Mobile Price Classification

Problem Statement

Exploratory Data Analysis

Feature Engineering

Classification

Conclusion

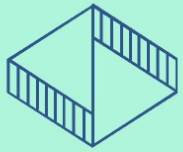
Ashwini Dharmarajan 002953107
Srikar Veldandi 001563236
Shruthi A 002112912
Sharanya Maryada 002921192

Introduction

With this fast moving world and constant development in technology, it’s tough for a commoner to catch up with the current trends. Our project is built to solve the problem for a common person who doesn’t understand the pricing of mobile phones. It is really important for every person to understand the features and the pricing of mobile phones before they invest their money in them.

Dataset

Additional Info



Null Values

Dataset

For this project we used a dataset from Kaggle - https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification. This dataset provides data with 21 features. We use the price_range label as the target value.

In [2]:
```
#Importing the dataset
df_train = pd.read_csv('train.csv')
```

In [3]:
```
#Shows the first 5 records of the dataset
df_train.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	thr
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	

5 rows × 21 columns

Classify mobiles based on the price range

Utilized by organizations who produce mobile devices to forecast pricing ranges for mobile devices with different characteristics and comprehend competition.

Used by customers of mobiles to determine whether the features they want fall within a certain price range.

Additional Information

This dataset is a total of 2000 records with 21 features including the target

This is a datatype of all features, consisting of either int or float

General info about the dataset

In [10]:
```
1 df_train.info()
2 #This prints the general information about the dataset.The dataset has 2000 entries. It has valu
3 #or float
```

```
<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   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
```

Treating Null Values

1. Missing data will influence how you deal with filling in the missing values.

2. Missing data creates imbalanced observations, cause biased estimates, and in extreme cases, can even lead to invalid conclusions.

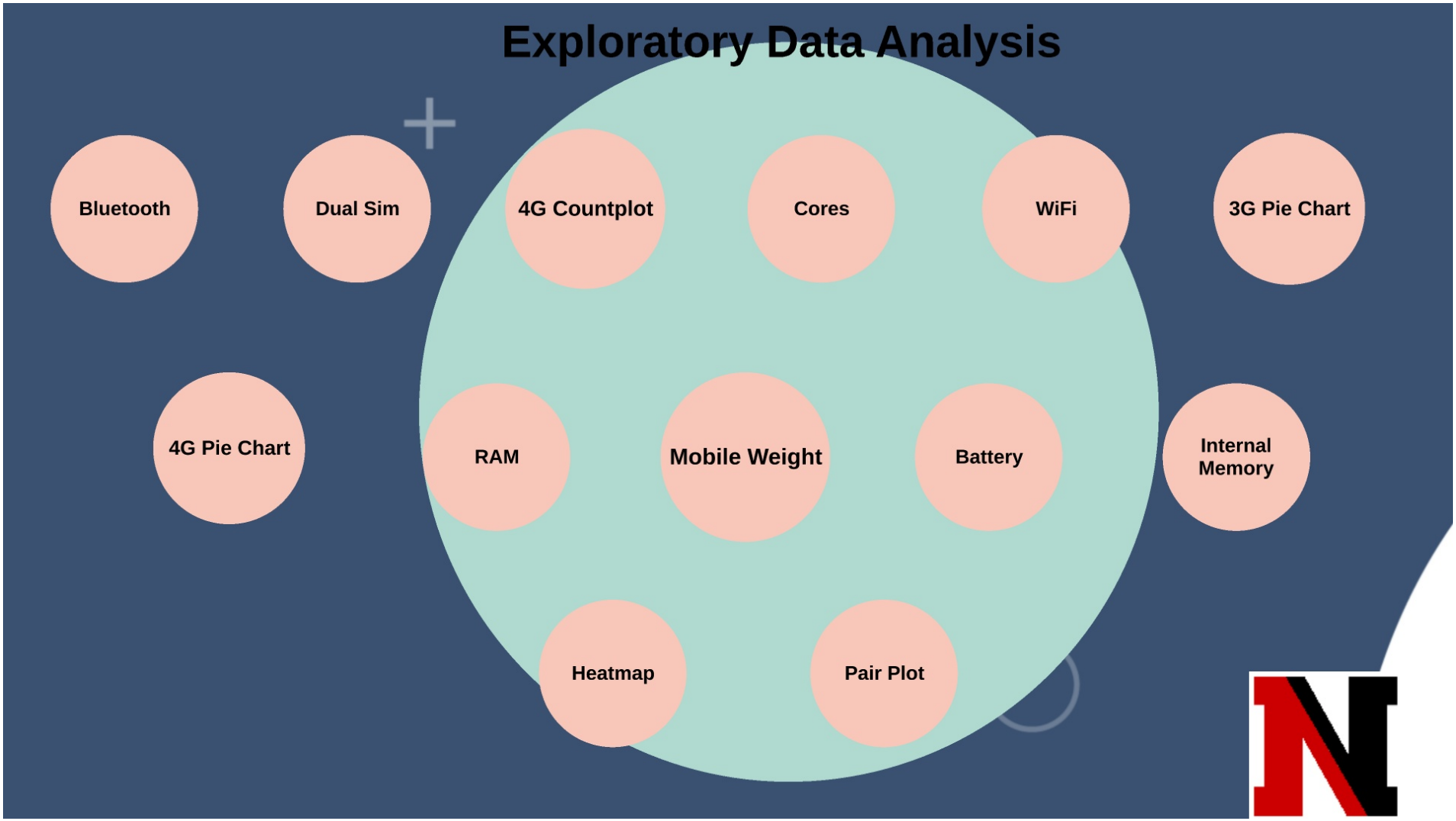
Another way of detecting missing values if any. The result shows that there are no missing/null values and therefore does not need further cleaning

In [11]:
```
1
2 df_train.isnull().sum()
3
```

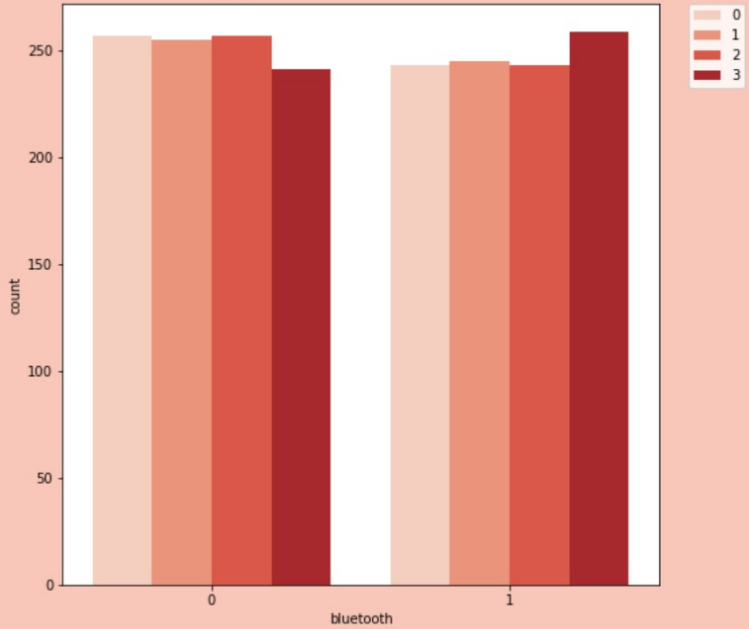
Out[11]:

battery_power	0
bluetooth	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

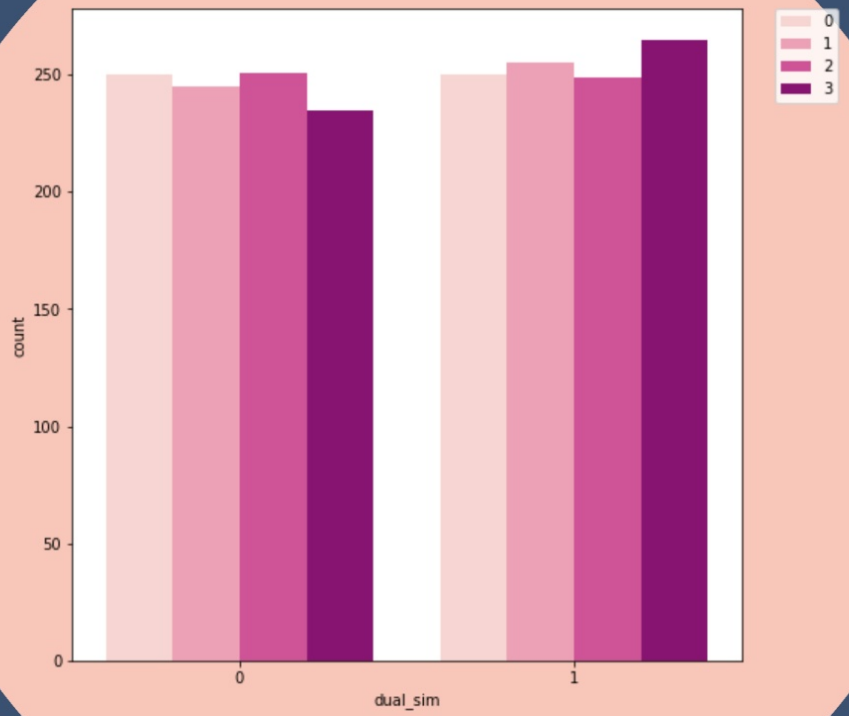

Exploratory Data Analysis
Bluetooth
Dual Sim
4G Countplot
Cores
WiFi
3G Pie Chart
4G Pie Chart
RAM
Mobile Weight
Battery
Internal Memory
Heatmap
Pair Plot

Count plot for Bluetooth



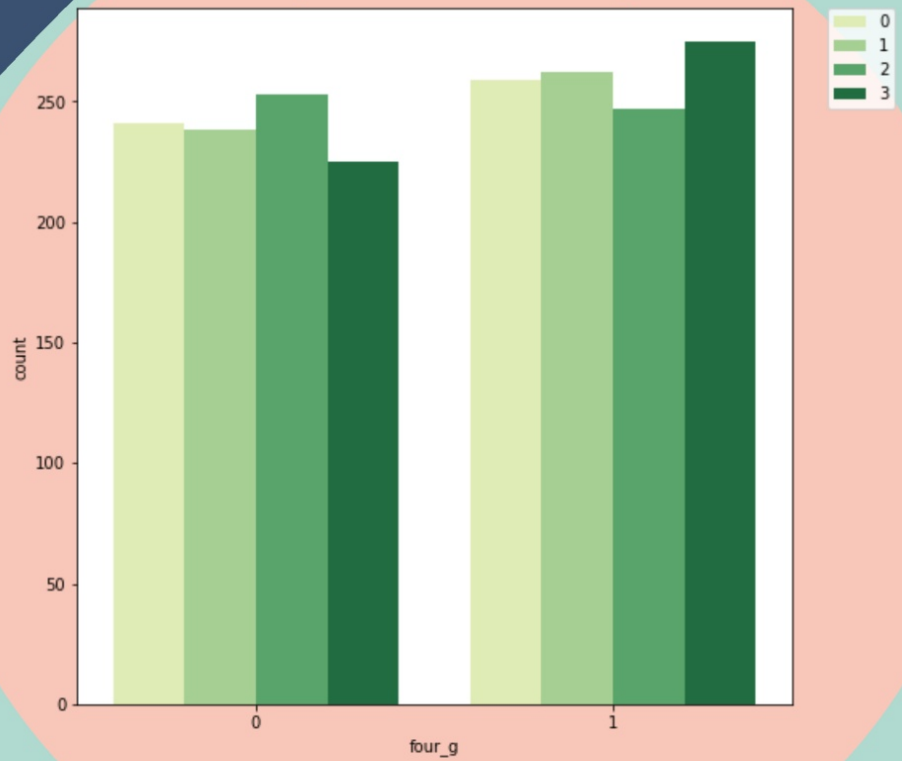
Count of mobiles is almost same irrespective of bluetooth status

Count plot for Dual Sim



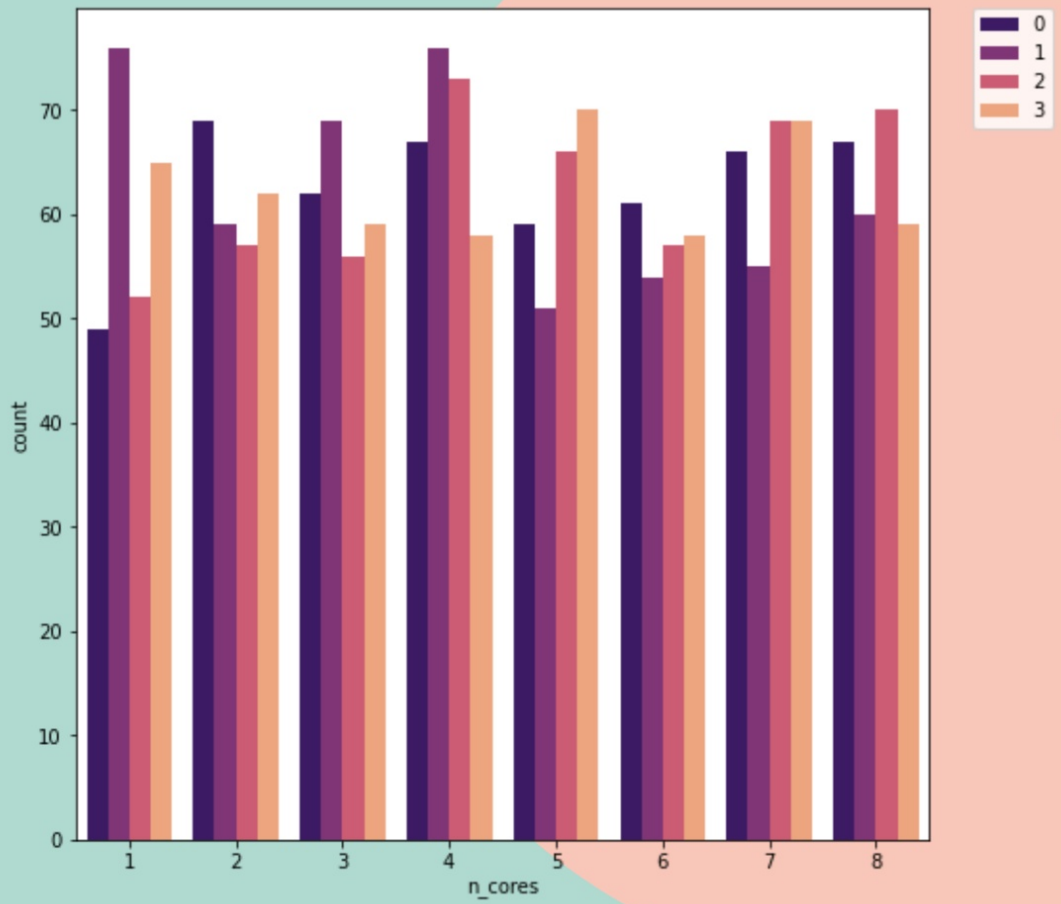
Count of Dual sim phones slightly on the higher price range

Count plot for 4G phones



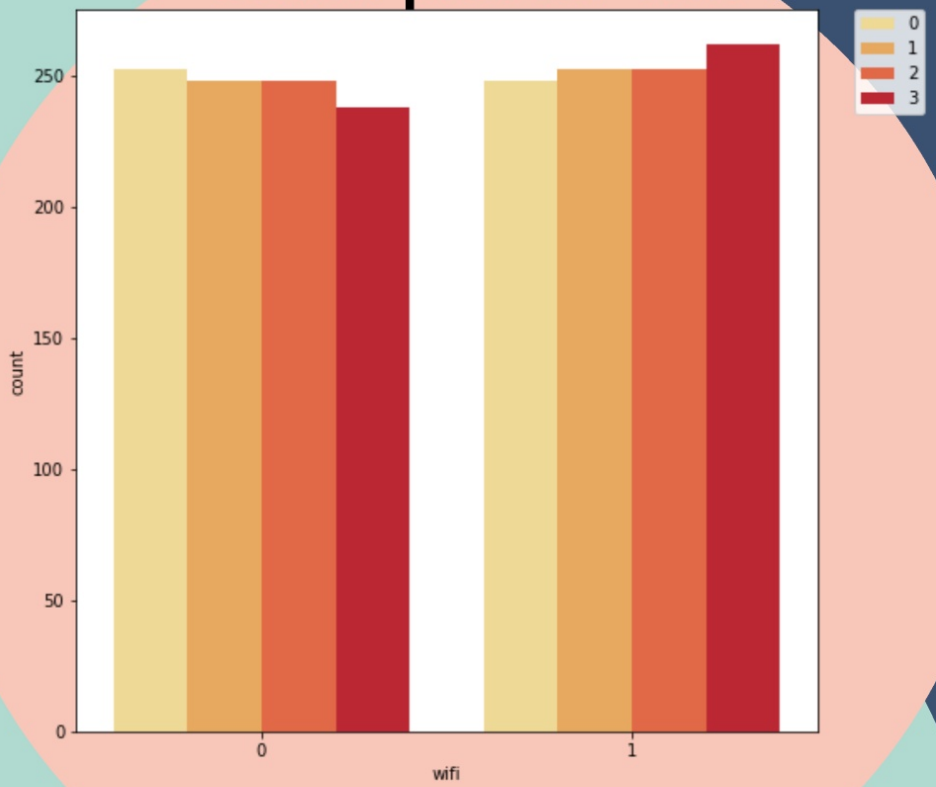
4G phones are a bit more in higher price range

Count plot for Cores



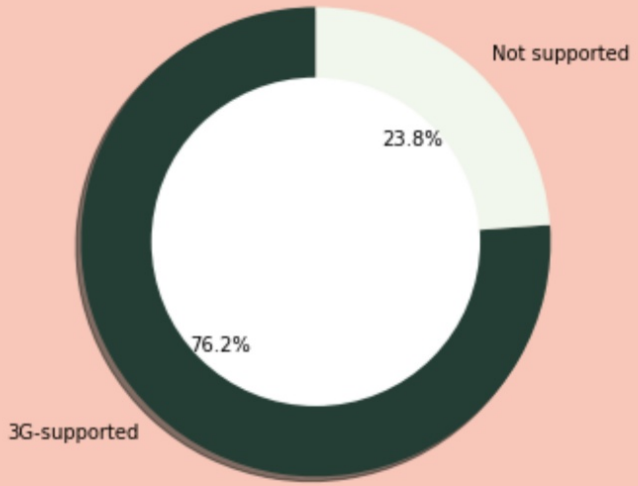
We can not make any inferences in their relationship

Count plot for Wifi

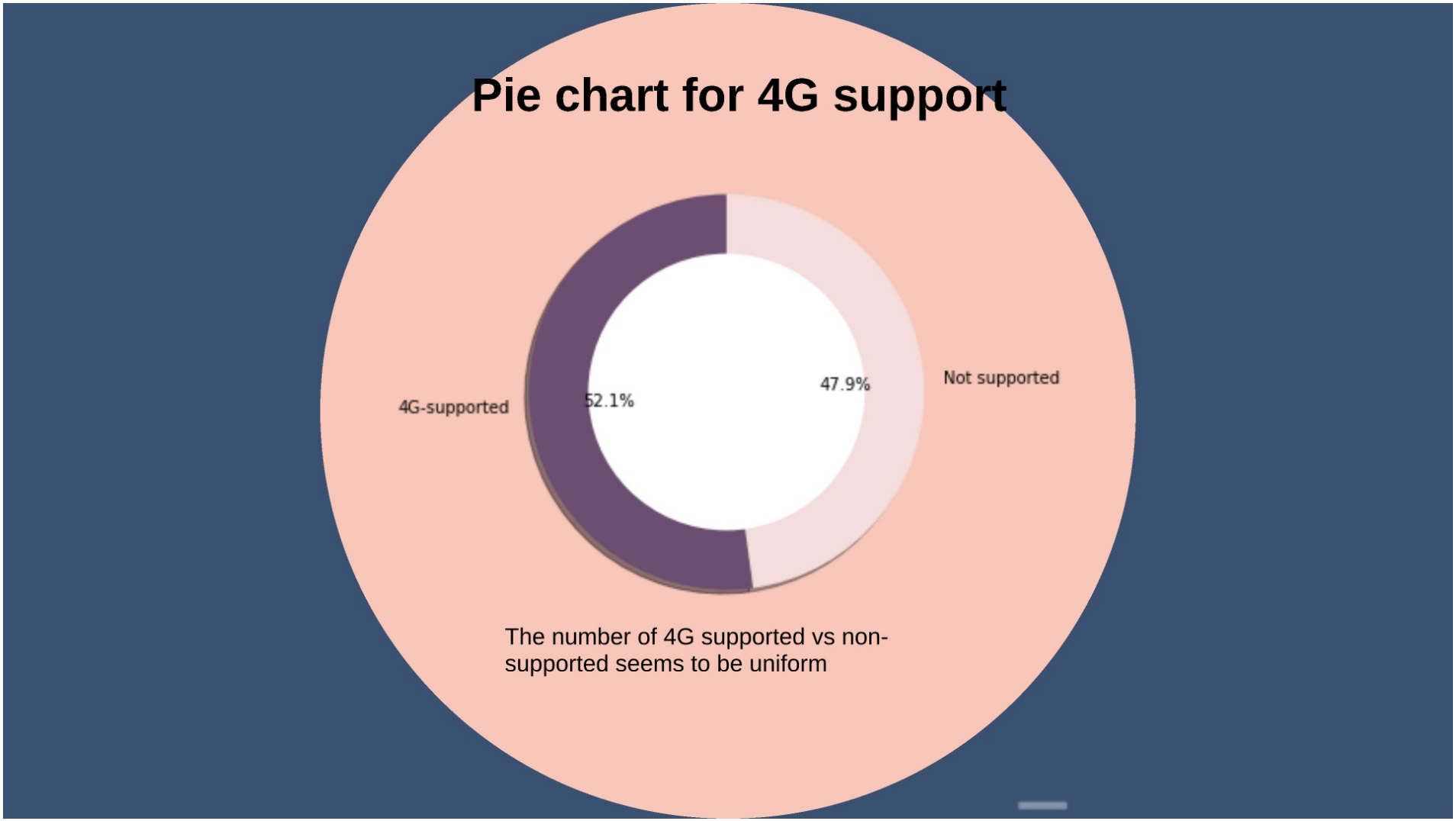


It can be inferred that phones with WiFi are more in the higher price range

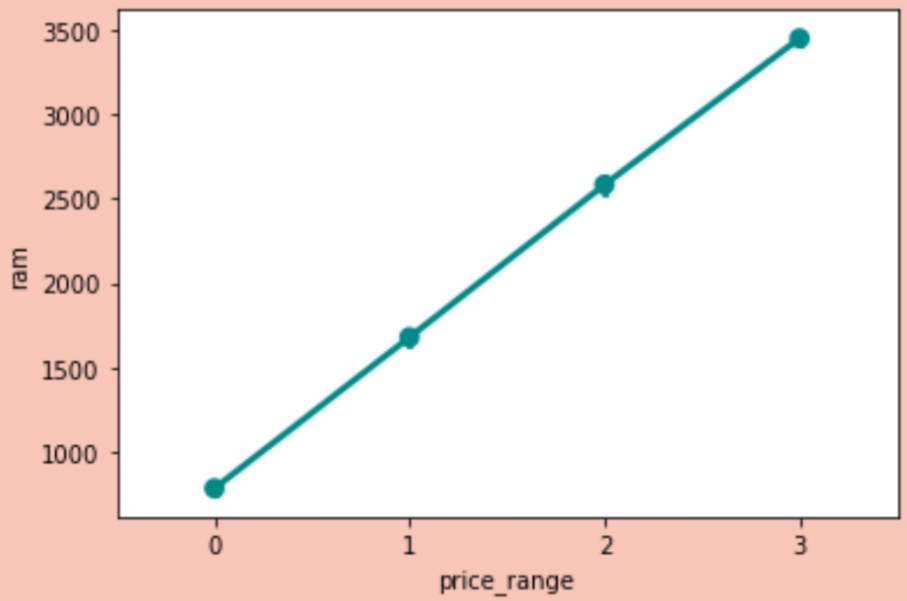
Pie Chart for 3G support



A large number of phones in our dataset has 3G support

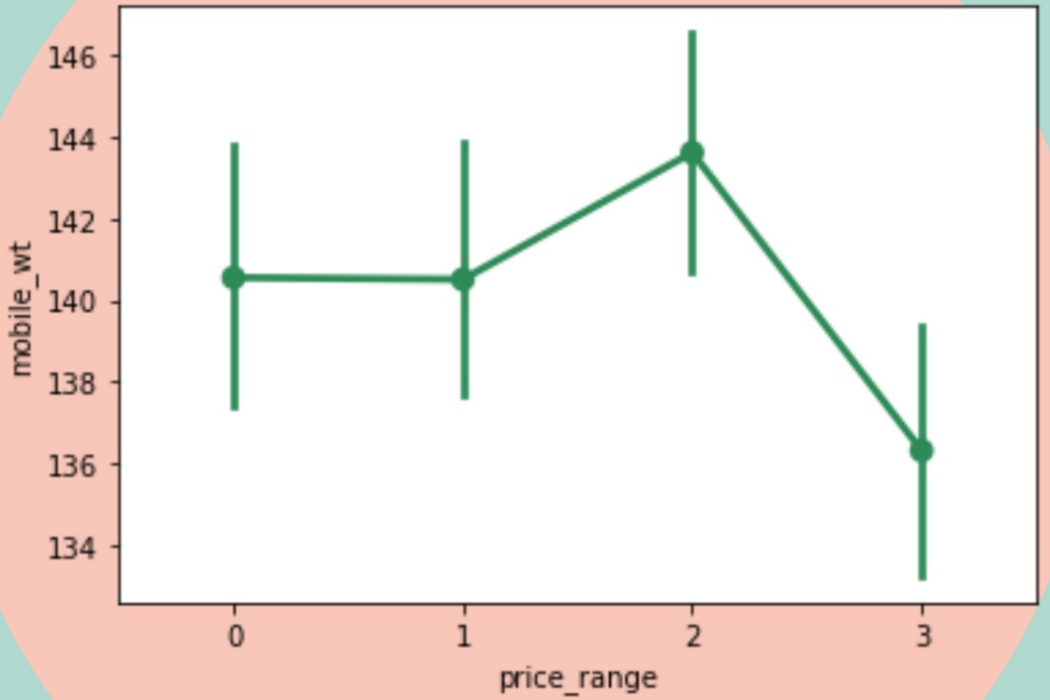

Pie chart for 4G support
4G-supported
52.1%
47.9%
Not supported
The number of 4G supported vs non-supported seems to be uniform

Relationship between Price Range and RAM



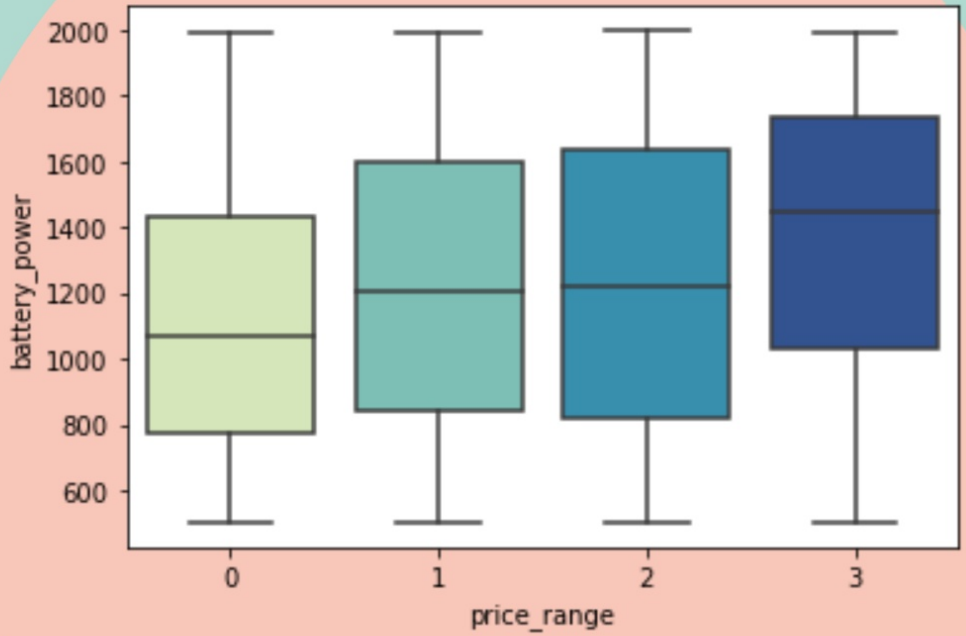
RAM has an strong linear relationship with price

Relationship between Price Range and Mobile Weight



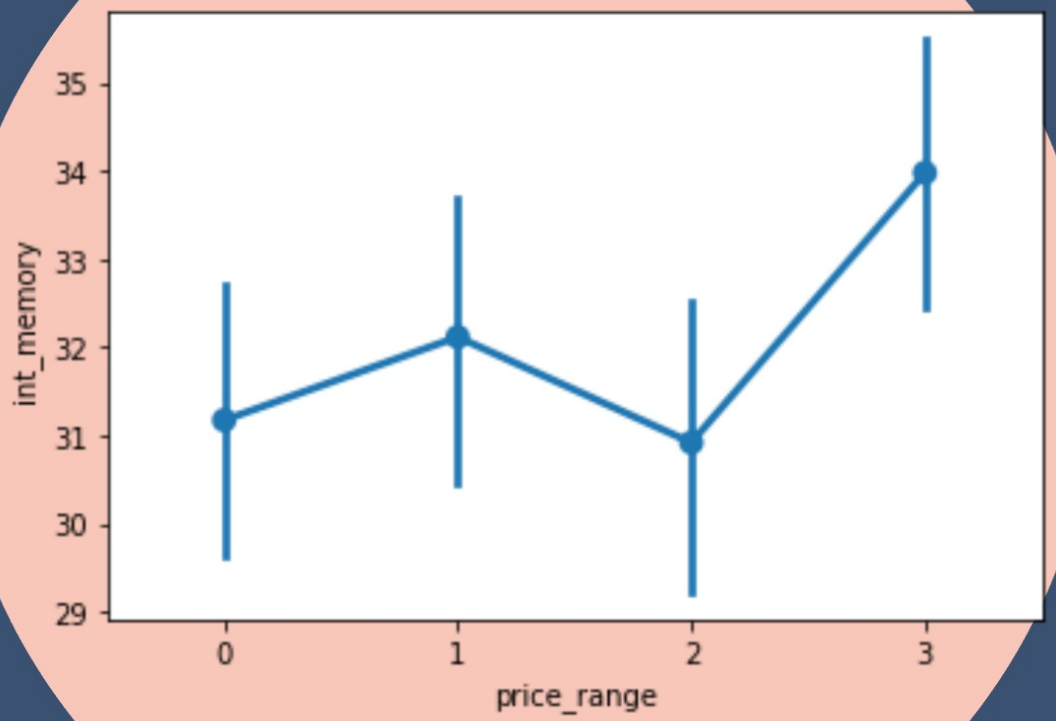
We see that low and medium range phones have similar weights. The weight for high price phones are higher. The costliest phones ie mobiles in range 3 have the lowest weight

Relationship between Battery and Price Range

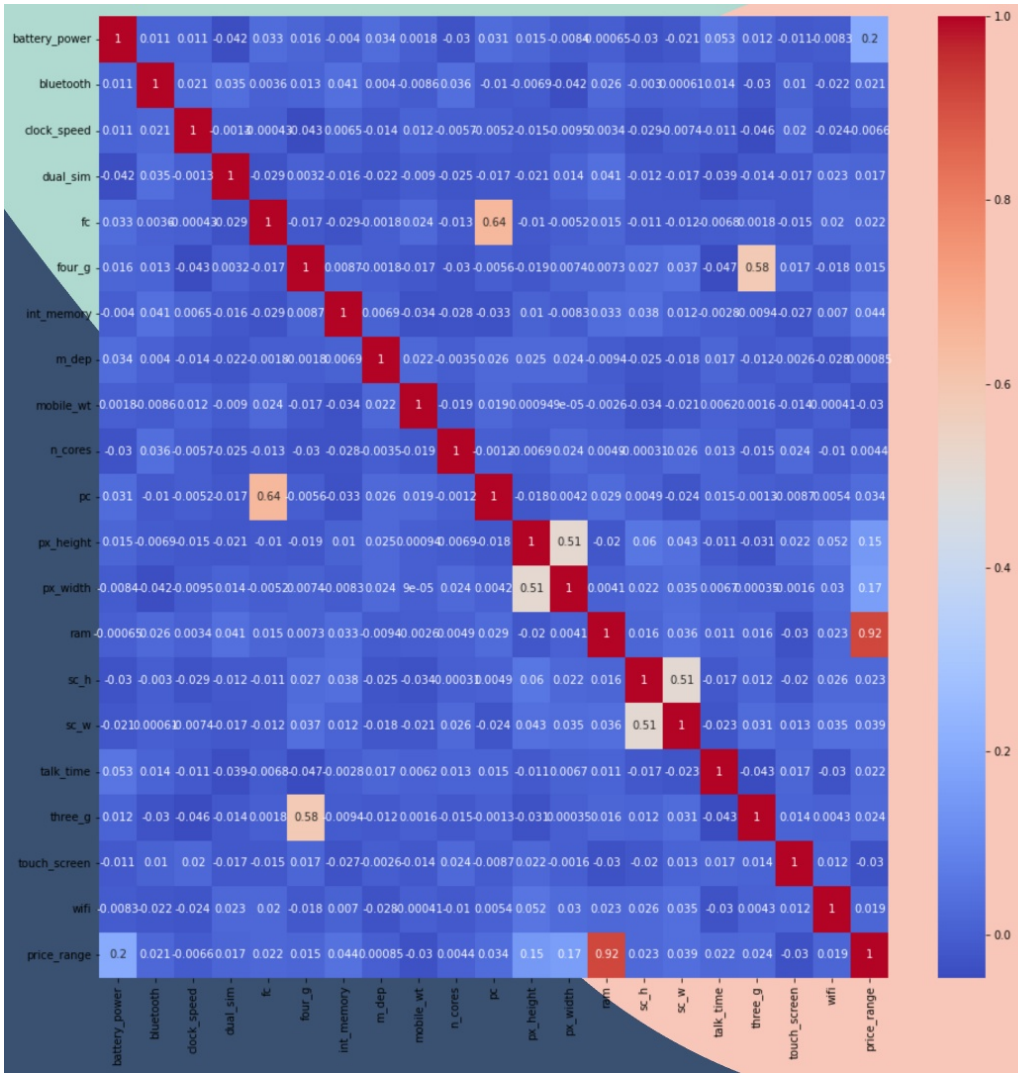


We see that as the price range increases that battery power also increases

Relationship between Price Range and Internal Memory



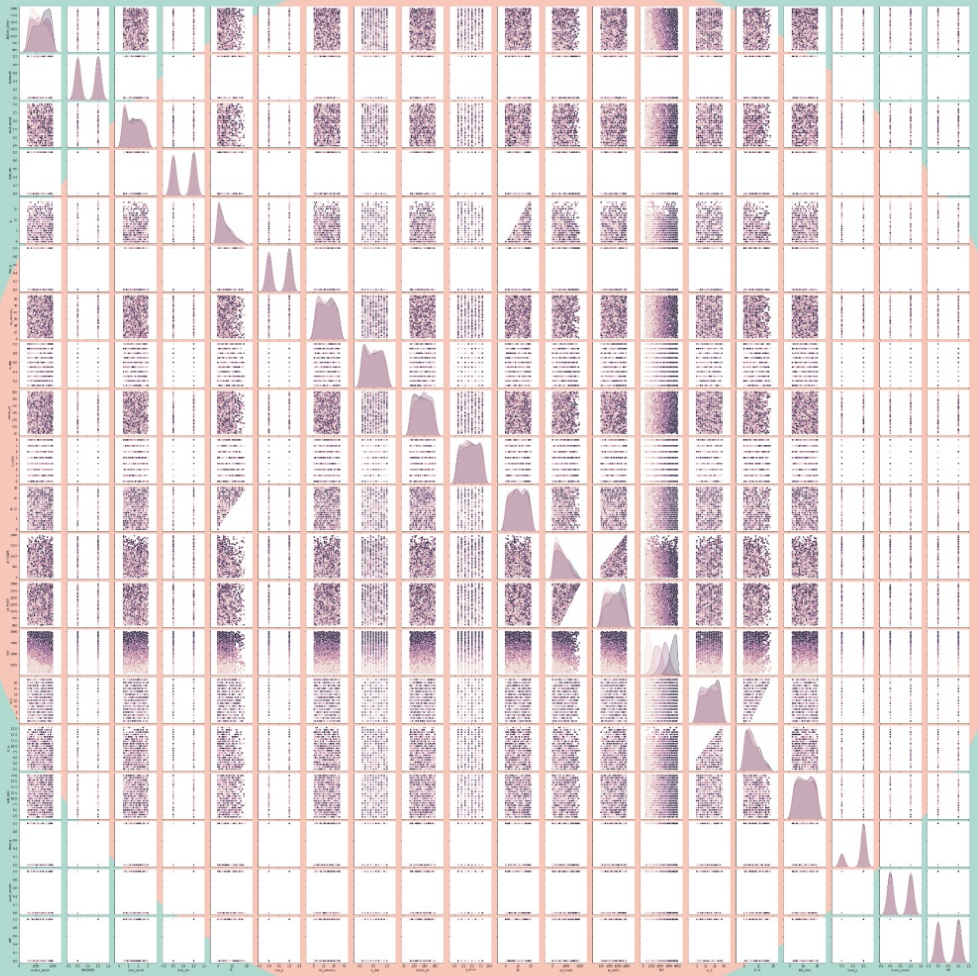
Price Range has a general upward trend except in price range 2. The costliest mobiles have the highest internal memory

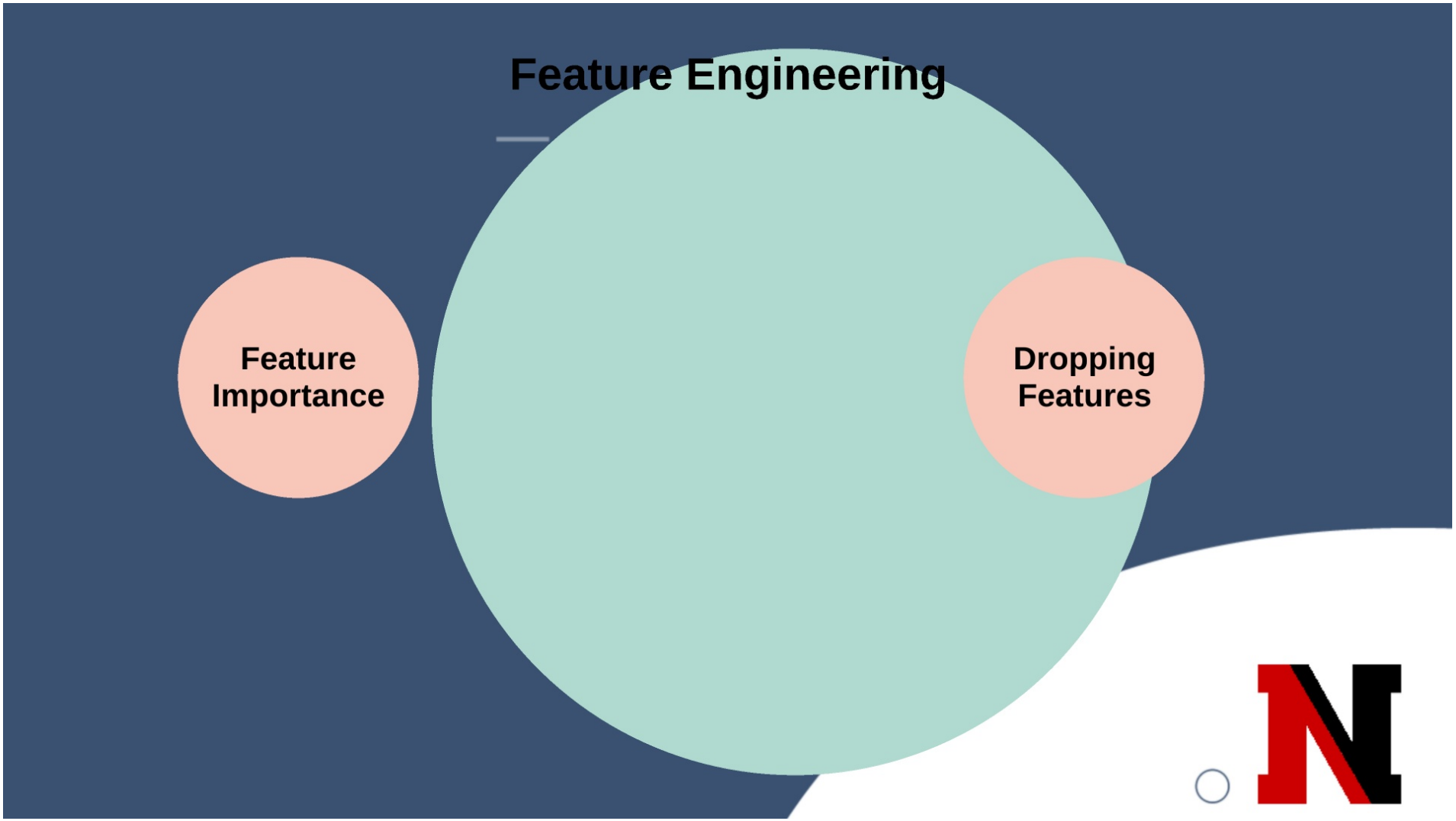


Heatmap

From the heatmap we can infer that price_range has a high positive correlation with RAM (higher the RAM, higher the price) followed by px_height and px_width whereas clock_speed three_g, four_g, n_cores have very less correlation with price_range

Pairplot




Feature Engineering
Feature Importance
Dropping Features

Feature Importance

```
ram              0.488554
battery_power    0.075082
px_width         0.058021
px_height        0.056073
mobile_wt        0.038660
int_memory       0.035827
talk_time        0.028914
pc               0.028701
sc_w             0.027934
clock_speed      0.026578
sc_h             0.026481
fc               0.024637
m_dep            0.023655
n_cores          0.022311
touch_screen     0.006924
wifi             0.006690
dual_sim         0.006504
four_g           0.006466
bluetooth        0.006431
three_g          0.005558
dtype: float64
```

Feature Importance using RandomForestClassifer - We use feature_importance property of this classifier to display the importance of the fetures.Scores of the importance is printed in the descending order. The results show that ram, battery_power, px_width, px_height have highest importance whereas wifi, blue, four_g, three_g have lowest importance.

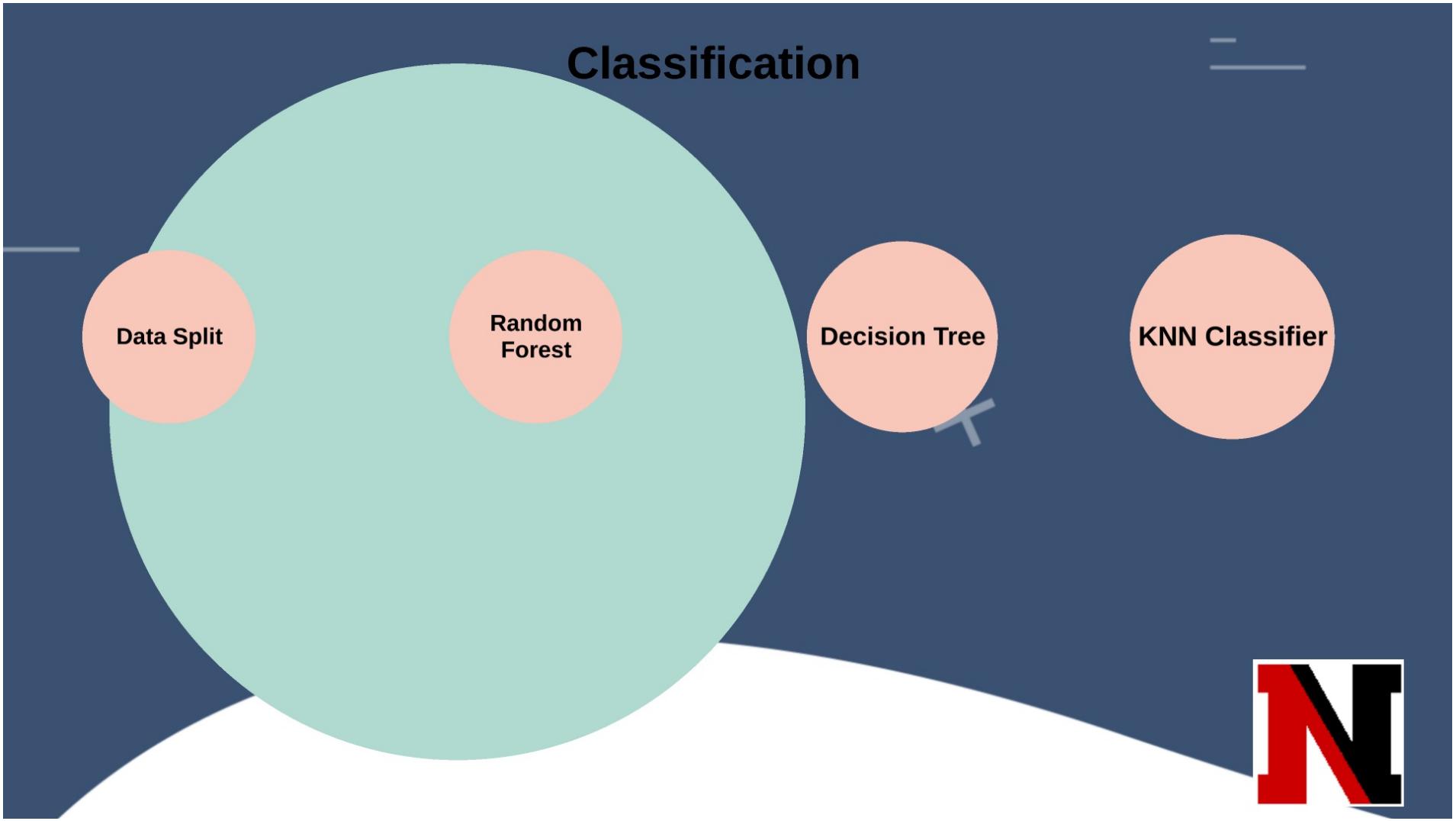
Dropping Features

```
X_Train = df_train.drop(['three_g','four_g','bluetooth','wifi'], axis=1)
X_Train
```

	battery_power	clock_speed	dual_sim	fc	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	touch_screen
0	842	2.2	0	1	7	0.6	188	2	2	20	756	2549	9	7	19	0
1	1021	0.5	1	0	53	0.7	136	3	6	905	1988	2631	17	3	7	1
2	563	0.5	1	2	41	0.9	145	5	6	1263	1716	2603	11	2	9	1
3	615	2.5	0	0	10	0.8	131	6	9	1216	1786	2769	16	8	11	0
4	1821	1.2	0	13	44	0.6	141	2	14	1208	1212	1411	8	2	15	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	794	0.5	1	0	2	0.8	106	6	14	1222	1890	668	13	4	19	1
1996	1965	2.6	1	0	39	0.2	187	4	3	915	1965	2032	11	10	16	1
1997	1911	0.9	1	1	36	0.7	108	8	3	868	1632	3057	9	1	5	1
1998	1512	0.9	0	4	46	0.1	145	5	5	336	670	869	18	10	19	1
1999	510	2.0	1	5	45	0.9	168	6	16	483	754	3919	19	4	2	1

2000 rows × 17 columns

Based on the analysis , we dropped 4 features with the least importance.


Classification
Data Split
Random Forest
Decision Tree
KNN Classifier

Data Split

```
x_features = df_train.drop('price_range', axis=1)
y_label = df_train['price_range']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_features, y_label, test_size=0.2, random_state=101)
```

We split the data 80% for training and 20% for testing

Random Forest

```
#Random Forest Classifier with n_estimator 100
from sklearn.ensemble import RandomForestClassifier
rfc_100 = RandomForestClassifier(n_estimators=100)
rfc_100.fit(X_train, y_train)

# Get the mean accuracy score for test set
rfc_100.score(X_test,y_test)*100
```

```
88.0
```

```
#Random Forest Classifier with n_estimator 150
from sklearn.ensemble import RandomForestClassifier
rfc_150 = RandomForestClassifier(n_estimators=150)
rfc_150.fit(X_train, y_train)
rfc_150.score(X_test,y_test)*100
```

```
88.0
```

```
#Random Forest Classifier with n_estimator 200
from sklearn.ensemble import RandomForestClassifier
rfc_200 = RandomForestClassifier(n_estimators=200)
rfc_200.fit(X_train, y_train)
rfc_200.score(X_test,y_test)*100
```

```
89.0
```

```
#Random Forest Classifier with n_estimator 250
from sklearn.ensemble import RandomForestClassifier
rfc_250 = RandomForestClassifier(n_estimators=250)
rfc_250.fit(X_train, y_train)
rfc_250.score(X_test,y_test)*100
```

```
88.25
```

Classificati on Report

Confusion Matrix

We trained the data using Random Forest Classifier. We tested the hyperparameter n_estimator with values 100, 150, 200 and 250. n_estimator of 200 gives us the be best score of 89. We then generated the classification report and confusion matrix.

Random Forest Classification Report

```
#Classification report and confusion matrix for random forest classifier with the best hyperparameter
from sklearn.metrics import classification_report, confusion_matrix
y_predict = rfc_200.fit(X_train, y_train).predict(X_test)
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.96      0.93      0.94        94
           1       0.78      0.89      0.83       102
           2       0.90      0.78      0.84       120
           3       0.92      0.95      0.94        84

    accuracy                           0.88       400
   macro avg       0.89      0.89      0.89       400
weighted avg       0.88      0.88      0.88       400
```

Precision and Recall look reasonable for label 0 and 3
Label 1 and 2 seem to have lower precision and lower recall respectively as can be also observed from F1 score.

Random Forest Confusion Matrix

```
print(confusion_matrix(y_test, y_predict))
```

```
[[87  7  0  0]
 [ 4 91  7  0]
 [ 0 19 94  7]
 [ 0  0  4 80]]
```

The principal diagonal is looking good showing overall True positive is good
The value 19 and 07 seems statistically significant, so improving the prediction between 1 and 2 label will boost overall metrics of the model

Decision Tree

```
#Decision Tree Classifier with max_depth = 5
from sklearn.tree import DecisionTreeClassifier
dtree_5 = DecisionTreeClassifier(max_depth=5)
dtree_5.fit(X_train,y_train)
dtree_5.score(X_train,y_train)*100
```

```
88.875
```

```
#Decision Tree Classifier with max_depth = 7
from sklearn.tree import DecisionTreeClassifier
dtree_7 = DecisionTreeClassifier(max_depth=7)
dtree_7.fit(X_train,y_train)
dtree_7.score(X_train,y_train)*100
```

```
96.875
```

```
#Decision Tree Classifier with max_depth = 10
from sklearn.tree import DecisionTreeClassifier
dtree_10 = DecisionTreeClassifier(max_depth=10)
dtree_10.fit(X_train,y_train)
dtree_10.score(X_train,y_train)*100
```

```
99.875
```

```
#Decision Tree Classifier with max_depth = 15
from sklearn.tree import DecisionTreeClassifier
dtree_15 = DecisionTreeClassifier(max_depth=10)
dtree_15.fit(X_train,y_train)
dtree_15.score(X_train,y_train)*100
```

```
99.875
```

Classification Report and Confusion Matrix

We train the data using Decision Tree Classifier. We tested the hyperparameter max_depth with values 5, 7 and 10. max_depth of 10 gives us the be best score of 99.875. We then generated the classification report and confusion matrix.

Decision Tree Confusion Matrix and Classification Report

```
y_predict = dtree_10.fit(X_train, y_train).predict(X_test)
```

```
#Classification report and confusion matrix for decision tree classifier
print(classification_report(y_test, y_predict))
print(confusion_matrix(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.92      0.86      0.89        94
           1       0.74      0.85      0.79       102
           2       0.85      0.75      0.80       120
           3       0.86      0.90      0.88        84

    accuracy                           0.83       400
   macro avg       0.84      0.84      0.84       400
weighted avg       0.84      0.83      0.84       400

[[81 13  0  0]
 [ 7 87  8  0]
 [ 0 18 90 12]
 [ 0  0  8 76]]
```

Precision and Recall look reasonable for label 0 and 3 Label 1 and 2 seem to have lower precision and lower recall respectively as can be also observed from F1 score.

The principal diagonal of the confusion matrix is looking good showing overall True positive. The value 18 and 9 seems statistically significant, so improving the prediction between 1 and 2 label will boost overall metrics of the model

KNN Classifier

```
#KNN Classifier with n_neighbors = 7
from sklearn.neighbors import KNeighborsClassifier
knn_7 = KNeighborsClassifier(n_neighbors=7)
knn_7.fit(X_train,y_train)
knn_7.score(X_train, y_train)*100
```

```
94.75
```

```
#KNN Classifier with n_neighbors = 10
from sklearn.neighbors import KNeighborsClassifier
knn_10 = KNeighborsClassifier(n_neighbors=10)
knn_10.fit(X_train,y_train)
knn_10.score(X_train, y_train)*100
```

```
95.0
```

```
#KNN Classifier with n_neighbors = 15
from sklearn.neighbors import KNeighborsClassifier
knn_15 = KNeighborsClassifier(n_neighbors=15)
knn_15.fit(X_train,y_train)
knn_15.score(X_train, y_train)*100
```

```
94.9375
```

Classification Report and Confusion Matrix

We train the data using KNN Classifier. We tested the hyperparameter n_neighbors with values 7, 10 and 15. n_neighbors of 10 gives us the be best score of 95. We then generated the classification report and confusion matrix.

KNN Classification Report and Confusion Matrix

```
#Classification report and confusion matrix for KNN classifier
y_predict = knn_10.fit(X_train, y_train).predict(X_test)
print(classification_report(y_test, y_predict))
print(confusion_matrix(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        94
           1       0.88      0.94      0.91       102
           2       0.94      0.88      0.91       120
           3       0.96      0.94      0.95        84

    accuracy                           0.93       400
   macro avg       0.94      0.94      0.94       400
weighted avg       0.93      0.93      0.93       400

[[ 92   2   0   0]
 [  4  96   2   0]
 [  0  11 106   3]
 [  0   0   5  79]]
```

Precision and Recall look reasonable for label 0 and 3
Label 1 and 2 seem to have lower precision and lower recall respectively as can be also observed from F1 score.

The principal diagonal of the confusion matrix is looking good showing overall True positive.
The value 11,5 and 5 seems statistically significant, so improving the prediction between 1 and 2 label will boost overall metrics of the model

Conclusion And Future Scope

After training 3 models - Random Forest, Decision Tree and K neighbors we see that the best model for this classification problem is Decision Tree followed by KNN. In the classification report we see that precision and recall for price range 1 and 2 is relatvely low where as for 0 and 3 it is good.

More features is perhaps needed to boost the P/R here as the data label distribution is equal among 4 classes.