, 3919], dtype=int64)
```

In [113…
```
data.shape #Number of row 2000 and columns 21
```

Out[113]:
```
(2000, 21)
```

In [114…
```
data.columns #All columns
```

Out[114]:
```
Index(['Battery_Power', 'Bluetooth', 'Clock_Speed', 'Dual_Sim', 'Selfi_Camera',
       '4G', 'Internal_Memory', 'Mobile_Depth', 'Mobile_Width',
       'Number_Of_Cores', 'Primary_Camera', 'Pixel_Height', 'Pixel_Width',
       'RAM', 'Screen_Height', 'Screen_Width', 'Talk_Time', '3G',
       'Touch_Screen', 'WiFi', 'Price_Range'],
      dtype='object')
```

In [115…
```
pd.set_option("display.max_rows",2000)
pd.set_option("display.max_rows",21)
```

In [116…
```
data.info()#NO Null Value
```

```
<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   Bluetooth        2000 non-null   int64  
 2   Clock_Speed      2000 non-null   float64
 3   Dual_Sim         2000 non-null   int64  
 4   Selfi_Camera     2000 non-null   int64  
 5   4G               2000 non-null   int64  
 6   Internal_Memory  2000 non-null   int64  
 7   Mobile_Depth     2000 non-null   float64
 8   Mobile_Width     2000 non-null   int64  
 9   Number_Of_Cores  2000 non-null   int64  
 10  Primary_Camera   2000 non-null   int64  
 11  Pixel_Height     2000 non-null   int64  
 12  Pixel_Width      2000 non-null   int64  
 13  RAM              2000 non-null   int64  
 14  Screen_Height    2000 non-null   int64  
 15  Screen_Width     2000 non-null   int64  
 16  Talk_Time        2000 non-null   int64  
 17  3G               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 [117…

```
print(min(data["Battery_Power"].unique()))#Range of battery power is 501 to 1998
print(max(data["Battery_Power"].unique()))
```

```
501
1998
```

In [118…

```
data.describe()#Statistical Summary
```

Out[118]:

	Battery_Power	Bluetooth	Clock_Speed	Dual_Sim	Selfi_Camera	4G	Internal_Memory	Mobile_Depth	Mobile_Width	Num
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	140.249000	
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	

8 rows × 21 columns

In [119…

```
data.describe().T #Change row and coumns position
```

Out[119]:

	count	mean	std	min	25%	50%	75%	max
Battery_Power	2000.0	1238.51850	439.418206	501.0	851.75	1226.0	1615.25	1998.0
Bluetooth	2000.0	0.49500	0.500100	0.0	0.00	0.0	1.00	1.0
Clock_Speed	2000.0	1.52225	0.816004	0.5	0.70	1.5	2.20	3.0
Dual_Sim	2000.0	0.50950	0.500035	0.0	0.00	1.0	1.00	1.0
Selfi_Camera	2000.0	4.30950	4.341444	0.0	1.00	3.0	7.00	19.0
4G	2000.0	0.52150	0.499662	0.0	0.00	1.0	1.00	1.0
Internal_Memory	2000.0	32.04650	18.145715	2.0	16.00	32.0	48.00	64.0
Mobile_Depth	2000.0	0.50175	0.288416	0.1	0.20	0.5	0.80	1.0
Mobile_Width	2000.0	140.24900	35.399655	80.0	109.00	141.0	170.00	200.0
Number_Of_Cores	2000.0	4.52050	2.287837	1.0	3.00	4.0	7.00	8.0
Primary_Camera	2000.0	9.91650	6.064315	0.0	5.00	10.0	15.00	20.0
Pixel_Height	2000.0	645.10800	443.780811	0.0	282.75	564.0	947.25	1960.0
Pixel_Width	2000.0	1251.51550	432.199447	500.0	874.75	1247.0	1633.00	1998.0
RAM	2000.0	2124.21300	1084.732044	256.0	1207.50	2146.5	3064.50	3998.0
Screen_Height	2000.0	12.30650	4.213245	5.0	9.00	12.0	16.00	19.0
Screen_Width	2000.0	5.76700	4.356398	0.0	2.00	5.0	9.00	18.0
Talk_Time	2000.0	11.01100	5.463955	2.0	6.00	11.0	16.00	20.0
3G	2000.0	0.76150	0.426273	0.0	1.00	1.0	1.00	1.0
Touch_Screen	2000.0	0.50300	0.500116	0.0	0.00	1.0	1.00	1.0
WiFi	2000.0	0.50700	0.500076	0.0	0.00	1.0	1.00	1.0
Price_Range	2000.0	1.50000	1.118314	0.0	0.75	1.5	2.25	3.0

In [120…

```
#data.describe(include='O')# No categorical columns
```

INSIGHTS

FROM describe() function:gives Statistics Summary- -Statistics summary gives a high-level idea to identify whether the data has any outliers, data entry error, distribution of data such as the data is normally distributed or left/right skewness. 1)Battery_Power,Bluetooth,Clock_Speed,Dual_Sim,Selfi_Camera,4G,Internal_Memory,Mobile_Depth,Mobile_Width,Number_Of_Cores,Prim overall data null value is not present,Data is normally distributed,no error,but in selfi camera have some outlier,and we can see ram and battery is highly correlated

In [121…

```
data.duplicated().sum()#No duplicate value
```

Out[121]: 0

In [122…

```
print(data.nunique().sort_values(ascending=False))#
```

```
RAM                1562
Pixel_Height       1137
Pixel_Width        1109
Battery_Power      1094
Mobile_Width        121
Internal_Memory      63
Clock_Speed          26
Primary_Camera       21
Selfi_Camera         20
Screen_Width         19
Talk_Time            19
Screen_Height        15
Mobile_Depth         10
Number_Of_Cores       8
Price_Range           4
Bluetooth             2
4G                    2
Dual_Sim              2
3G                    2
Touch_Screen          2
WiFi                  2
dtype: int64
```

In [123…

```
data.dtypes#Types of data
```

Out[123]:
```
Battery_Power        int64
Bluetooth            int64
Clock_Speed        float64
Dual_Sim             int64
Selfi_Camera         int64
4G                   int64
Internal_Memory      int64
Mobile_Depth       float64
Mobile_Width         int64
Number_Of_Cores      int64
Primary_Camera       int64
Pixel_Height         int64
Pixel_Width          int64
RAM                  int64
Screen_Height        int64
Screen_Width         int64
Talk_Time            int64
3G                   int64
Touch_Screen         int64
WiFi                 int64
Price_Range          int64
dtype: object
```

In [124…
```
data.isnull().sum()#NO Missing Value
```

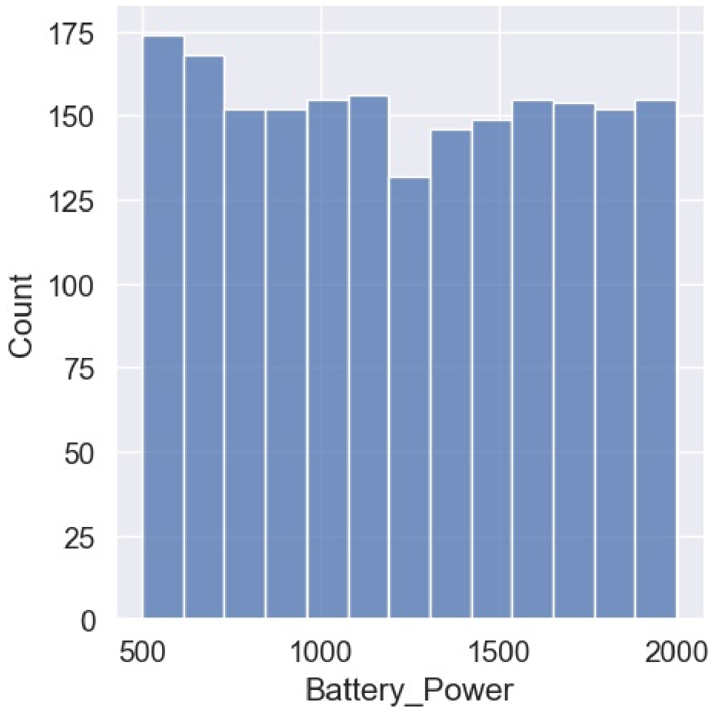
Out[124]:
```
Battery_Power      0
Bluetooth          0
Clock_Speed        0
Dual_Sim           0
Selfi_Camera       0
4G                 0
Internal_Memory    0
Mobile_Depth       0
Mobile_Width       0
Number_Of_Cores    0
Primary_Camera     0
Pixel_Height       0
Pixel_Width        0
RAM                0
Screen_Height      0
Screen_Width       0
Talk_Time          0
3G                 0
Touch_Screen       0
WiFi               0
Price_Range        0
dtype: int64
```

2. EXPLORATORY DATA ANALYSIS -with data analysis

-Using EDA :Visualize fesatures, insight /observation from the data

- Missing Values
- All The Numerical Variables
- Distribution of the Numerical Variables
- Categorical Variables
- Cardinality of Categorical Variables
- Outliers

In [125…
```
sns.displot(data["Battery_Power"]) #displot always use for the continous data
plt.show()
```

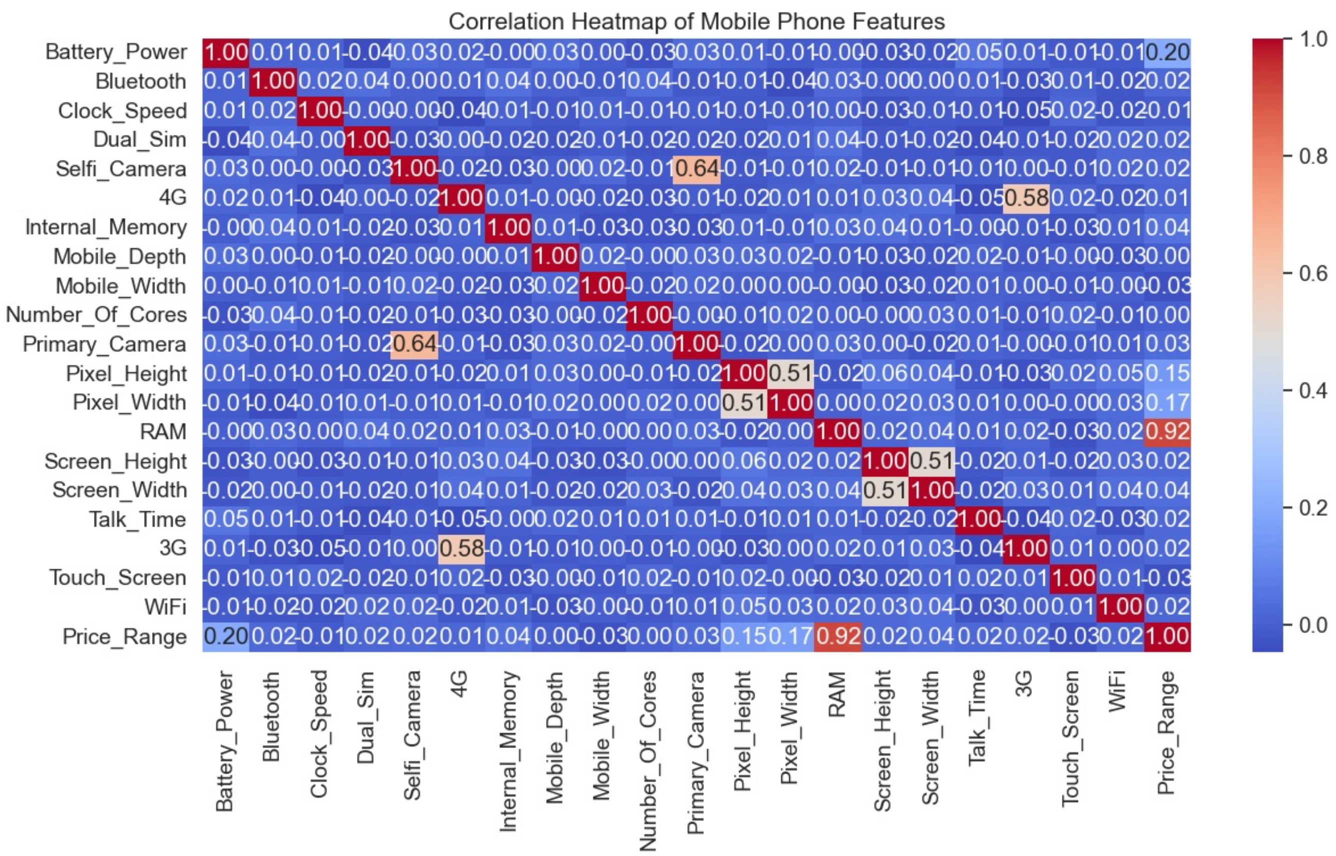


insight

- The range of battery_power are 600 to 2000
- The range of 600 mah battery are widely used
- The range of 1200 Mah battery are very less used

In [126…

```python
# Create a correlation heatmap
plt.figure(figsize=(14, 7))
sns.heatmap(data.corr(), annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Heatmap of Mobile Phone Features')
plt.show()
```



Insight

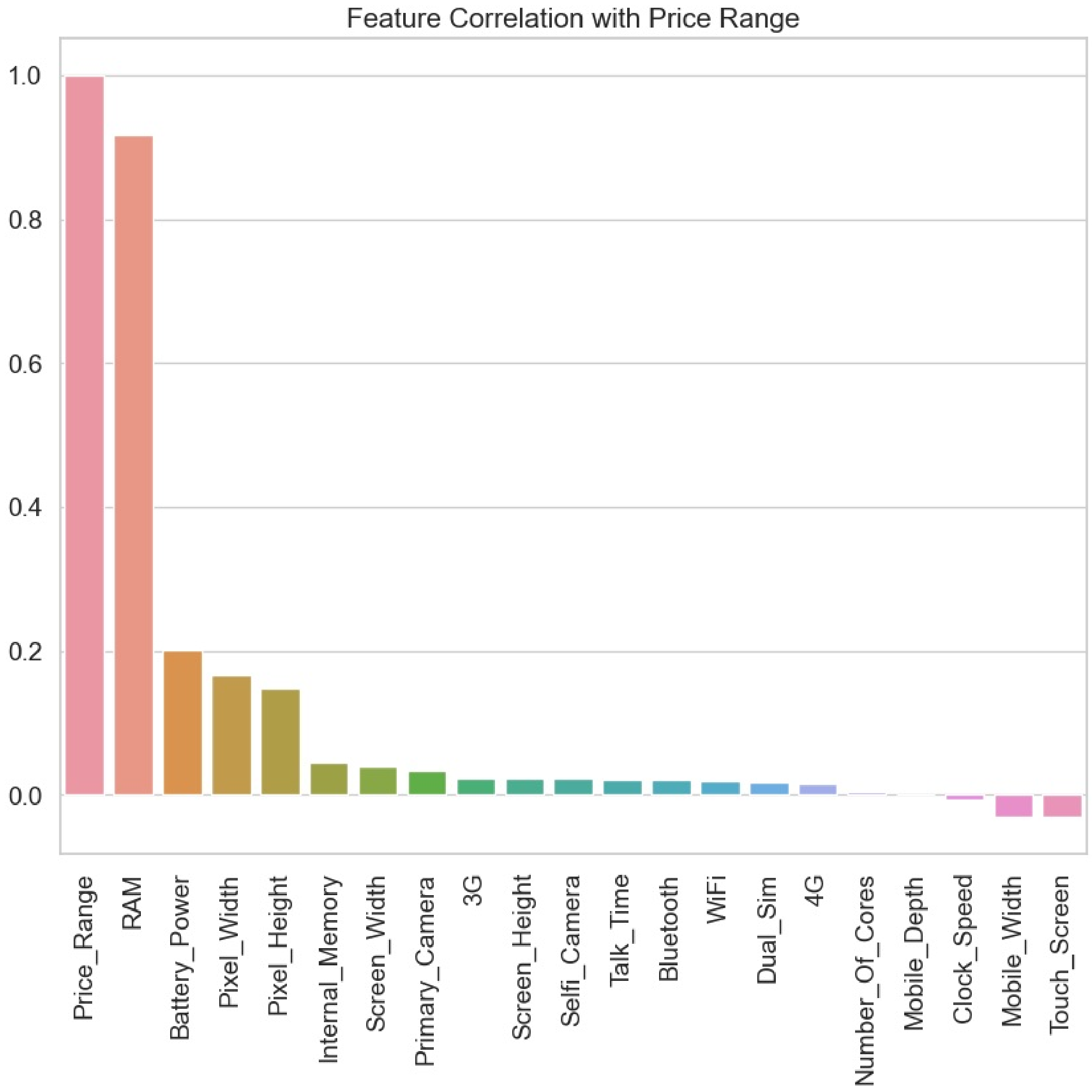
- There is high Correlation between the primary camera and selfi camera (Primary camera and selfi camera - 64%)
- 4G and 3G also have good correlation (4G and 3G : 58%)
- there is highest correlation between the RAM and Price range- 92%

In [127…

```
sns.set_style('whitegrid')# Identify numeric features correlated with 'price_range'
correlated_features = data.corr()['Price_Range'].sort_values(ascending=False)
```

In [128…

```
# bar plot for the correlation of numeric features with 'price_range'
plt.figure(figsize=(10, 8))
sns.barplot(x=correlated_features.index, y=correlated_features.values)
plt.xticks(rotation=90)
plt.title('Feature Correlation with Price Range')
plt.show()
```



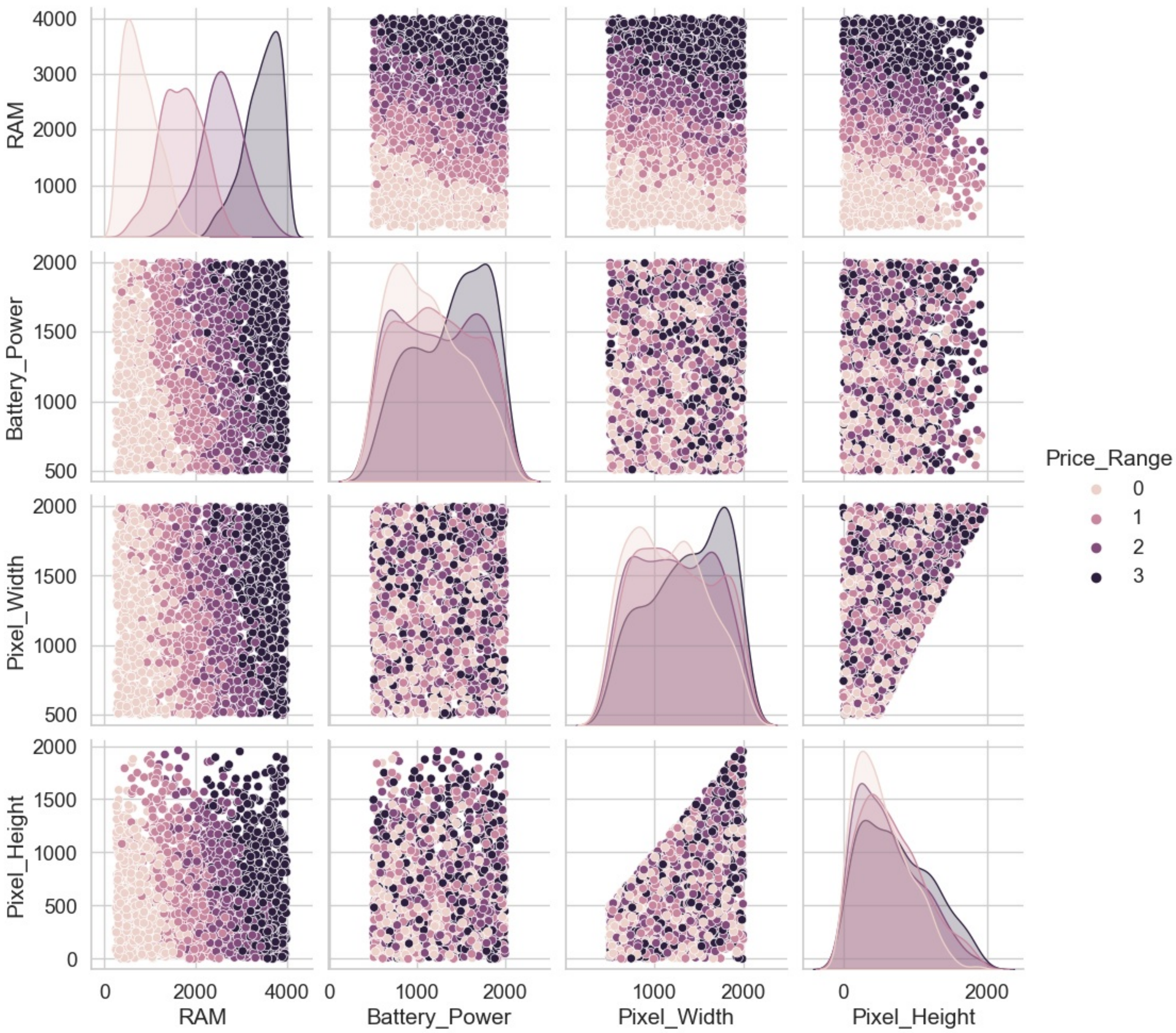
Insight

-Top 5

- This is graphical representation of how feature is correlated with price range.
- Price range,RAM,Battery Pixel_width,Pixle_hight this top5 feature is highly correlated
- Many more

In [129…

```
# Multivariate Analysis
# Pairplot for the most correlated features with 'price_range'
most_correlated_features = correlated_features.index[1:5] # Skip the first one as it is 'price_range' itself
sns.pairplot(data, vars=most_correlated_features, hue='Price_Range')
plt.show()
```



Data Pre-Processing

- Data preprocessing- is the process of cleaning and preparing the raw data to enable feature engineering.
- Feature Engineering covers various data engineering techniques such as adding/removing relevant features, handling missing data, encoding the data, handling categorical variables, etc
- handling missing values
- handling outliers
- drop duplicates
- handling categorical varaibles
- scaling

In [130…

```
#sum of missing data
data.isnull().sum().sort_values(ascending=False)#Data is clean
```

Out[130]:
```
Battery_Power      0
Pixel_Height       0
WiFi               0
Touch_Screen       0
3G                 0
Talk_Time          0
Screen_Width       0
Screen_Height      0
RAM                0
Pixel_Width        0
Primary_Camera     0
Bluetooth          0
Number_Of_Cores    0
Mobile_Width       0
Mobile_Depth       0
Internal_Memory    0
4G                 0
Selfi_Camera       0
Dual_Sim           0
Clock_Speed        0
Price_Range        0
dtype: int64
```

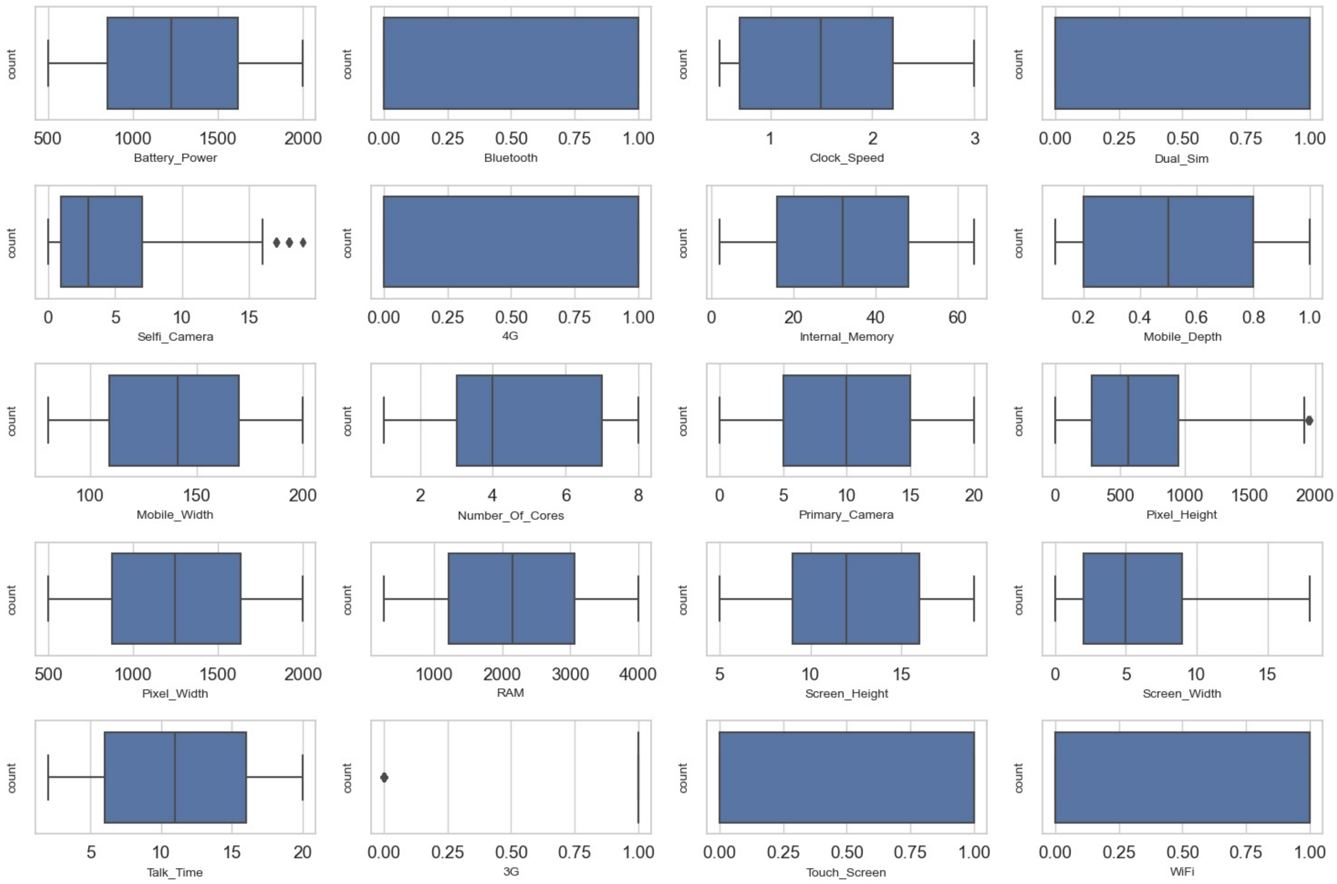
- No missing data

In [131…
```
data.duplicated().sum()#No duplicate value
```

Out[131]:
```
0
```

- No duplicates data

In [133…
```
#Check outlier
plt.figure(figsize=(15,10),facecolor="White")
plotnumber=1
for column in data.drop("Price_Range",axis=1):
    if plotnumber<21:
        ax=plt.subplot(5,4,plotnumber)
        sns.boxplot(x=data[column])
        plt.xlabel(column,fontsize=10)
        plt.ylabel("count",fontsize=10)
        plotnumber+=1
    plt.tight_layout()
```



Insight

- Out of all these feature we found out selfi camera and Pixel_Height have outlier

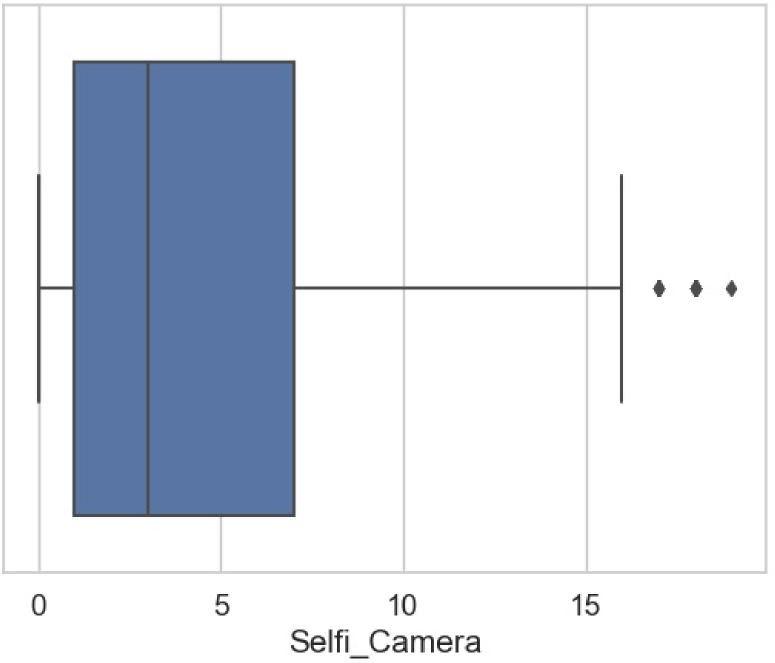
- Other wise all feature are good

OUTLIER DETECTION AND REMOVAL : MOST IMP

- Removing outliers is important step in data analysis. # However, while removing outliers in ML we should be careful, because we do not know if there are not any outliers in test set.
- checked the outliers then decide to drop outliers or handle the outliers.

In [134…
```
sns.boxplot(x="Selfi_Camera",data=data)#We found the outlier
```

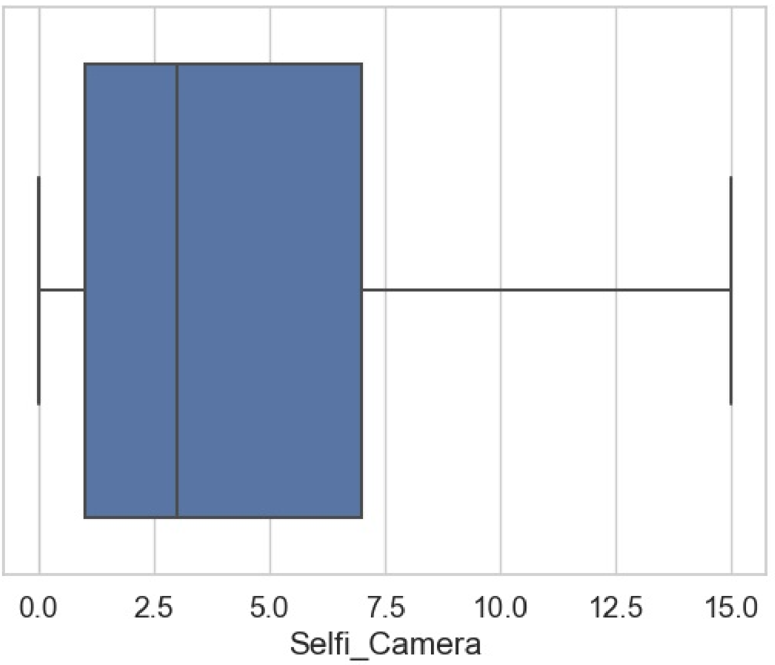
Out[134]:
```
<Axes: xlabel='Selfi_Camera'>
```



In [135…
```
data.drop(data.loc[data["Selfi_Camera"]>15].index,axis=0,inplace=True)# Remove these outlier
```

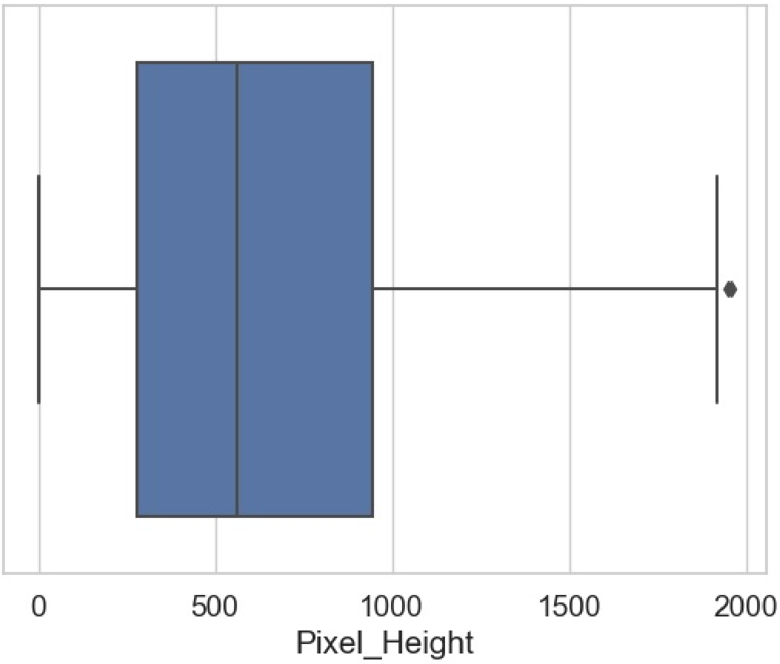
In [136…
```
sns.boxplot(x="Selfi_Camera",data=data)#No outlier present
```

Out[136]:
```
<Axes: xlabel='Selfi_Camera'>
```



In [137…
```
sns.boxplot(x="Pixel_Height",data=data)#We found the outlier
```

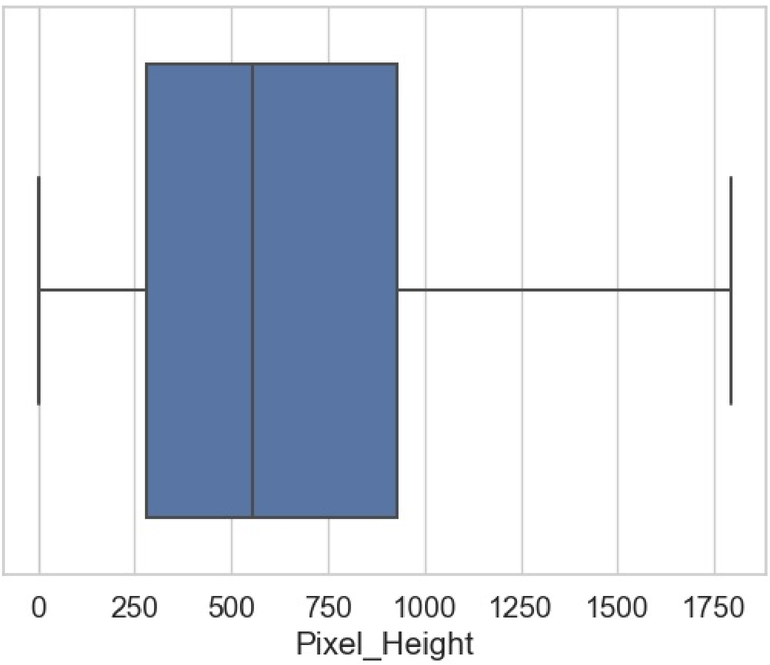
Out[137]:
```
<Axes: xlabel='Pixel_Height'>
```



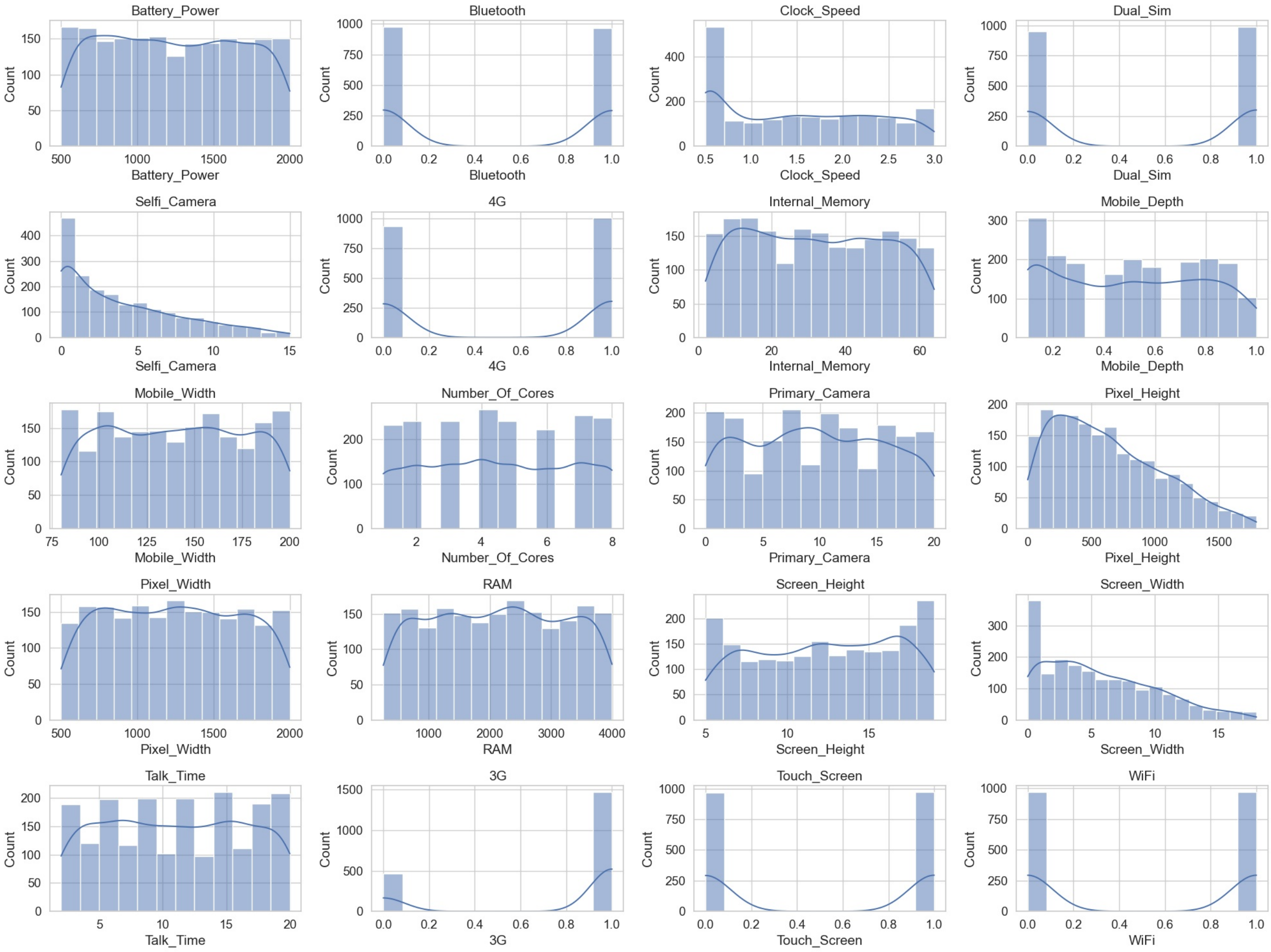
In [138…
```
data.drop(data.loc[data["Pixel_Height"]>1800].index,axis=0,inplace=True)# Remove these outlier
```

In [139…
```
sns.boxplot(x="Pixel_Height",data=data)#No outlier present
```

Out[139]:
```
<Axes: xlabel='Pixel_Height'>
```



In [140…
```
# Univariate Analysis
# Histograms for all numeric features
plt.figure(figsize=(20, 15))
for i, column in enumerate(data.drop('Price_Range', axis=1).columns, 1):
    plt.subplot(5, 4, i)
    sns.histplot(data[column], kde=True)
    plt.title(column)
plt.tight_layout()
plt.show()
```



Insight

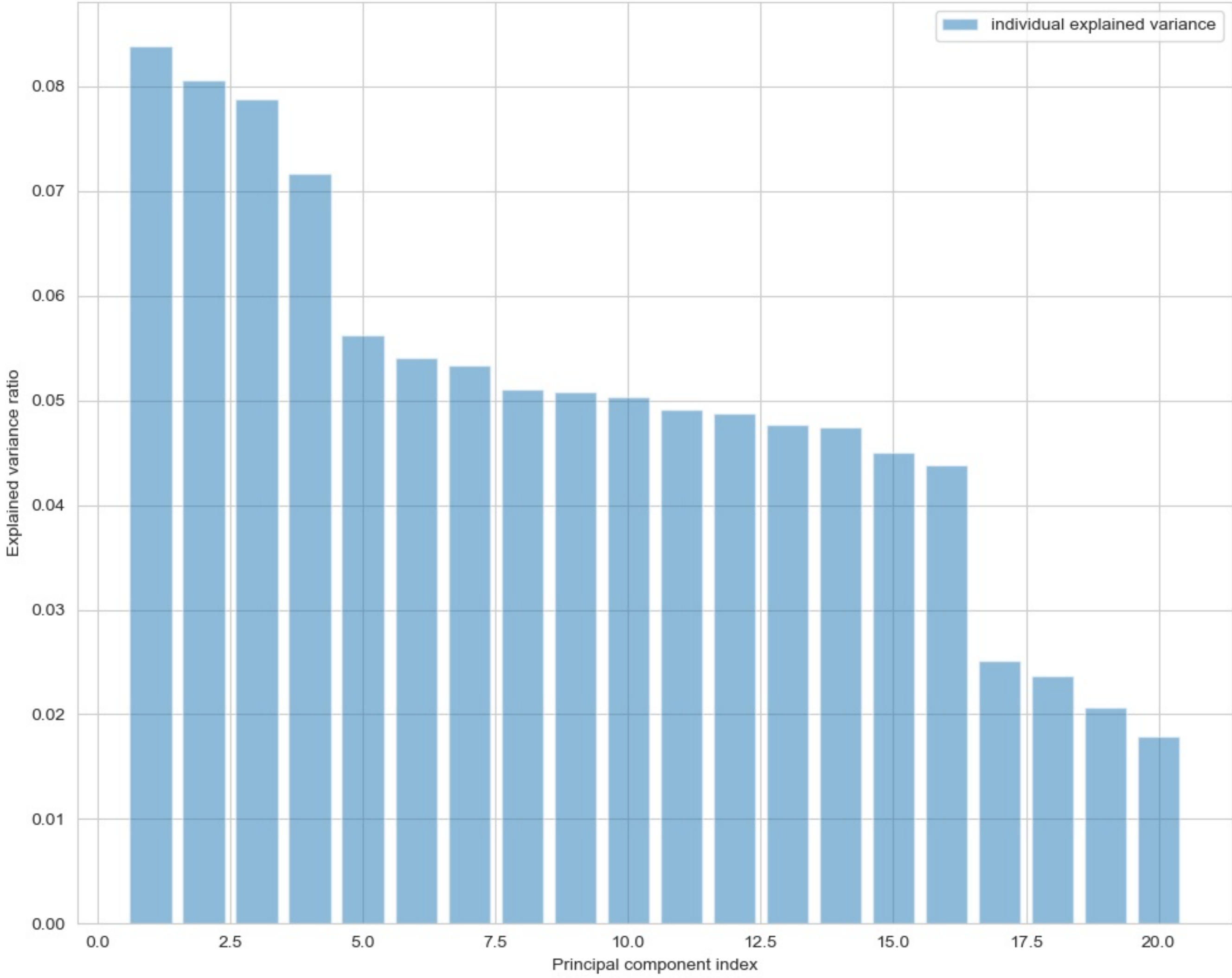
- Battery Power: Shows a fairly uniform distribution, indicating that battery capacity varies widely across the mobile phones.
- Bluetooth: Indicates a nearly balanced presence of Bluetooth capability across the dataset.
- Clock Speed: Suggests that most phones have lower clock speeds, with fewer phones having high clock speeds.
- Dual SIM: Shows that dual SIM functionality is quite common among the phones.
- Front Camera Megapixels: Reveals a right-skewed distribution, meaning most phones have lower front camera megapixels.
- 4G: Highlights that a significant number of phones support 4G.
- Internal Memory: Displays a wide distribution, suggesting varied internal storage options.
- Mobile Depth: Indicates a concentration of phones with slimmer profiles.
- Mobile Weight: Shows a broad distribution, implying a variety of phone weights.
- Number of Cores: Suggests that phones with 2 to 4 cores are most common, with fewer phones having higher core counts.
- Primary Camera Megapixels: Also right-skewed, similar to the front camera, with most phones having lower megapixels.
- Pixel Resolution Height and Width: Shows varied pixel resolutions, with a slight right skew indicating some phones have very high resolutions.
- RAM: Displays a wide range of RAM sizes, with a concentration at the lower end.
- Screen Height and Width: Indicates a variety of screen sizes, with a tendency towards larger screens.
- Talk Time: Shows a broad range of battery life as measured by talk time.
- 3G and 4G: Reflects the availability of 3G and 4G across the dataset, with a large number of phones supporting these technologies.
- Touch Screen: Shows that touch screen functionality is common.
- WiFi: Indicates that WiFi capability is also common among the phones.
- These distributions help in understanding the range and commonality of features in mobile phones, which can be crucial for market segmentation and targeting specific customer groups.

Feature Selection : PCA

In [319…

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(data_no_outliers.drop('Price_Range', axis=1))

# Initialize PCA with the correct number of components
```

```
pca = PCA(n_components=20)

# Fit PCA on the standardized data
pca.fit(X_scaled)

# Get the explained variance ratio
explained_variance = pca.explained_variance_ratio_

# Get the most important features according to the first principal component
most_important_features_indices = np.argsort(-pca.components_[0])
most_important_features = [data_no_outliers.drop('Price_Range', axis=1).columns[i] for i in most_important_feat

# Plot the explained variance
plt.figure(figsize=(10, 8))
plt.bar(range(1, 21), explained_variance, alpha=0.5, align='center', label='individual explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal component index')
plt.legend(loc='best')
plt.tight_layout()
plt.show()

# Print the most important features
print('Most important features for prediction according to PCA:')
for feature in most_important_features:
    print(feature)
```



```
Most important features for prediction according to PCA:
Selfi_Camera
Primary_Camera
Talk_Time
Mobile_Width
Battery_Power
Clock_Speed
Mobile_Depth
RAM
Bluetooth
Number_Of_Cores
Dual_Sim
Touch_Screen
WiFi
Internal_Memory
Pixel_Width
Pixel_Height
Screen_Height
Screen_Width

3G
4G
```

In [320…

```
# Calculate the percentage of variance explained by each feature
feature_importance = 100 * pca.explained_variance_ratio_ / np.sum(pca.explained_variance_ratio_)
# Create a DataFrame for feature importance
feature_importance_df = pd.DataFrame({'Feature': data_no_outliers.drop('Price_Range', axis=1).columns,
                                      'Importance': feature_importance})
feature_importance_df
```

Out[320]:

	Feature	Importance
0	Battery_Power	8.388593
1	Bluetooth	8.063694
2	Clock_Speed	7.884486
3	Dual_Sim	7.163236
4	Selfi_Camera	5.625021
5	4G	5.404908
6	Internal_Memory	5.335467
7	Mobile_Depth	5.108204
8	Mobile_Width	5.077484
9	Number_Of_Cores	5.031511
10	Primary_Camera	4.913596
11	Pixel_Height	4.873031
12	Pixel_Width	4.770415
13	RAM	4.748656
14	Screen_Height	4.505166
15	Screen_Width	4.380756
16	Talk_Time	2.511257
17	3G	2.367994
18	Touch_Screen	2.062059
19	WiFi	1.784470

- The table above lists the features of the mobile phones along with their respective importance percentages as determined by PCA.
- These percentages indicate how much of the variance in the dataset each feature explains, which is a proxy for their importance in predicting the price range.
- The feature battery_power is the most important explaining approximately 8.39% of the variance followed by (Bluetooth) at about 8.06%, and clock_speed at roughly 7.88%.
- The least important feature is wifi explaining about 1.78% of the variance

Model Creation

In [321…

```
x=data.drop("Price_Range",axis=1) #Independent Feature
y=data.Price_Range #dependent feature
```

In [322…

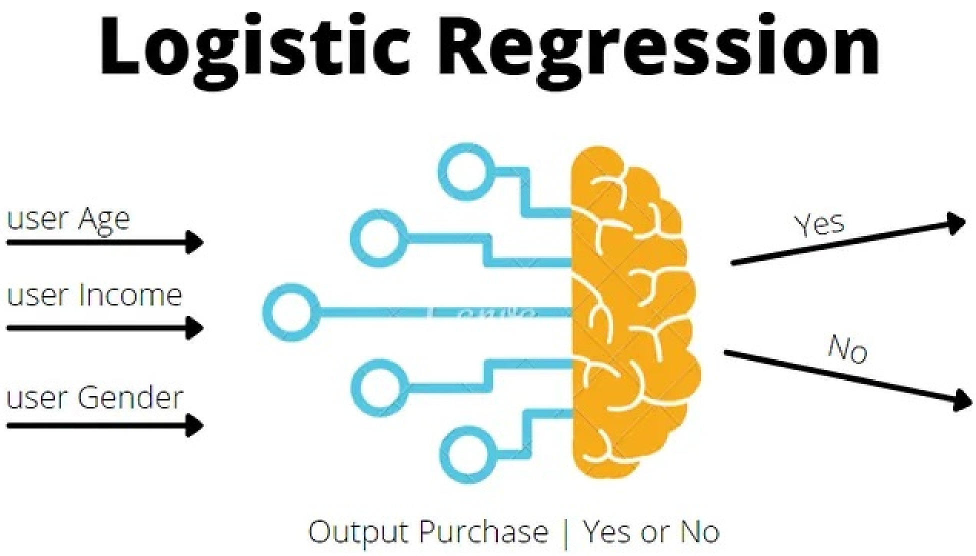
```
#Preparing Training and Testing data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

LogisticRegression

In [323…
```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(x_train,y_train)
```

Out[323]:
```
▾ LogisticRegression
LogisticRegression()
```

In [324…
```
Prediction_of_testdata=model.predict(x_test)#Test Prediction
x_train_pred=model.predict(x_train)#Tranning Prediction
```



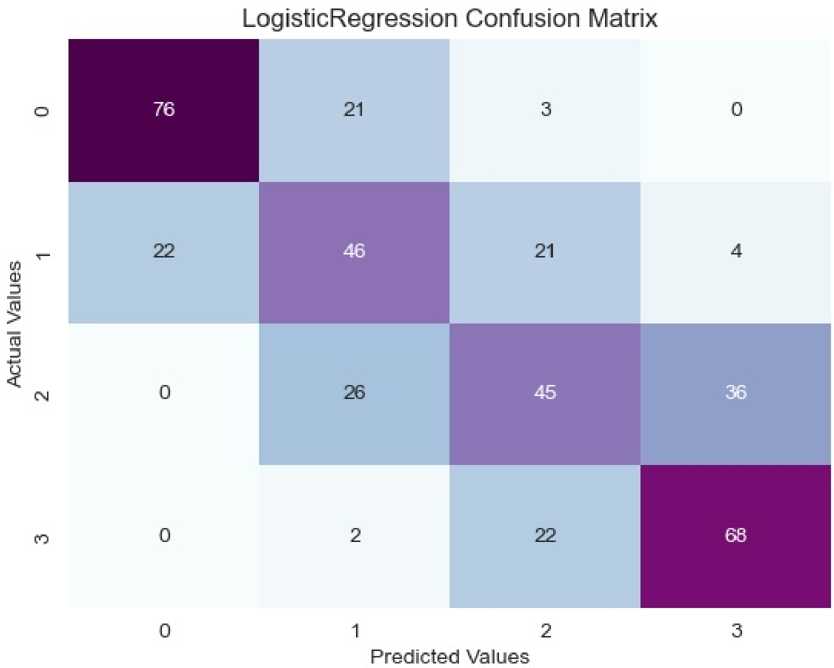
Evaluate the models

In [325…
```
print(accuracy_score(y_test,Prediction_of_testdata))#Test Acuracy
print(accuracy_score(y_train,x_train_pred))#Tranning Accuracy
```

```
0.5994897959183674
0.6455938697318008
```

In [326…
```
def my_confusion_matrix(y_test, Prediction_of_testdata, plt_title):
    cm=confusion_matrix(y_test, Prediction_of_testdata)
    print(classification_report(y_test, Prediction_of_testdata))
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, cmap='BuPu')
    plt.xlabel('Predicted Values')
    plt.ylabel('Actual Values')
    plt.title(plt_title)
    plt.show()
    return cm
print('LogisticRegression Accuracy Score: ',accuracy_score(y_test,Prediction_of_testdata))
cm_rfc=my_confusion_matrix(y_test, Prediction_of_testdata, 'LogisticRegression Confusion Matrix')
```

```
LogisticRegression Accuracy Score:  0.5994897959183674
              precision    recall  f1-score   support

           0       0.78      0.76      0.77       100
           1       0.48      0.49      0.49        93
           2       0.49      0.42      0.45       107
           3       0.63      0.74      0.68        92

    accuracy                           0.60       392
   macro avg       0.60      0.60      0.60       392
weighted avg       0.60      0.60      0.60       392
```



- LogisticRegression work well with Binary classification problem but we have multiclass classification problem that why we are not got good accuracy

DecisionTreeClassifier

In [327…

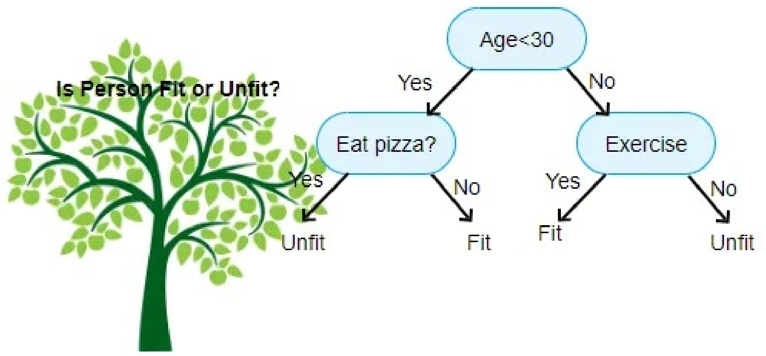
```
from sklearn.tree import DecisionTreeClassifier#importing decision tree from sklearn.tree
dt=DecisionTreeClassifier(criterion="entropy",max_depth=10,min_samples_leaf=1,min_samples_split=30,splitter="ra
dt.fit(x_train,y_train)#training the model
```

Out[327]:

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=10, min_samples_split=30,
                       splitter='random')
```

In [328…

```
Prediction_of_DT=dt.predict(x_test)#Test Prediction
x_train_preDT=model.predict(x_train)#Traning Prediction
```



Evaluate the models

In [329…

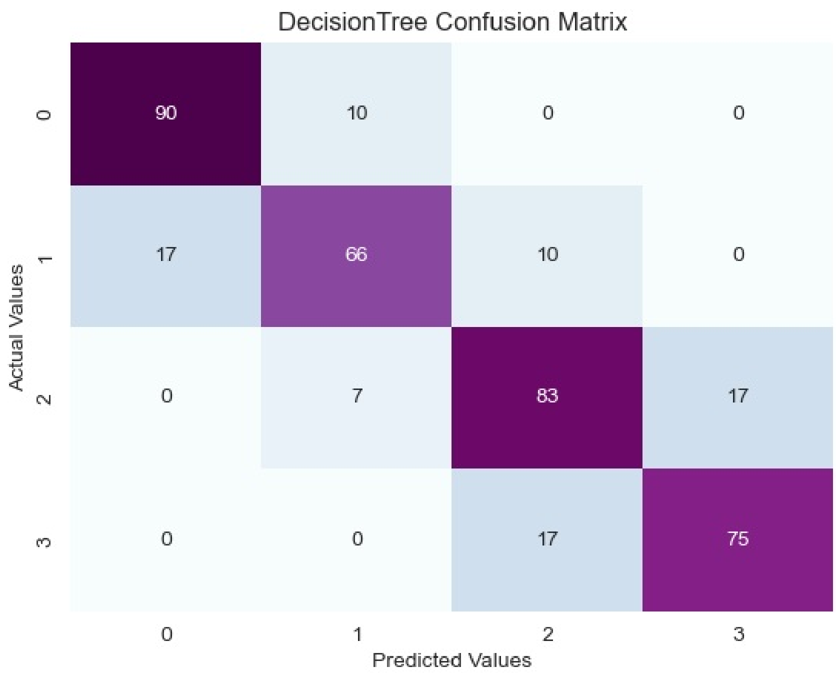
```
print(accuracy_score(y_test,Prediction_of_DT))#Test Accuracy
print(accuracy_score(y_train,x_train_preDT))#Traning Accuracy
```

```
0.8010204081632653
0.6455938697318008
```

In [330…

```python
def my_confusion_matrix(y_test, Prediction_of_DT, plt_title):
    cm=confusion_matrix(y_test, Prediction_of_DT)
    print(classification_report(y_test, Prediction_of_DT))
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, cmap='BuPu')
    plt.xlabel('Predicted Values')
    plt.ylabel('Actual Values')
    plt.title(plt_title)
    plt.show()
    return cm
print(' DecisionTree Classifier Accuracy Score: ',accuracy_score(y_test,Prediction_of_DT))
cm_rfc=my_confusion_matrix(y_test, Prediction_of_DT, 'DecisionTree Confusion Matrix')
```

```
 DecisionTree Classifier Accuracy Score:  0.8010204081632653
              precision    recall  f1-score   support

           0       0.84      0.90      0.87       100
           1       0.80      0.71      0.75        93
           2       0.75      0.78      0.76       107
           3       0.82      0.82      0.82        92

    accuracy                           0.80       392
   macro avg       0.80      0.80      0.80       392
weighted avg       0.80      0.80      0.80       392
```



- We are got good accuracy when we used DecisionTree beacaused decisionTree select best Leaf Nodes
- And train models that why we got 80% Accuracy

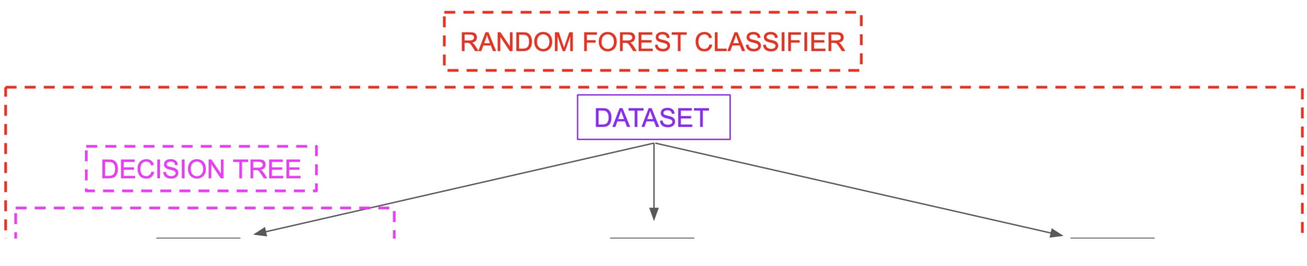
Random Forest Classifier

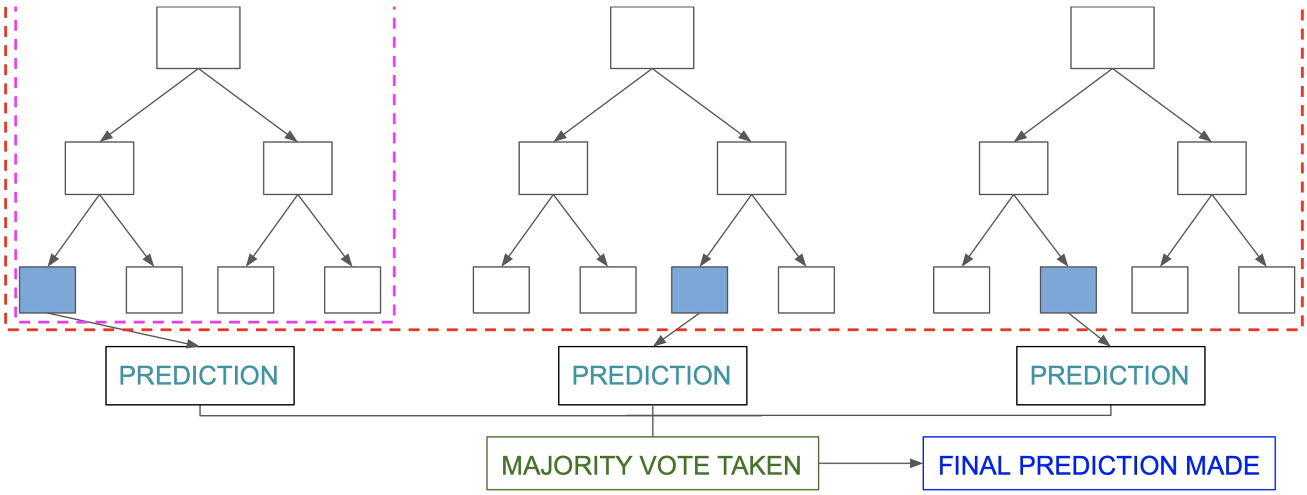
In [331…

```python
#building the model
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(bootstrap= True,
                           max_depth= 7,
                           max_features= 15,
                           min_samples_leaf= 3,
                           min_samples_split= 10,
                           n_estimators= 200,
                           random_state=7)
```

In [332…

```python
#Now, we do the training and prediction.
rfc.fit(x_train, y_train)
y_pred_rfc=rfc.predict(x_test)#Test prediction
x_train_preRFC=model.predict(x_train)#Traning Prediction
```





Evaluate the models

In [333…

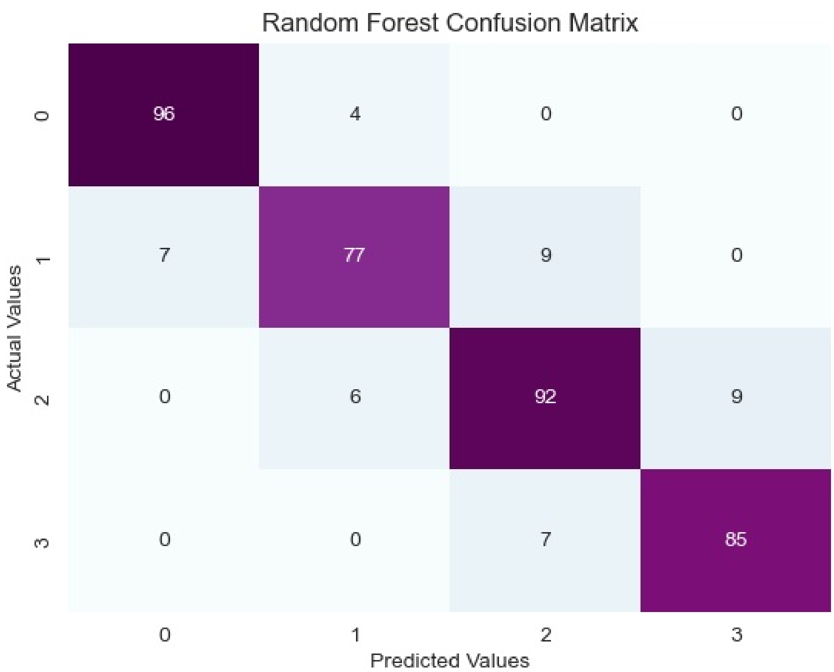
```
print(accuracy_score(y_test,y_pred_rfc))#Test accuracy
print(accuracy_score(y_train,x_train_preRFC))#Traning accuracy
```

```
0.8928571428571429
0.6455938697318008
```

In [334…

```
def my_confusion_matrix(y_test, y_pred_rfc, plt_title):
    cm=confusion_matrix(y_test, y_pred_rfc)
    print(classification_report(y_test, y_pred_rfc))
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, cmap='BuPu')
    plt.xlabel('Predicted Values')
    plt.ylabel('Actual Values')
    plt.title(plt_title)
    plt.show()
    return cm
print(' Random Forest Classifier Accuracy Score: ',accuracy_score(y_test,y_pred_rfc))
cm_rfc=my_confusion_matrix(y_test, y_pred_rfc, 'Random Forest Confusion Matrix')
```

```
 Random Forest Classifier Accuracy Score:  0.8928571428571429
              precision    recall  f1-score   support

           0       0.93      0.96      0.95       100
           1       0.89      0.83      0.86        93
           2       0.85      0.86      0.86       107
           3       0.90      0.92      0.91        92

    accuracy                           0.89       392
   macro avg       0.89      0.89      0.89       392
weighted avg       0.89      0.89      0.89       392
```



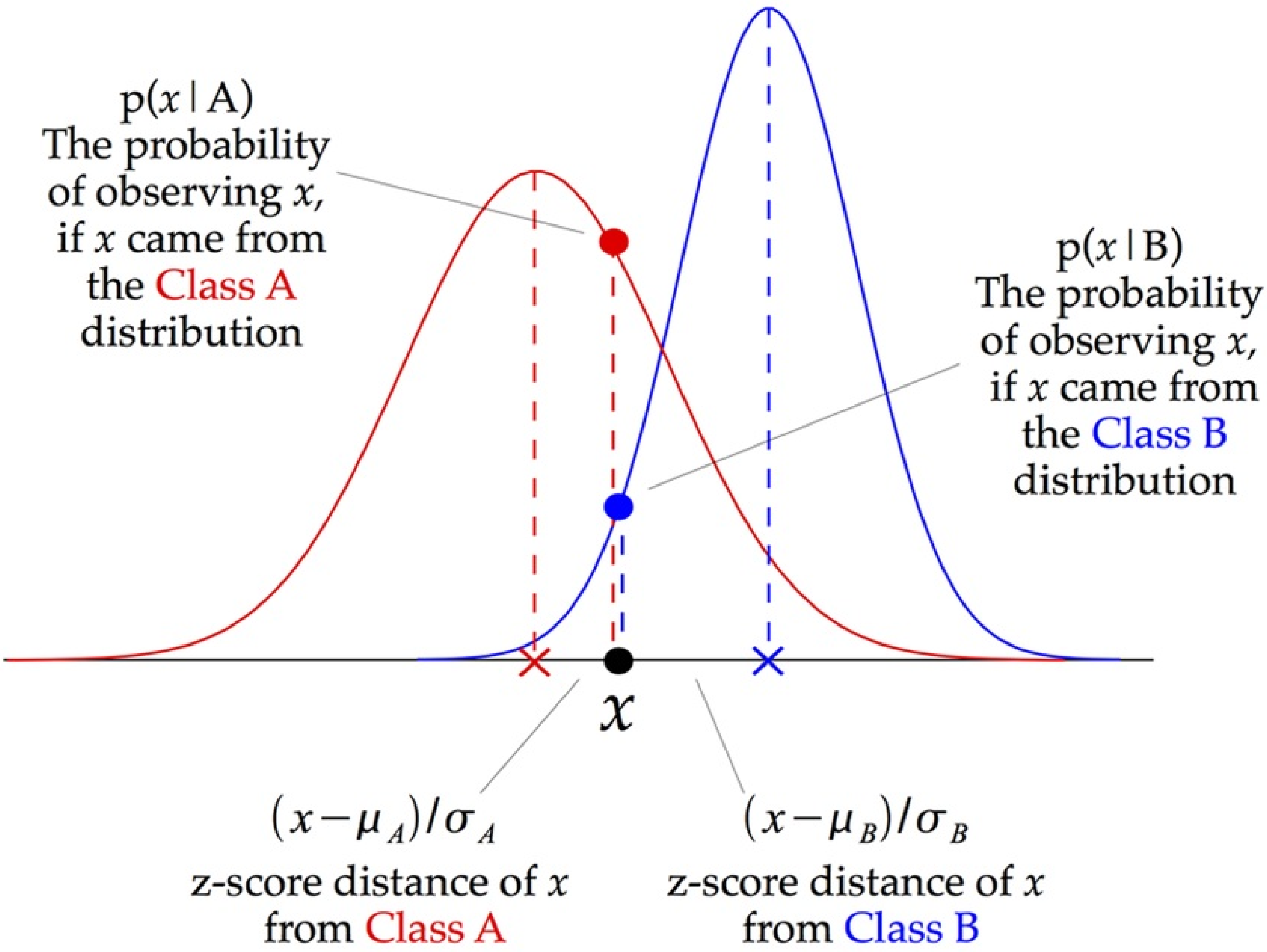
- RandomForestClassifier used bagging technique to improve the accuracy of model
- They create multiple Decision Tree and find their prediction
- Out top that take majoraity of vote and gives final prediction model

Naive Bayes

In [335…
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)
```

Out[335]:
```
▾ GaussianNB
GaussianNB()
```

In [336…
```
y_pred_gnb=gnb.predict(x_test)#Test Prediction
x_train_preGNB=model.predict(x_train)#Traning Prediction
```



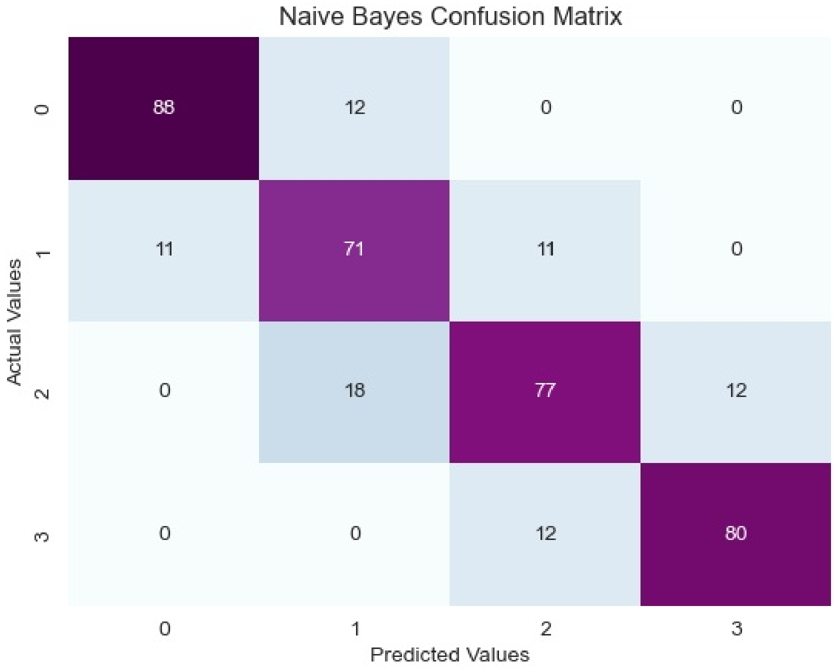
Evaluate the models

In [337…
```
print(accuracy_score(y_test,y_pred_gnb))#Test Accuracy
print(accuracy_score(y_train,x_train_preGNB))#Traning accuracy
```

```
0.8061224489795918
0.6455938697318008
```

In [338…
```
def my_confusion_matrix(y_test, y_pred_gnb, plt_title):
    cm=confusion_matrix(y_test, y_pred_gnb)
    print(classification_report(y_test, y_pred_gnb))
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, cmap='BuPu')
    plt.xlabel('Predicted Values')
    plt.ylabel('Actual Values')
    plt.title(plt_title)
    plt.show()
    return cm
print(' Naive Bayes Accuracy Score: ',accuracy_score(y_test,y_pred_gnb))
cm_rfc=my_confusion_matrix(y_test, y_pred_gnb, 'Naive Bayes Confusion Matrix')
```

```
Naive Bayes Accuracy Score:  0.8061224489795918
              precision    recall  f1-score   support

           0       0.89      0.88      0.88       100
           1       0.70      0.76      0.73        93
           2       0.77      0.72      0.74       107
           3       0.87      0.87      0.87        92

    accuracy                           0.81       392
   macro avg       0.81      0.81      0.81       392
weighted avg       0.81      0.81      0.81       392
```



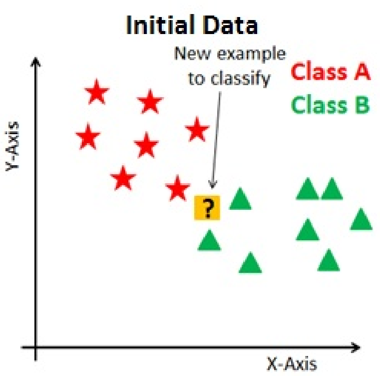
- Naive Bayes is a supervised machine learning algorithm that uses Bayes' theorem to calculate the probability of a class label given some features.
- Naive Bayes work well with Classification Problem and we got 80% accuracy

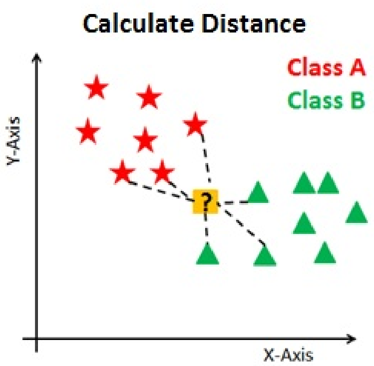
KNN Classifier

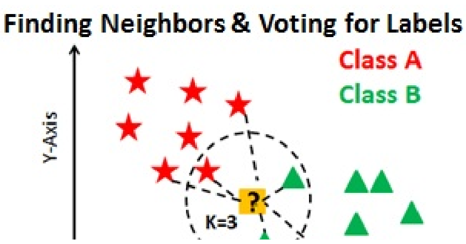
In [339…
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3,leaf_size=25)
knn.fit(x_train, y_train)
```

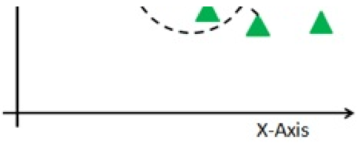
Out[339]:
```
▾            KNeighborsClassifier
KNeighborsClassifier(leaf_size=25, n_neighbors=3)
```

In [340…
```
y_pred_knn=knn.predict(x_test)#Test prediction
x_train_preKNN=model.predict(x_train)#Traning Prediction
```









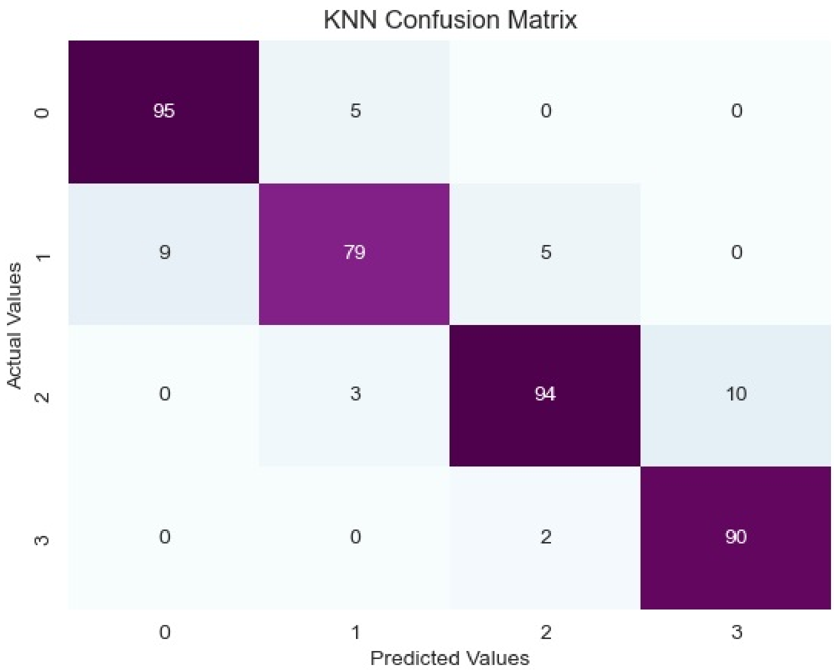
Evaluate the models

In [341…
```python
print(accuracy_score(y_test,y_pred_knn))#Test Accuracy
print(accuracy_score(y_train,x_train_preKNN))#Traning Accuracy
```

```
0.9132653061224489
0.6455938697318008
```

In [342…
```python
def my_confusion_matrix(y_test, y_pred_knn, plt_title):
    cm=confusion_matrix(y_test, y_pred_knn)
    print(classification_report(y_test, y_pred_knn))
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, cmap='BuPu')
    plt.xlabel('Predicted Values')
    plt.ylabel('Actual Values')
    plt.title(plt_title)
    plt.show()
    return cm
print(' KNN Classifier Accuracy Score: ',accuracy_score(y_test,y_pred_knn))
cm_rfc=my_confusion_matrix(y_test, y_pred_knn, 'KNN Confusion Matrix')
```

```
 KNN Classifier Accuracy Score:  0.9132653061224489
              precision    recall  f1-score   support

           0       0.91      0.95      0.93       100
           1       0.91      0.85      0.88        93
           2       0.93      0.88      0.90       107
           3       0.90      0.98      0.94        92

    accuracy                           0.91       392
   macro avg       0.91      0.91      0.91       392
weighted avg       0.91      0.91      0.91       392
```



- KNN stands for K-Nearest Neighbors, a supervised machine learning algorithm that can be used for both classification and regression problems. It works by finding the K most similar data points in the training set to a new data point, and then assigning it the label or value based on the majority vote or average of the K neighbors.
- KNN is also a lazy learner algorithm
- KNN is a non-parametric method

SVM Classifier

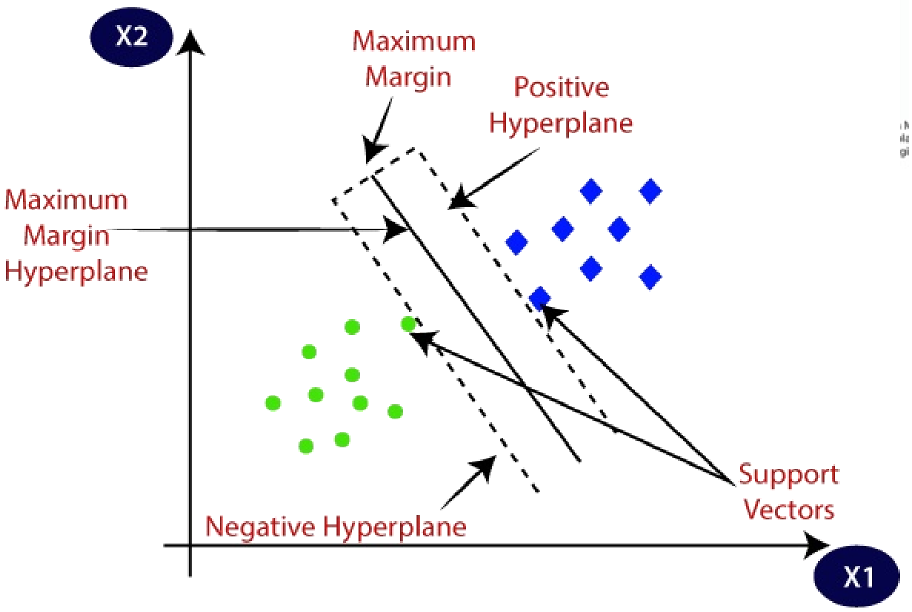
In [343…
```python
from sklearn import svm
svm_clf = svm.SVC(decision_function_shape='ovo')
svm_clf.fit(x_train, y_train)
```

Out[343]:

```
▾ SVC
SVC(decision_function_shape='ovo')
```

In [344…

```
y_pred_svm=svm_clf.predict(x_test)#Test Prediction
x_train_preSVM=model.predict(x_train)#Traning Prediction
```



Evaluate the models

In [345…

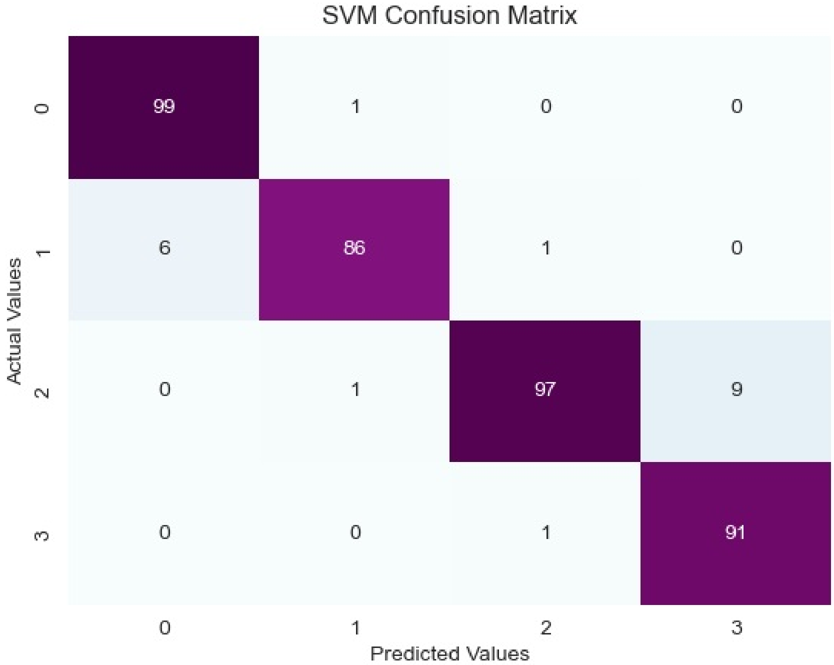
```
print(accuracy_score(y_test,y_pred_svm))#Test Accuracy
print(accuracy_score(y_train,x_train_preSVM))#Traning Accuracy
```

```
0.951530612244898
0.6455938697318008
```

In [346…

```
def my_confusion_matrix(y_test, y_pred_svm, plt_title):
    cm=confusion_matrix(y_test, y_pred_svm)
    print(classification_report(y_test, y_pred_svm))
    sns.heatmap(cm, annot=True, fmt='g', cbar=False, cmap='BuPu')
    plt.xlabel('Predicted Values')
    plt.ylabel('Actual Values')
    plt.title(plt_title)
    plt.show()
    return cm
print(' SVM Classifier Accuracy Score: ',accuracy_score(y_test,y_pred_svm))
cm_rfc=my_confusion_matrix(y_test, y_pred_svm, 'SVM Confusion Matrix')
```

```
 SVM Classifier Accuracy Score:  0.951530612244898
              precision    recall  f1-score   support

           0       0.94      0.99      0.97       100
           1       0.98      0.92      0.95        93
           2       0.98      0.91      0.94       107
           3       0.91      0.99      0.95        92

    accuracy                           0.95       392
   macro avg       0.95      0.95      0.95       392
weighted avg       0.95      0.95      0.95       392
```



- SVM stands for Support Vector Machine, a supervised machine learning algorithm that can be used for both classification and regression problems. SVM works by finding the optimal hyperplane that separates the data points in different classes with the maximum margin.
- Out of all models we are got height accuracy 95% becaused they used marginal distance technique- Eigen vector, Eigen value and draw hyperplane

Final Conclusion

- All over project we used so many technique For Example- find relation between various feature ,remove outlier,check missing value,Perform EDA,find best feature using PCA,model tranning model evaluation etc..
- We got good accuracy with two model 1) SVM Classifier 2) KNN Classifier
- SVM have 95% Accuracy so this is best model for production
- like that we soveld cellphone price prediction problem

```
In [ ]:
```

Loading [MathJax]/extensions/Safe.js