Capstone Project-3

Mobile Price Range Prediction

(Classification)

By- Prasad Khedkar
(Individual Project)

Points of Discussion :

1. The Problem Statement
2. Summary of the Data Set
3. Data Info and Data Cleaning
4. EDA
5. Data Preparation
6. Data Splitting & Feature Scaling
7. Different Model Implementations
8. Final Model Selection
9. Hyper-parameter tuning
10. Final conclusion

The Problem Statement

In the competitive phone market, companies want to understand sales data of mobile phones and factors which drive the prices.

The objective is to find out how features of mobile phone(e.g., RAM, Storage etc) are related to its price.

<u>In this classification problem</u>, we are not to predict the actual price of mobile but to classify it price range into 4 categories, namely- <u>0(Low cost), 1(Moderate cost), 2(High cost), 3(Expensive)</u>

Summary of the Data Set :

Here, we are provided with the data of different mobile features :

Fields	Description
Battery_power	Battery capacity in mAh
Blue	Has bluetooth or not
Clock_speed	speed at which microprocessor executes instructions
Dual_sim	Has dual sim support or not
Fc	Front Camera megapixels
Four_g	Has 4G or not
Int_memory	Internal memory capacity
M_dep	Mobile depth in cm
MObile_wt	Weight of mobiles phone
N_cores	Number of cores in processor
Pc	Primary Camera mega pixels
Px_height	Pixel resolution height
Px_width	Pixel resolution width
Ram	Random Access Memory in MB
Sc_h	Screen Height
Sc_w	Screen width
Talk_time	Longest that a single battery can last over a call
Three_g	Has 3g or not
Wifi	Has wifi or not
Price_range	This is the target variable with a value of 0(low cost) 1(medium cost), 2 (high cost) 3(very high cost)

We will see how these features are related to mobile **price_range**, which is our target variable

The original shape of dataset is (2000,21) indicating 2000 rows and 21 columns(features)

```
df.shape
```

```
(2000, 21)
```

Data Info :

```
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)
```

As we can see,
Total 21 features are present in our data set
Among them 19 features have datatype int64 and remaining two having float64

Data Cleaning :

As can be seen,
no null values present in the dataset

And after using drop_duplicates function, the shape didn't change indicating no duplicate entries present either.

```
df.drop_duplicates(keep='first',inplace=True)
```

```
df.shape
```

```
(2000, 21)
```

```
df.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
```

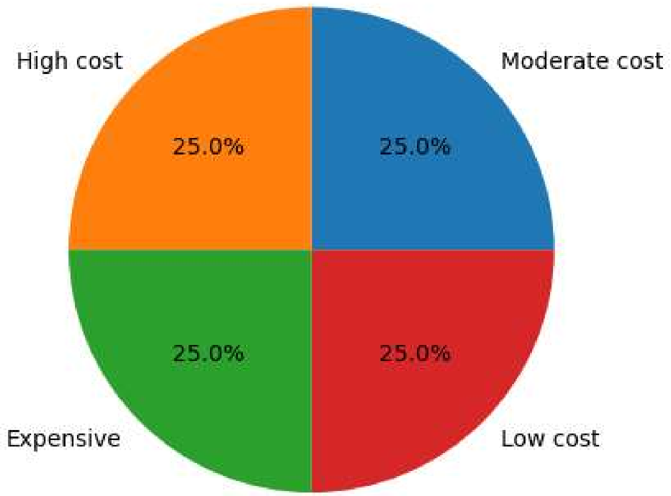
EDA(Exploratory Data Analysis) :

Our target variable 'price_range' has equal no. of shares in each categories as shown in pie chart.

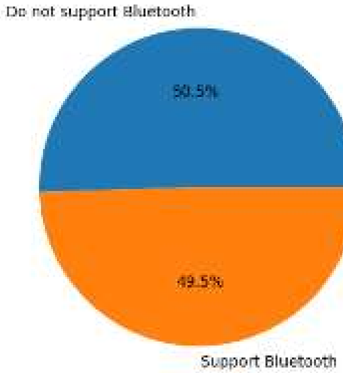
0 ⇨ Low cost
1 ⇨ Moderate cost
2 ⇨ High Cost
3 ⇨ Expensive

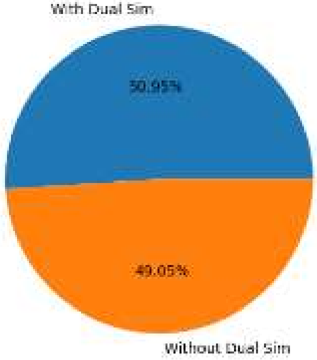
```
df['price_range'].value_counts()
```

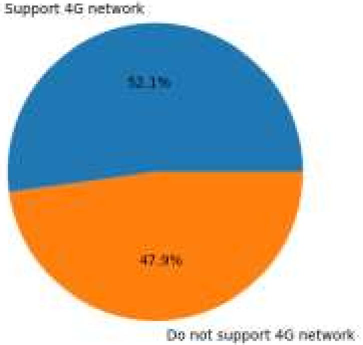
```
1    500
2    500
3    500
0    500
Name: price_range, dtype: int64
```

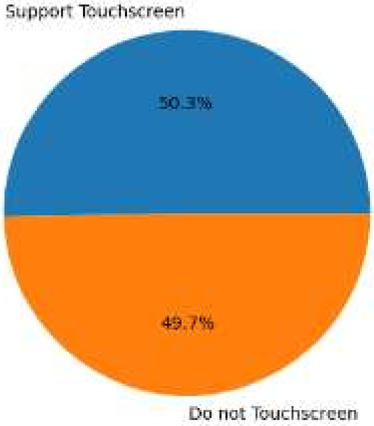


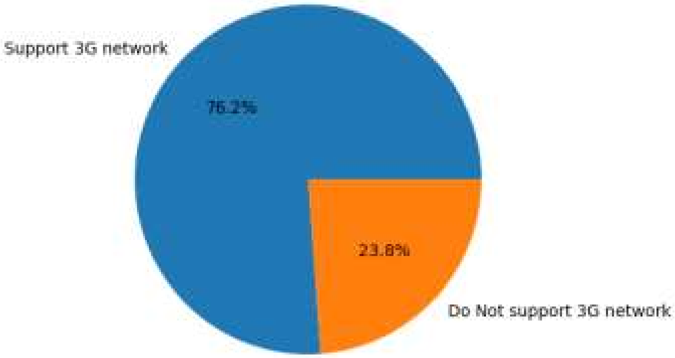
Univariate Analysis(of diff. features):

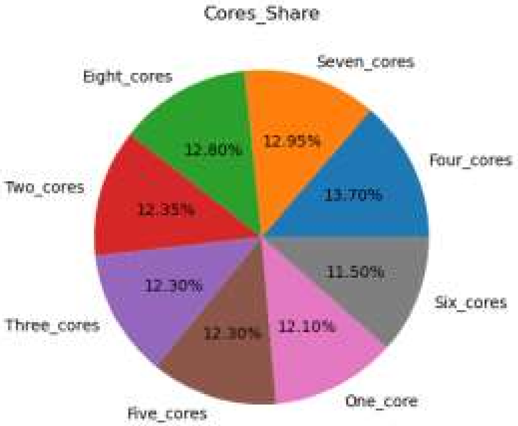




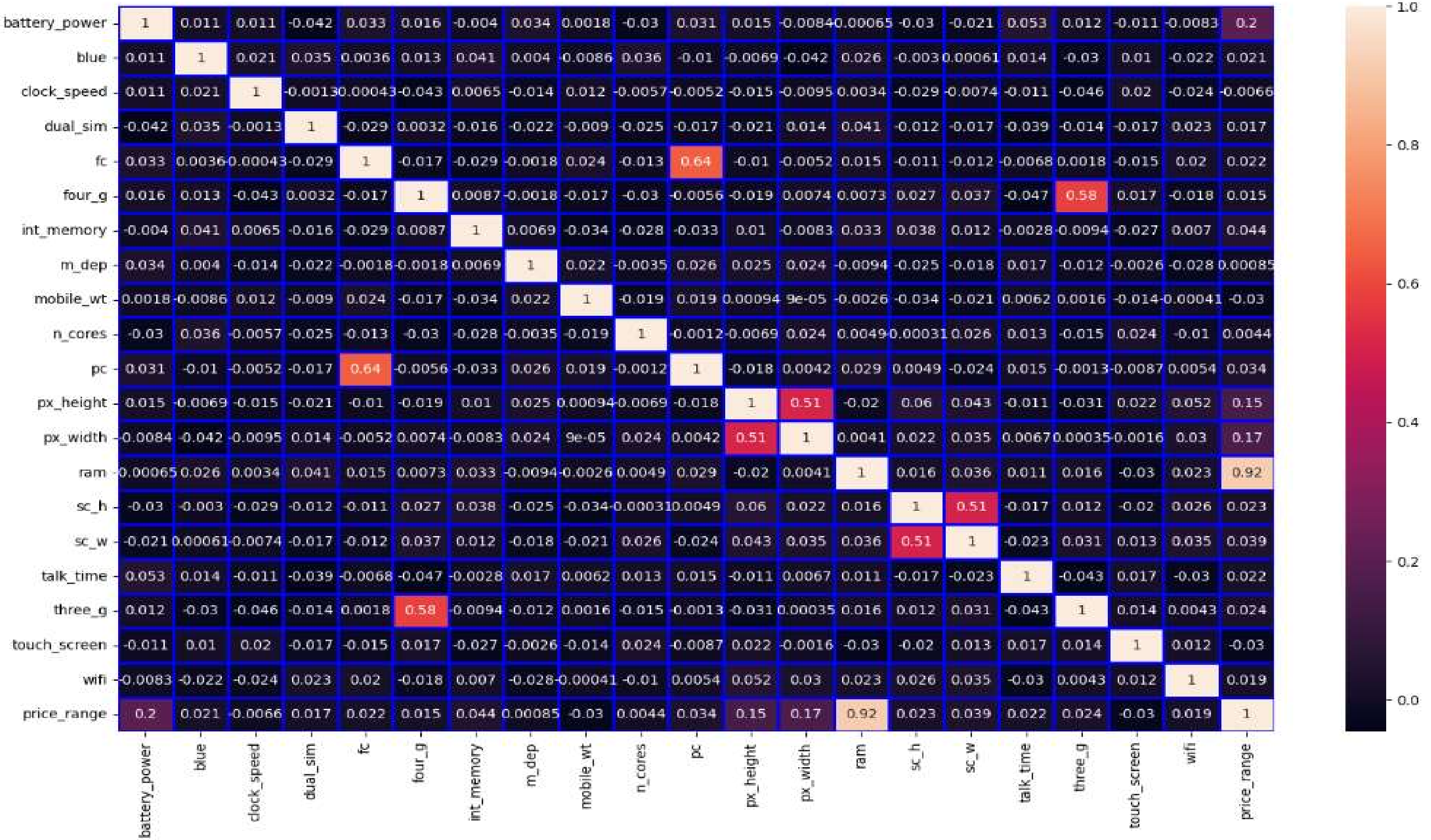




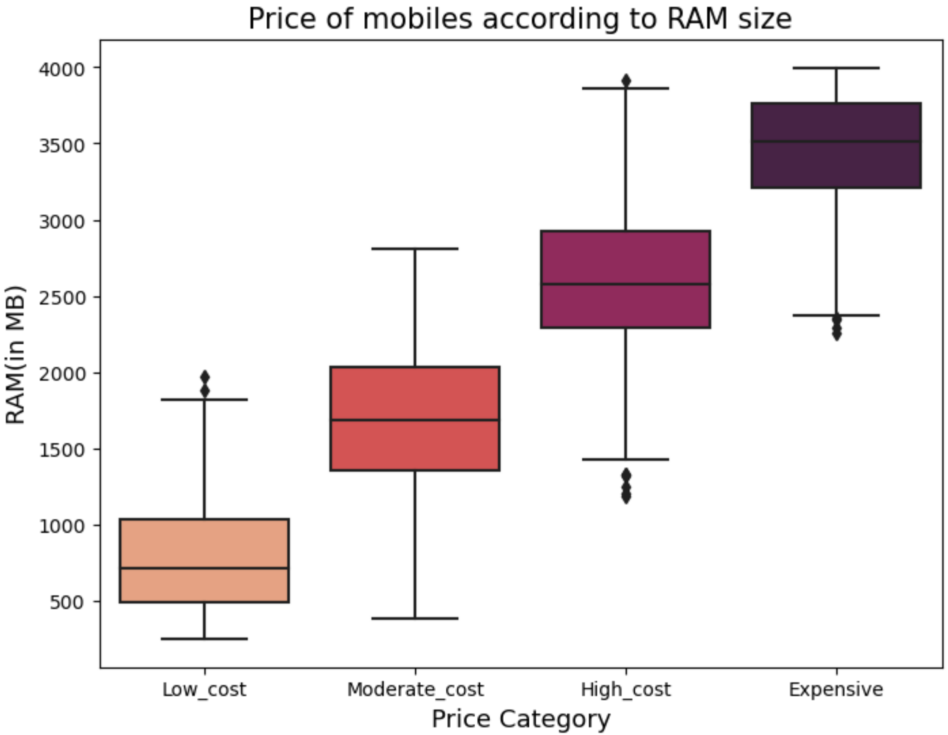


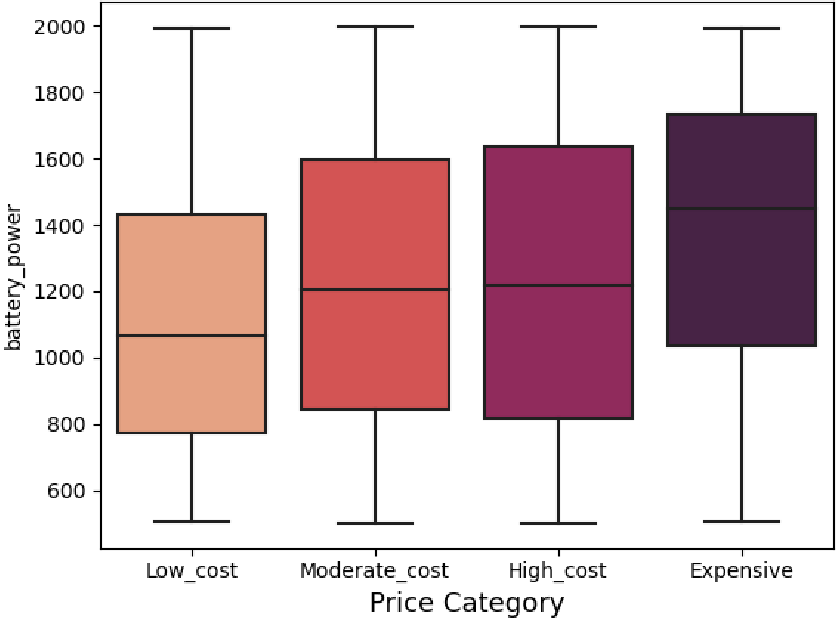


Multivariate/Bivariate Analysis:



Followings are the box plots of features which affects price_range significantly-

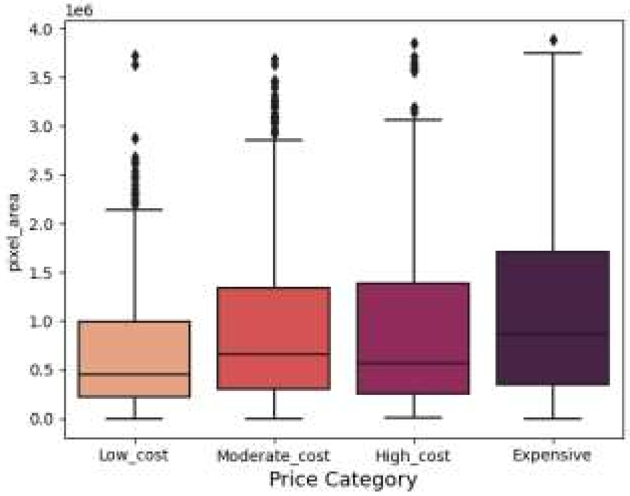


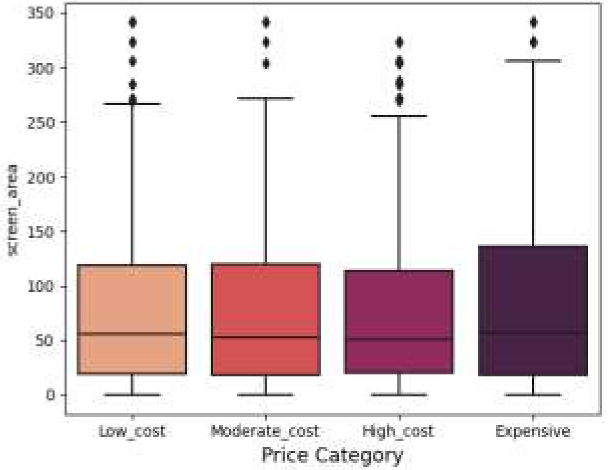


Feature Engineering:

Added new columns –
- pixel_area(Actual display)
- screen(Total display)

And dropping unwanted features like-
- -'px_height',
- -'px_width',
- -'sc_h',
- -'sc_w'





Data Preparation:

After selecting price_range as target variable we divided feature set into numerical and categorical features like-

```
num = X.drop(['blue','dual_sim','four_g','n_cores','three_g','touch_screen','wifi'], axis = 1)
cat = X.filter(['blue','dual_sim','four_g','n_cores','three_g','touch_screen','wifi'])
```

```
num.head()
```

	battery_power	clock_speed	fc	int_memory	m_dep	mobile_wt	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	pixel_area	screen_area
0	842	2.2	1	7	0.6	188	2	20	756	2549	9	7	19	15120	63
1	1021	0.5	0	53	0.7	136	6	905	1988	2631	17	3	7	1799140	51
2	563	0.5	2	41	0.9	145	6	1263	1716	2603	11	2	9	2167308	22
3	615	2.5	0	10	0.8	131	9	1216	1786	2769	16	8	11	2171776	128
4	1821	1.2	13	44	0.6	141	14	1208	1212	1411	8	2	15	1464096	16

Numerical features

```
cat.head()
```

	blue	dual_sim	four_g	n_cores	three_g	touch_screen	wifi
0	0	0	0	2	0	0	1
1	1	1	1	3	1	1	0
2	1	1	1	5	1	1	0
3	1	0	0	6	1	0	0
4	1	0	1	2	1	1	0

Categorical features which need encoding

After encoding the categorical features by OneHotEncoder using panda's get_dummies method, we successfully converted categories into numerical values.

And our data is now ready for training on different models -

Data Splitting and Feature Scaling :

Data is splited into - **70% - Training set**

& - **30% - Test set**

StandardScaler is used for scaling the data.

Different Model Implementations:

Different algorithms used for final model implementation:

- ➢ Logistic Regressor
- ➢ Random Forest Classifier
- ➢ K-NN Classifier

Among three, Logistic Regressor performed exceedingly well and will be used for final implementation.

```
accuracy_score(y_train , y_pred_train)
```

```
0.9478571428571428
```

```
accuracy_score(y_test,y_pred)
```

```
0.905
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       135
           1       0.86      0.89      0.88       149
           2       0.90      0.83      0.87       168
           3       0.93      0.95      0.94       148

    accuracy                           0.91       600
   macro avg       0.91      0.91      0.91       600
weighted avg       0.91      0.91      0.90       600
```

Hyper-Parameter Tuning:

After selecting Logistic Regressor for final model implementation, we did hyper-parameter tuning to further increase the performance of the model,
For that GridSearchCV method is used-

```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [100, 10, 1.0, 0.1, 0.01],
                         'max_iter': [500, 1000],
                         'random_state': [0, 1, 10, 55, 99, 555, 999, 9094,
                                          4494]},
             scoring='r2')
```

```
grid_model.best_params_
```

```
{'C': 100, 'max_iter': 500, 'random_state': 0}
```

```
accuracy_score(y_train, y_pred_train_final)
```

```
0.9592857142857143
```

```
accuracy_score(y_test, y_pred_test_final)
```

```
0.915
```

```
print(classification_report(y_train, y_pred_train_final))
```

	precision	recall	f1-score	support
0	0.98	0.98	0.98	365
1	0.95	0.95	0.95	351
2	0.94	0.93	0.94	332
3	0.96	0.97	0.97	352
accuracy			0.96	1400
macro avg	0.96	0.96	0.96	1400
weighted avg	0.96	0.96	0.96	1400

```
print(classification_report(y_test, y_pred_test_final))
```

	precision	recall	f1-score	support
0	0.95	0.96	0.95	135
1	0.88	0.89	0.88	149
2	0.89	0.88	0.89	168
3	0.95	0.94	0.95	148
accuracy			0.92	600
macro avg	0.92	0.92	0.92	600
weighted avg	0.92	0.92	0.92	600

Final Conclusion :

- ✓ No null values or duplicates present in our dataset.
- ✓ ‘price_range’ was highly correlated to ‘ram’
- ✓ After going through different algorithms, Logistic Regressor is selected for final model implementation.
- ✓ The algorithm performed good even without tuning it’s parameters.
- ✓ Upon tuning, the performance increased from 94% on train score to 95% & from 90% on test score to 91% .