Precise Mobile Price Range Classification and Prediction Harnessing ML and Neural Networks

Project 1

Sudhish Subramaniam

NUID: 002752858

subramaniam.su@northeastern.edu

Submission Date:______________29th November, 2023_______________

Objective

The primary aim of this project is to have phones classified based on their specifications using advanced machine learning and neural network methodologies. By employing a predictive model, phones will be accurately assigned to distinct price ranges, contributing to the enhancement of classification accuracy, and facilitating a comprehensive understanding of the relationships between diverse phone features and their corresponding price categories.

Stakeholders

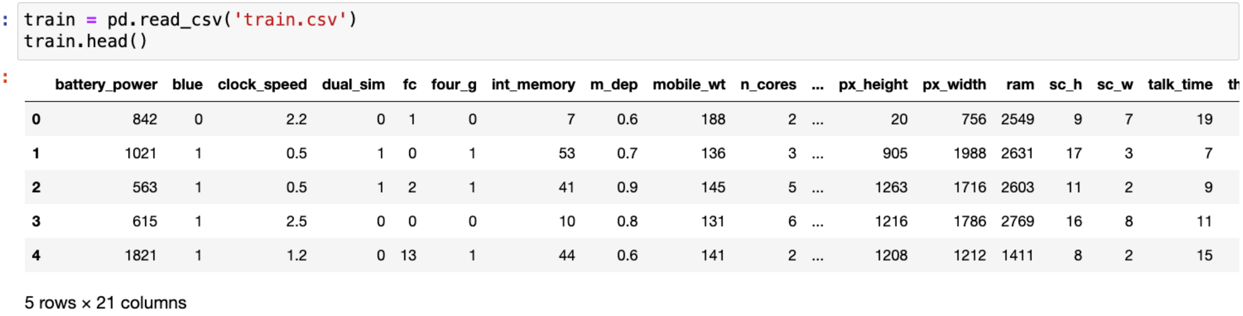
Firstly, stakeholders, including manufacturers, retailers, and consumers, will benefit from improved decision-making processes. With accurate classification, manufacturers can align their pricing strategies more effectively, ensuring that their products are competitively positioned within the market. Retailers can optimize their inventory management by stocking products according to their predicted price categories. Additionally, consumers can make more informed purchasing decisions based on a comprehensive understanding of the relationships between various phone features and their corresponding price points.

Secondly, the enhanced classification accuracy achieved through advanced machine learning methodologies will contribute to improved market intelligence. Stakeholders will gain valuable insights into consumer preferences, allowing for the development of targeted marketing strategies. By comprehensively understanding the relationships between diverse phone features and price categories, manufacturers can tailor their product development efforts to meet specific market demands.

Furthermore, the project's outcomes can facilitate the identification of emerging trends within the smartphone industry. Stakeholders will be better positioned to anticipate shifts in consumer preferences and technological advancements, enabling them to proactively adapt their business strategies.

Data

Several libraries have been imported to facilitate various aspects of machine learning and data analysis. Firstly, warnings have been imported to manage warning messages during the execution of the code. NumPy and Pandas have been imported as 'np' and 'pd', respectively, for efficient numerical and data manipulation. Matplotlib, designated as 'plt', and Seaborn have been imported for data visualization, allowing for the creation of informative plots. Additionally, the 'mpl' alias has been assigned to Matplotlib for further configuration options. The 'train_test_split' and 'GridSearchCV' modules from the scikit-learn library have been imported to facilitate data splitting for model training and hyperparameter tuning, respectively. StandardScaler from scikit-learn is imported for feature scaling. DecisionTreeClassifier and tree from scikit-learn are imported to implement decision tree-based models. GradientBoostingClassifier, RandomForestClassifier, and SVC are imported to utilize ensemble methods and support vector machines for classification tasks. Various modules from scikit-learn.metrics are imported for evaluating model performance, including accuracy, precision, recall, and F1 score. KFold and StratifiedKFold from scikit-learn.model_selection are imported for cross-validation purposes. Finally, the %notebook matplotlib magic command has been employed to ensure that Matplotlib plots are displayed inline in the Jupyter Notebook, and warnings have been filtered to ignore them during code execution. The train data is seen is first imported and a glimpse can be seen in Fig. 1. Description of the train data is seen in Fig. 2. A glimpse of test data is seen in Fig. 3.

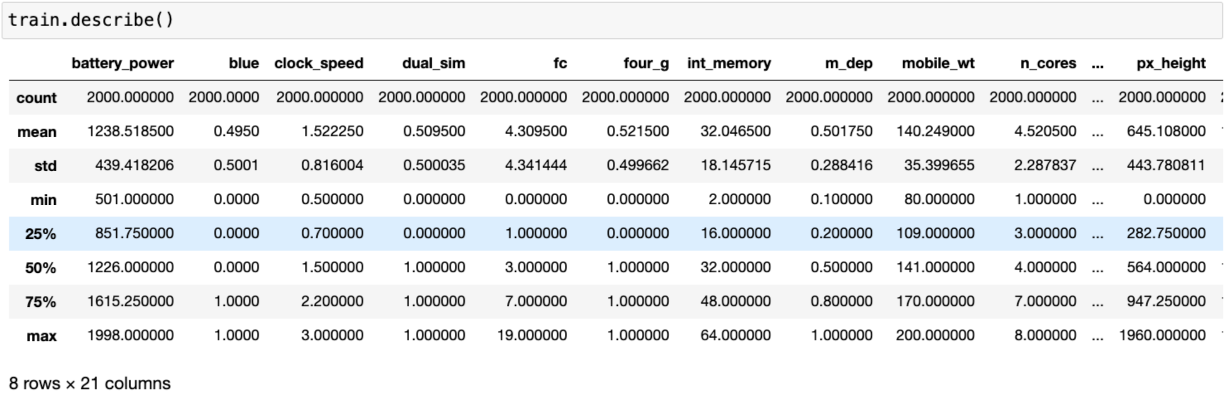


```
train = pd.read_csv('train.csv')
train.head()
```

	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
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

Fig. 1. Glimpse of Train Data

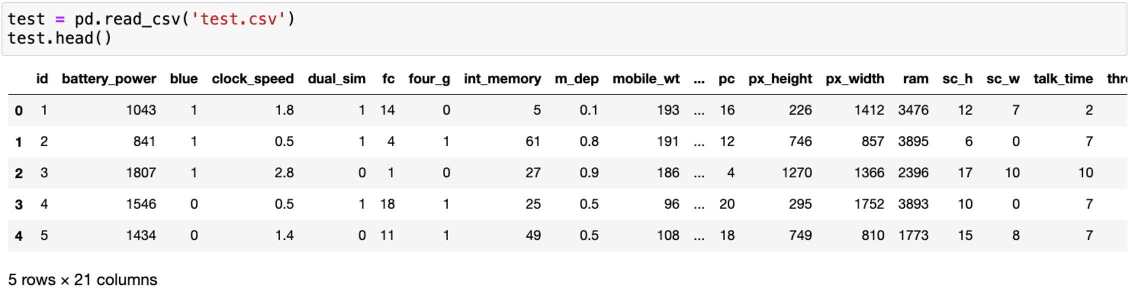


```
train.describe()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height
count	2000.000000	2000.0000	2000.000000	2000.000000	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	4.520500	...	645.108000
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	2.287837	...	443.780811
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	...	0.000000
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	...	282.750000
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	4.000000	...	564.000000
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	...	947.250000
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	...	1960.000000

8 rows × 21 columns

Fig. 2. Description of Train Data



```
test = pd.read_csv('test.csv')
test.head()
```

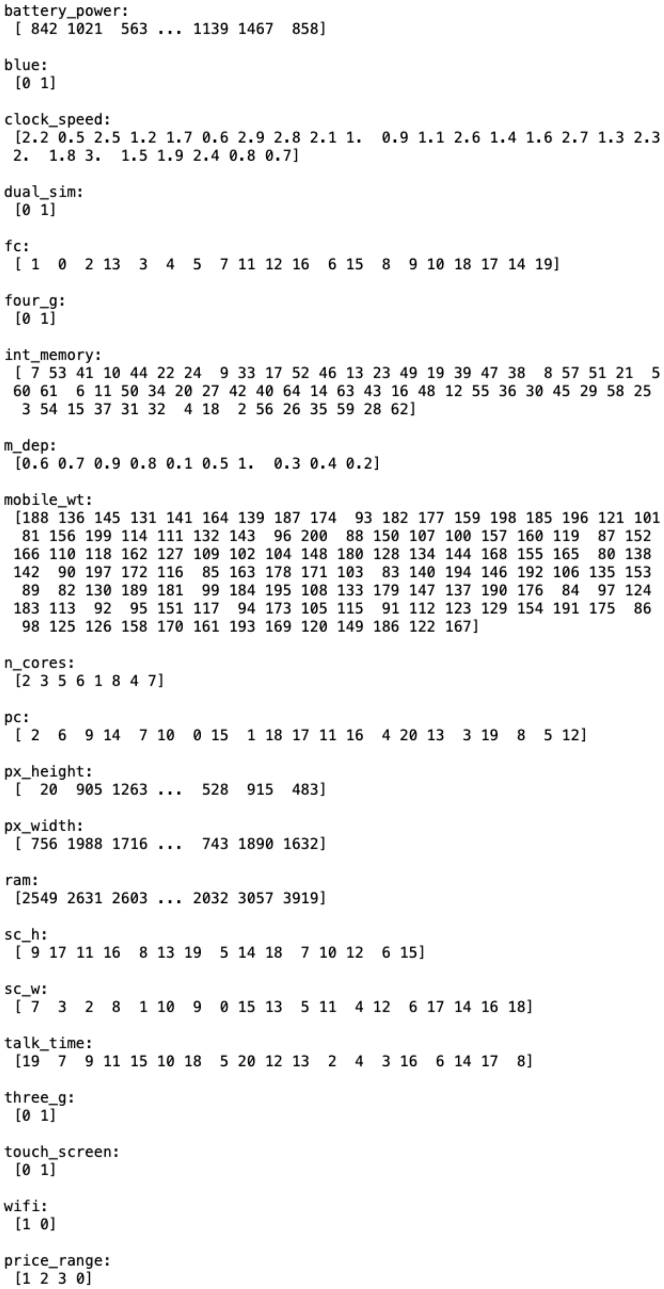
	id	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	...	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	thr
0	1	1043	1	1.8	1	14	0	5	0.1	193	...	16	226	1412	3476	12	7	2	
1	2	841	1	0.5	1	4	1	61	0.8	191	...	12	746	857	3895	6	0	7	
2	3	1807	1	2.8	0	1	0	27	0.9	186	...	4	1270	1366	2396	17	10	10	
3	4	1546	0	0.5	1	18	1	25	0.5	96	...	20	295	1752	3893	10	0	7	
4	5	1434	0	1.4	0	11	1	49	0.5	108	...	18	749	810	1773	15	8	7	

5 rows × 21 columns

Fig. 3. Glimpse of Test Data

Exploratory Data Analysis

The exploratory data analysis (EDA) was conducted on the dataset to gain insights into the key features and their distributions. The value in the data is shown in Fig. 4. The dataset comprised both numerical and categorical variables, with a focus on the 'price_range' variable as the target. Initially, a thorough examination of the dataset's columns was performed, revealing unique values and their distributions.



```
battery_power:
 [ 842 1021  563 ... 1139 1467  858]

blue:
 [0 1]

clock_speed:
 [2.2 0.5 2.5 1.2 1.7 0.6 2.9 2.8 2.1 1.  0.9 1.1 2.6 1.4 1.6 2.7 1.3 2.3
 2.  1.8 3.  1.5 1.9 2.4 0.8 0.7]

dual_sim:
 [0 1]

fc:
 [ 1  0  2 13  3  4  5  7 11 12 16  6 15  8  9 10 18 17 14 19]

four_g:
 [0 1]

int_memory:
 [ 7 53 41 10 44 22 24  9 33 17 52 46 13 23 49 19 39 47 38  8 57 51 21  5
 60 61  6 11 50 34 20 27 42 40 64 14 63 43 16 48 12 55 36 30 45 29 58 25
  3 54 15 37 31 32  4 18  2 56 26 35 59 28 62]

m_dep:
 [0.6 0.7 0.9 0.8 0.1 0.5 1.  0.3 0.4 0.2]

mobile_wt:
 [188 136 145 131 141 164 139 187 174  93 182 177 159 198 185 196 121 101
  81 156 199 114 111 132 143  96 200  88 150 107 100 157 160 119  87 152
 166 110 118 162 127 109 102 104 148 180 128 134 144 168 155 165  80 138
 142  90 197 172 116  85 163 178 171 103  83 140 194 146 192 106 135 153
  89  82 130 189 181  99 184 195 108 133 179 147 137 190 176  84  97 124
 183 113  92  95 151 117  94 173 105 115  91 112 123 129 154 191 175  86
  98 125 126 158 170 161 193 169 120 149 186 122 167]

n_cores:
 [2 3 5 6 1 8 4 7]

pc:
 [ 2  6  9 14  7 10  0 15  1 18 17 11 16  4 20 13  3 19  8  5 12]

px_height:
 [  20  905 1263 ...  528  915  483]

px_width:
 [ 756 1988 1716 ...  743 1890 1632]

ram:
 [2549 2631 2603 ... 2032 3057 3919]

sc_h:
 [ 9 17 11 16  8 13 19  5 14 18  7 10 12  6 15]

sc_w:
 [ 7  3  2  8  1 10  9  0 15 13  5 11  4 12  6 17 14 16 18]

talk_time:
 [19  7  9 11 15 10 18  5 20 12 13  2  4  3 16  6 14 17  8]

three_g:
 [0 1]

touch_screen:
 [0 1]

wifi:
 [1 0]

price_range:
 [1 2 3 0]
```

Fig. 4. Glimpse into all values in the data

The numerical features, including 'battery_power,' 'clock_speed,' and 'ram,' were explored along with categorical attributes such as 'blue,' 'dual_sim,' and 'wifi.' Visualization techniques were employed to illustrate the distribution of the target variable across different price ranges. A bar chart showcased the count of instances for each price range, providing an overview of the dataset's class distribution, this is shown in Fig. 5.

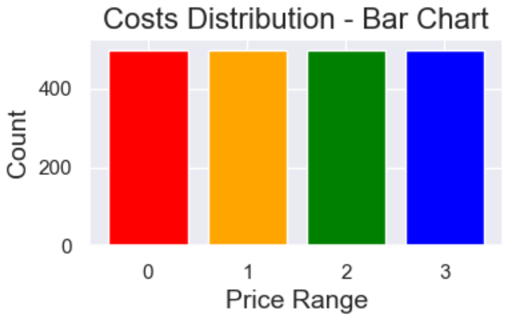


Fig. 5. Bar Chart of Costs distribution

Further exploration involved analyzing the distribution of features in both the training and test datasets. Kernel density estimation (KDE) plots (shown in Fig. 6 (a), 6 (b)) were utilized to visualize the density of numerical features, such as 'battery_power' and 'ram,' in both the training and test datasets. Additionally, bar charts (shown in Fig. 7) displayed the number of unique values in both numerical and categorical columns, aiding in understanding the diversity of the dataset. To enhance interpretability, categorical variables were transformed into more descriptive categories, such as 'no-blue' and 'has-blue' for the 'blue' feature.

Pie charts (Fig. 8) were employed to visualize the distribution of categorical variables like 'dual_sim' and 'four_g.' The transformation of numerical variables, such as 'battery_power' and 'ram,' into kernel density plots against the target variable 'price_range' provided insights into potential patterns and relationships. Contour plots (Fig. 9) were utilized to visualize the relationships between the target variable and various features, highlighting potential trends and patterns.

Anomalies within the dataset were addressed, such as instances where 'px_height' and 'px_width' values were below certain thresholds. Similarly, concerns regarding 'sc_w' and 'sc_h' values below specified thresholds were identified. The presence of outliers was examined through boxplots and scatter plots (shown in Fig. 10 (a) and 10(b)), revealing insights into the spread and distribution of numerical features. Finally, a correlation heatmap (shown in Fig. 11) was generated to illustrate the relationships between numerical variables, aiding in identifying potential multicollinearity.

The exploratory data analysis provided a comprehensive understanding of the dataset, uncovering patterns, distributions, and potential outliers across numerical and categorical features. This analysis laid the foundation for subsequent modeling efforts by informing feature engineering and selection strategies.

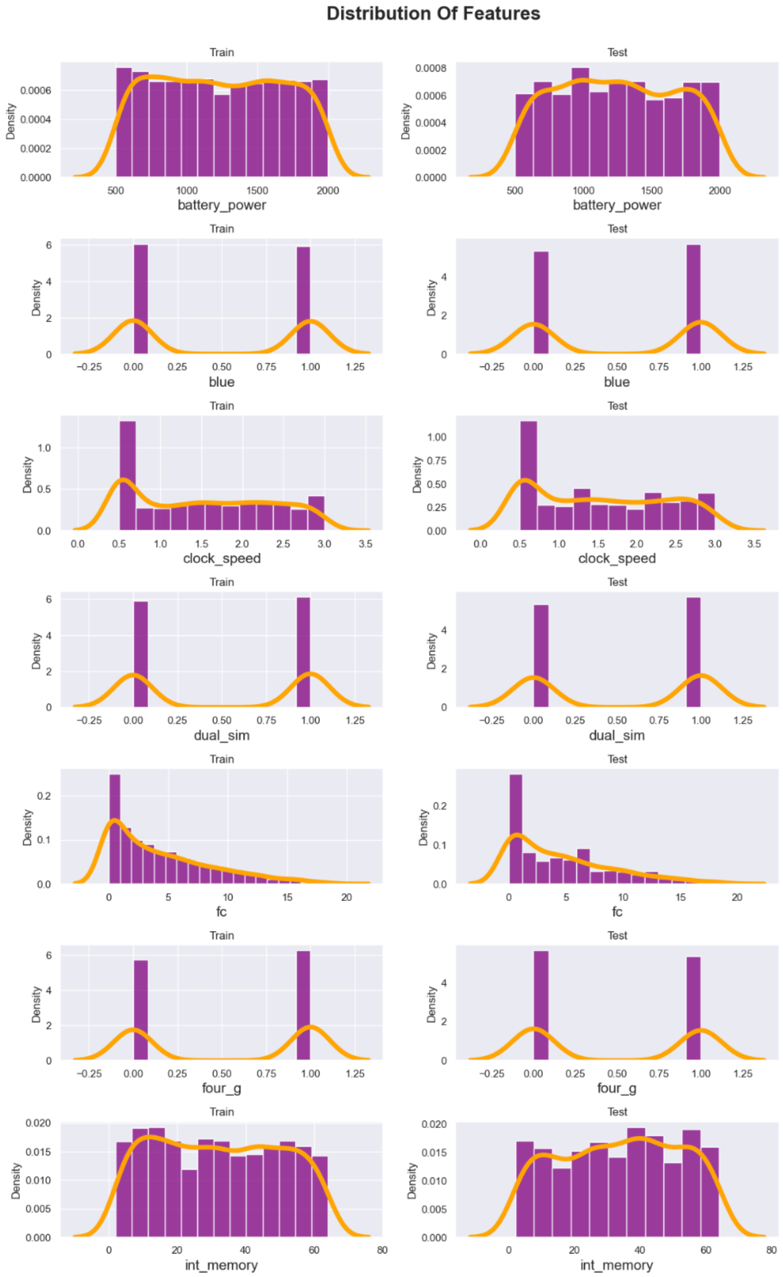


Fig. 6. (a). KDE Plots of features

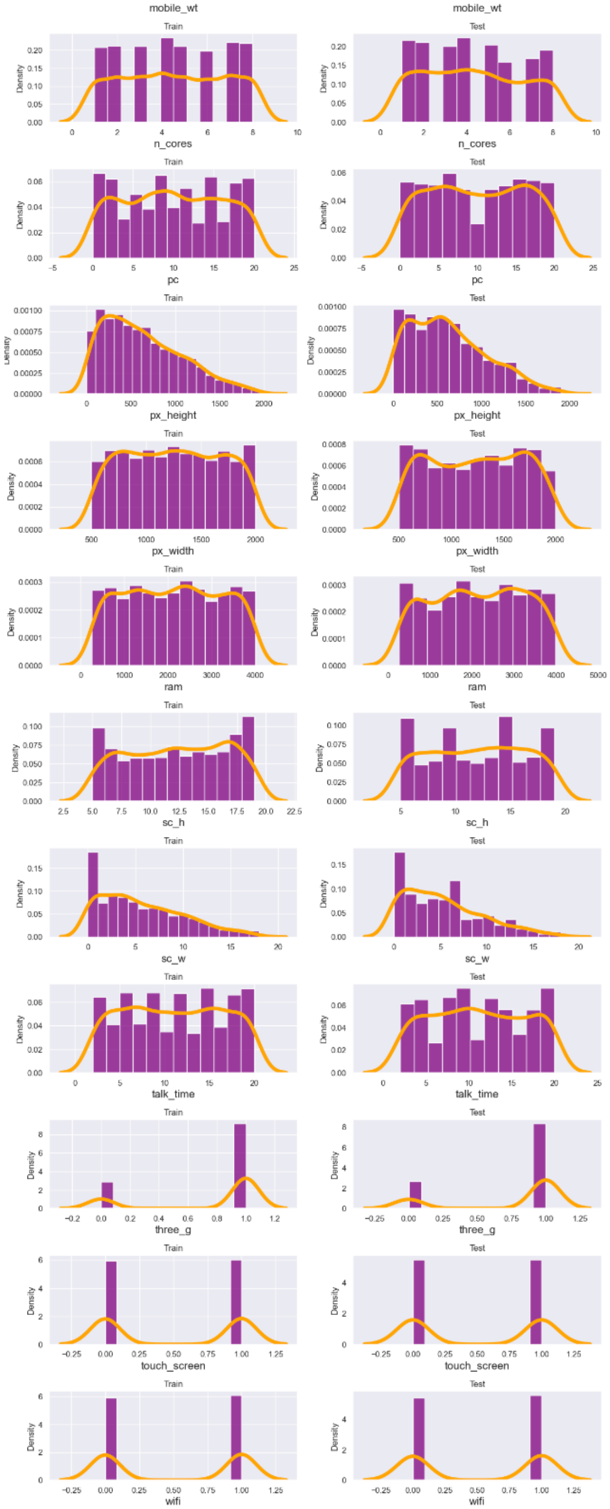


Fig. 6. (b). KDE Plots of features

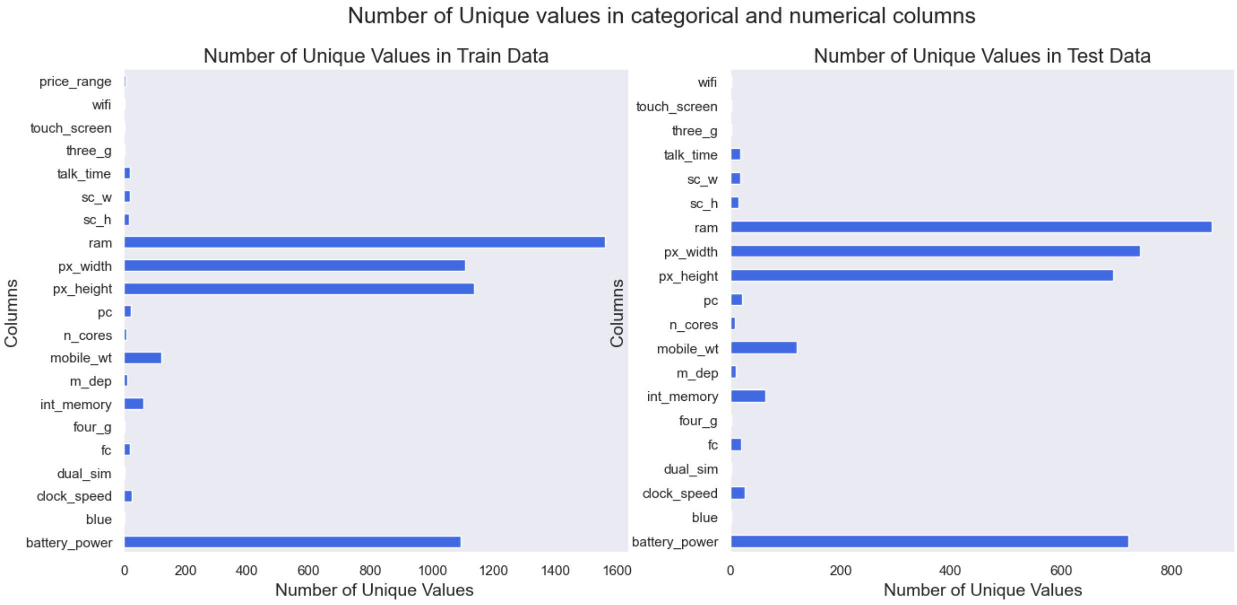


Fig. 7. Number of Unique values in categorical and numerical columns

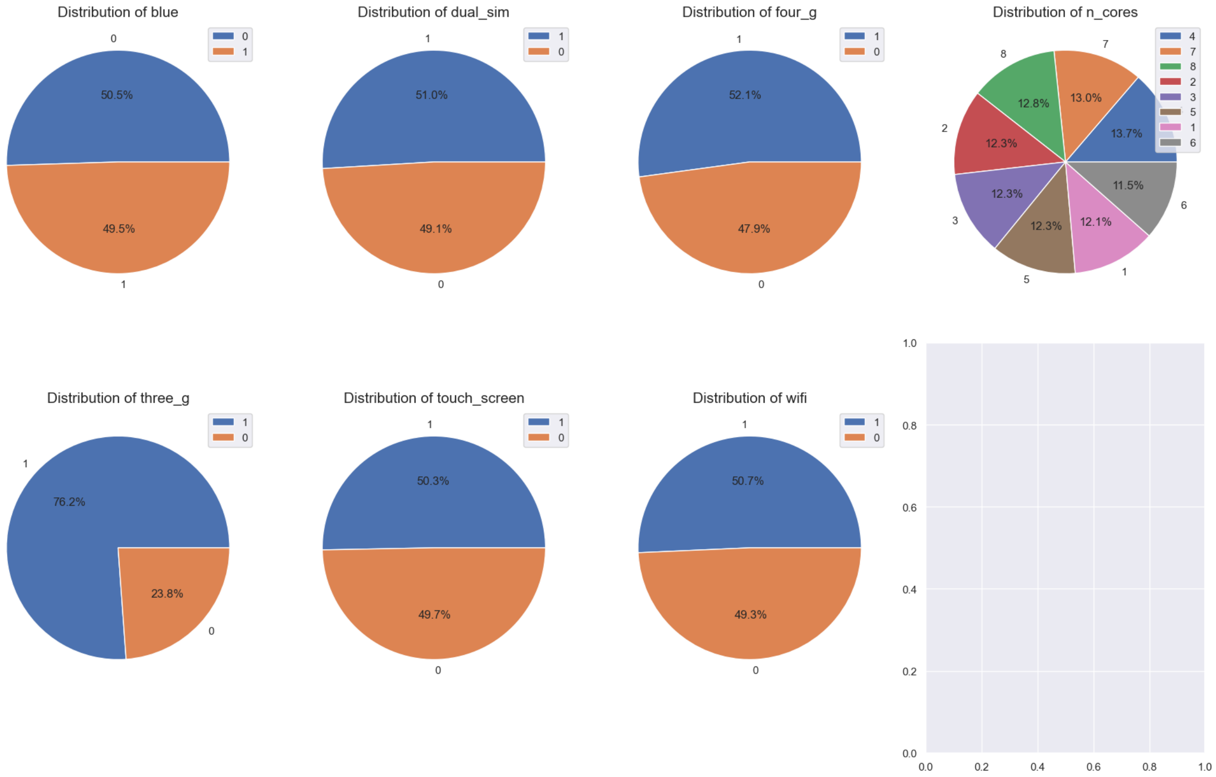


Fig. 8. Pie Charts of Categorical Columns

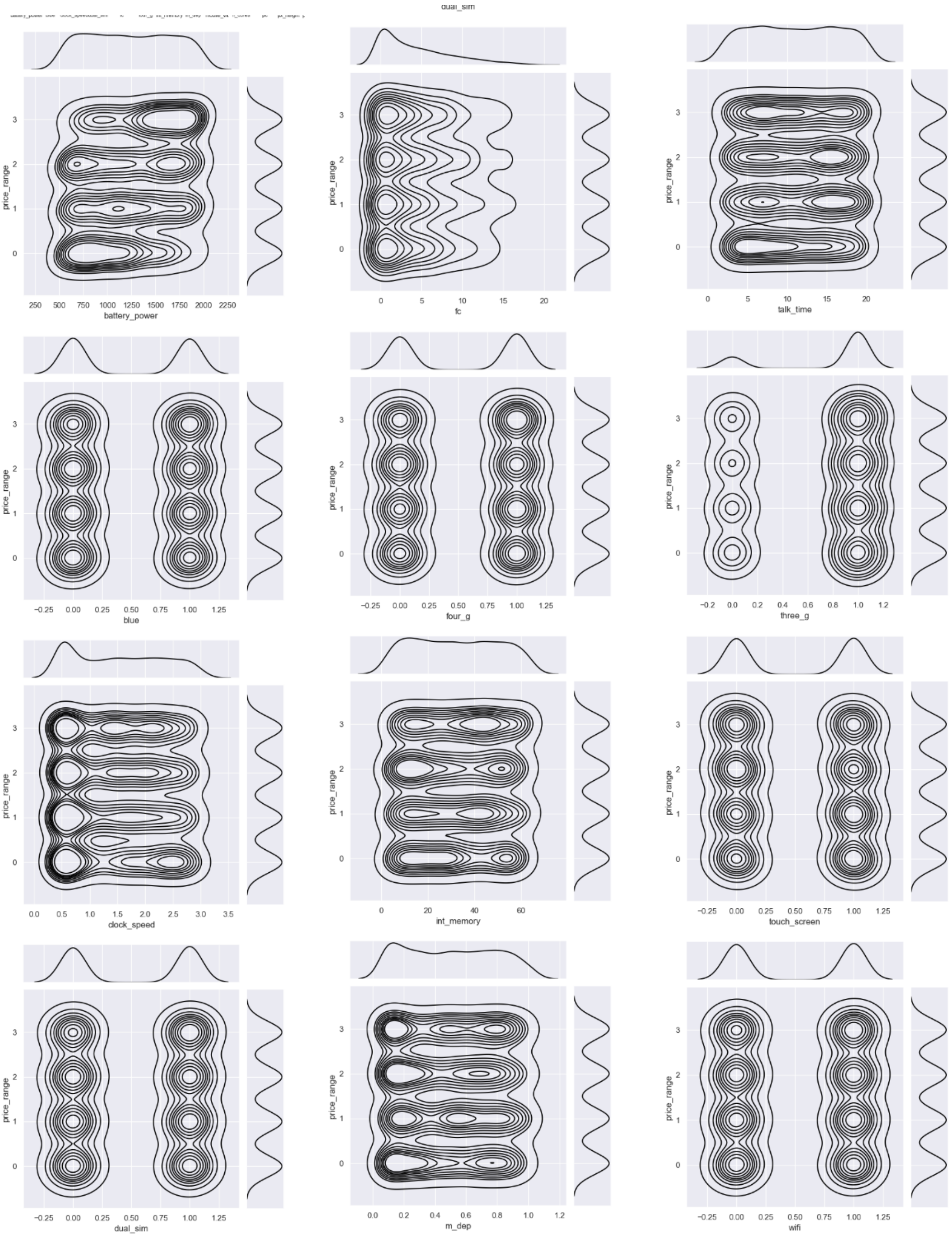


Fig. 9. Contour plots of columns in the data

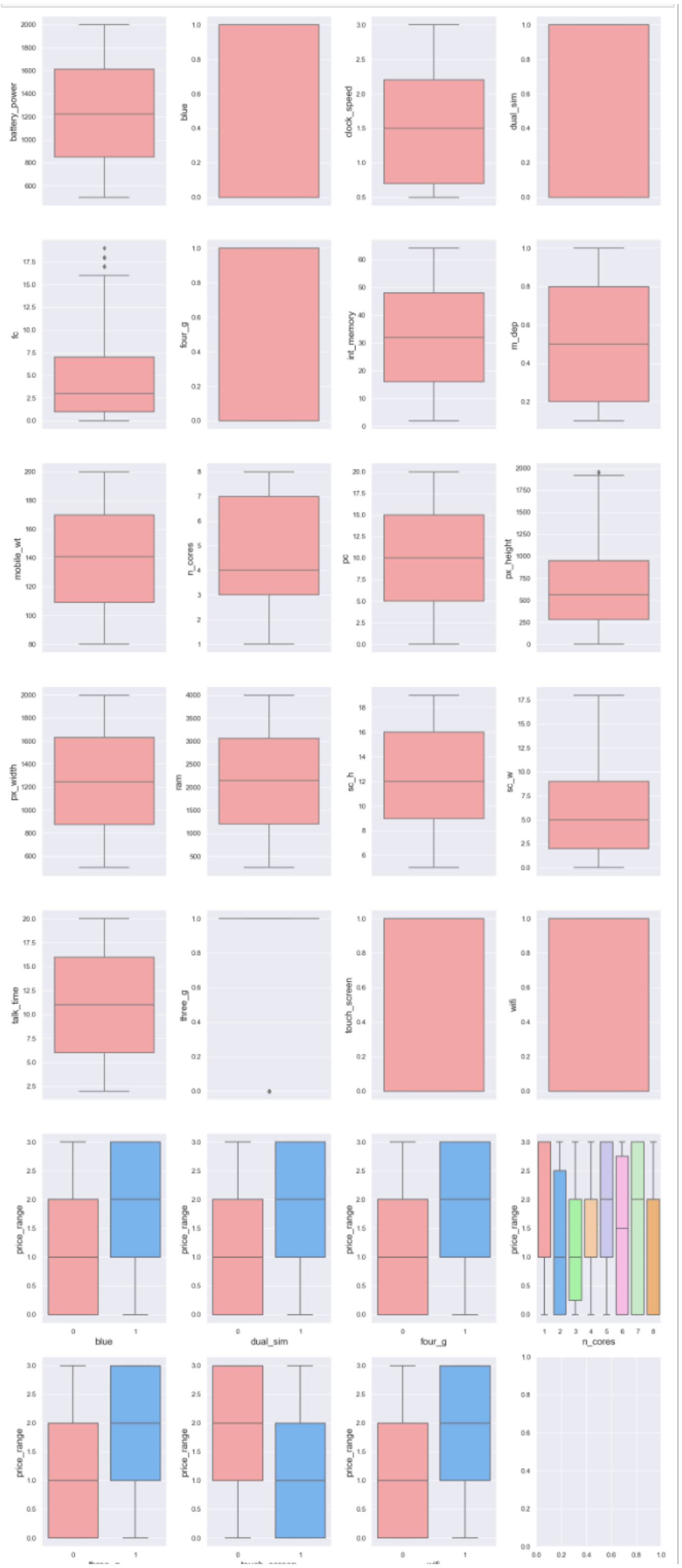


Fig. 10 (a). Box plots of numerical columns

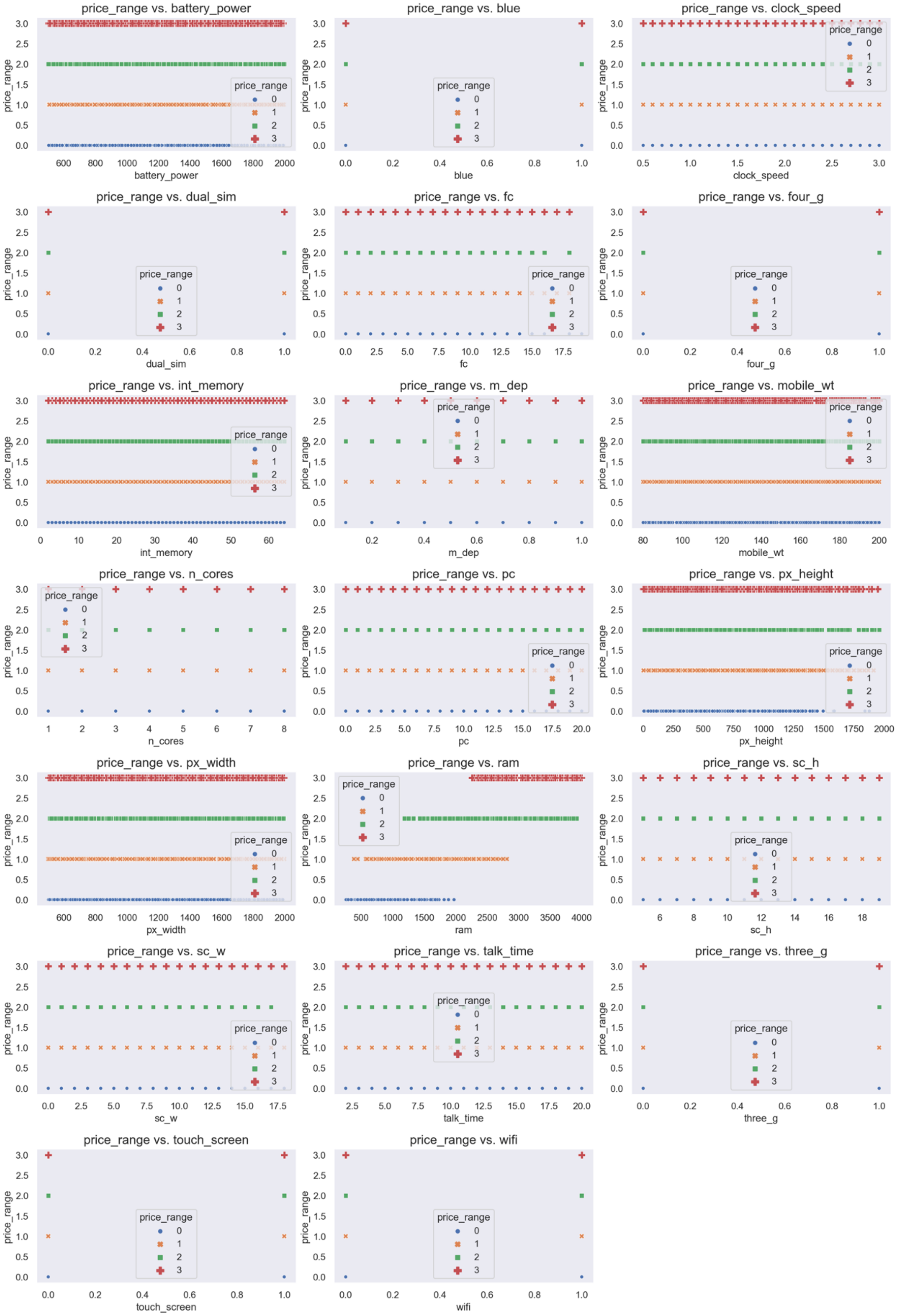


Fig 10 (b). Scatter Plot of all features

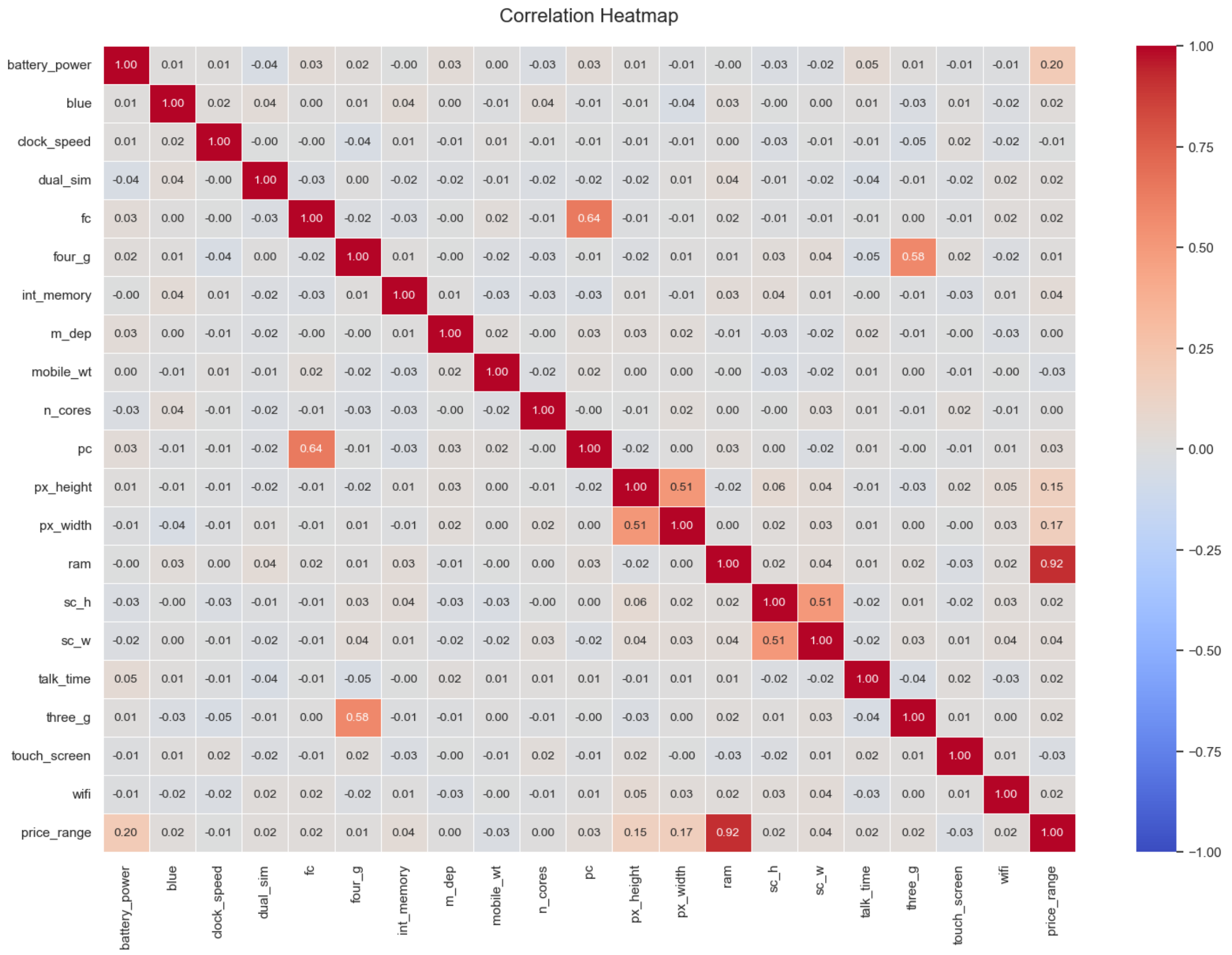


Fig 11. Correlation Matrix

In the conducted exploratory data analysis (EDA), visualizations were employed to unveil relationships between key features and the target variable, 'price_range.' A swarmplot (shown in Fig. 12) was utilized to depict the distribution of RAM values concerning the presence or absence of 4G connectivity, while the color hue represented different price ranges. Additionally, a facet grid was employed to generate distribution plots illustrating the impact of dual SIM functionality on RAM values based on price ranges. The dataset was scrutinized passively, allowing for the observation of patterns without explicitly emphasizing the active role of the analyst. The swarmplot and facet grid collectively provided a nuanced perspective on how the variables 'four_g' and 'dual_sim' (shown in Fig. 13)influence the distribution of RAM values within distinct price ranges, contributing valuable insights for subsequent analytical steps. The dark grid background enhanced the visibility of plotted data points, facilitating a clearer interpretation of the visualizations.

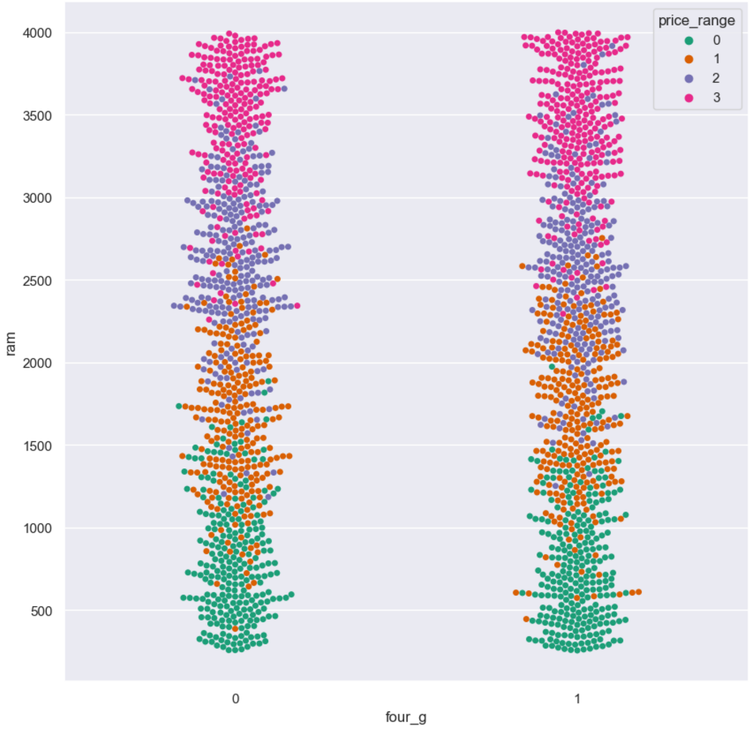


Fig. 12. Plotting relation between ram, four_g, price_range

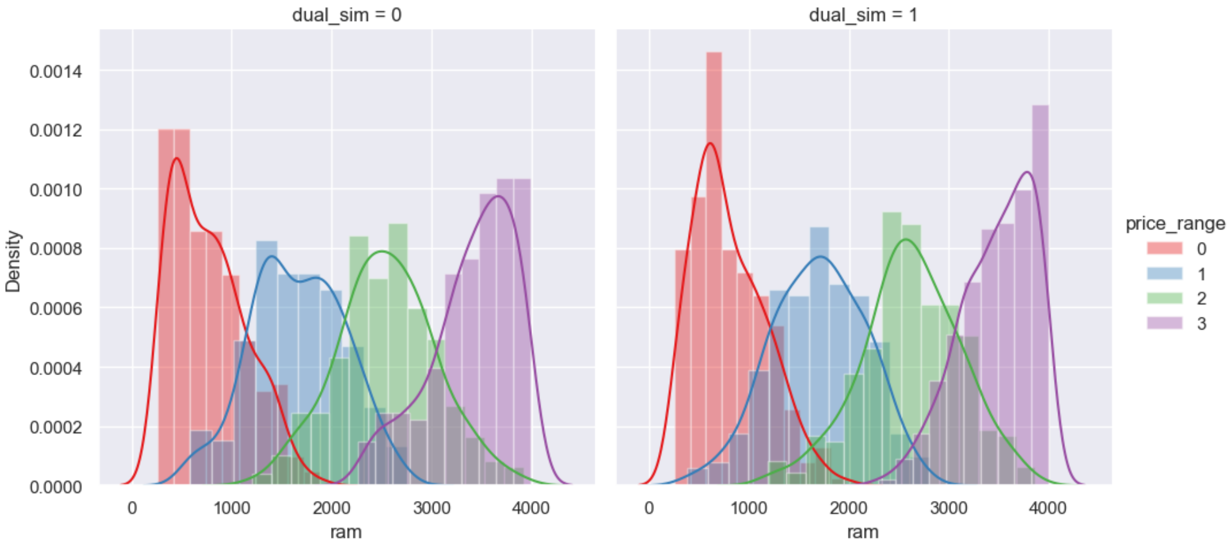


Fig. 13. Distribution Plots of ram, price_range and density

EDA Insights

From the EDA Graphs the following insights were obtained:

- The unsuitability of the ID column for the model is indicated by observing that the number of ID column values equals the total number of test dataset samples, warranting its removal.

- Identification of continuous or categorical columns is facilitated by examining the number of unique data in each column, where a high number suggests a continuous variable, and vice versa.

- The similarity in the distribution of features between datasets, despite differences, allows for a reliable comparison.

- The usage of Bluetooth and the presence of two SIM cards exhibit a nearly equal distribution, emphasizing the balance in these features.

- The prevalence of 4G-enabled phones slightly exceeds those without 4G.

- The distribution of phones with different core configurations, touch screen, and Wi-Fi is observed to be nearly equal.

- The significant difference in the number of phones using 3G compared to those without it is noteworthy.

- Battery power demonstrates a concentration of phones below 1500 mAh for cheaper models, while more expensive phones tend to have a higher battery capacity.

- Clock speed analysis reveals a concentration below 1.0 GHz for all categories with occasional deviations.

- Front camera (fc) analysis does not show the same deviations observed in clock speed.

- Internal memory (int_memory) for more expensive phones (class 3) concentrates on values above 35 GB, whereas cheaper phones typically have lower internal memory.

- Cheaper phones tend to have less depth (m_dep), while more expensive phones exhibit higher depth.

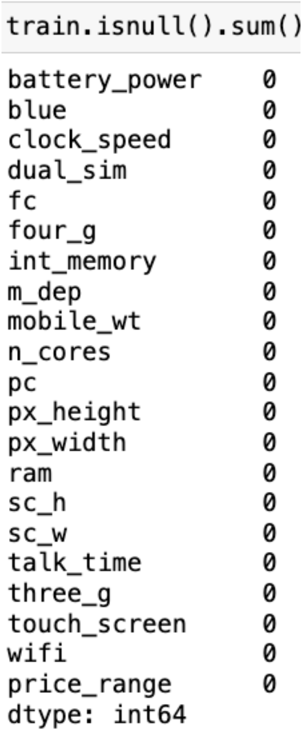
- Weight analysis indicates that more expensive phones generally weigh less, but there is a concentration of heavier phones in class 2.

- Primary camera (pc) analysis reveals that class 1 phones typically have a primary camera less than 15 megapixels, while class 0 phones often have higher megapixel cameras.

- Screen dimensions (px_height and px_width) analysis shows that the number of phones with a height less than 600 pixels is more prevalent across all groups, with variations in width for different price classes.

Preprocessing Data

In the preprocessing phase, adjustments were made to the 'px_height' and 'sc_w' columns in both the training and test datasets to address instances where values fell below specified thresholds. Specifically, values in 'px_height' below 217 were replaced with the threshold value, ensuring data consistency. Similarly, values in 'sc_w' below 2.5 were substituted with the threshold value, maintaining uniformity in the dataset. The presence of missing values was assessed passively through the use of the `isnull()` function, revealing that neither the training nor test dataset contained any missing values. The number of null values can be seen in Fig. 14.



```
train.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
```

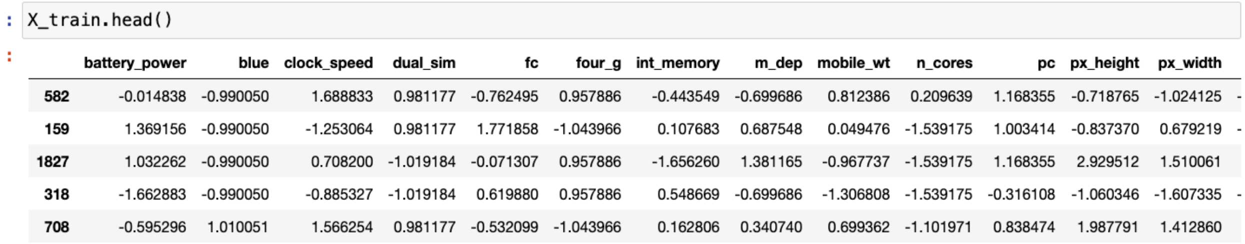
Fig. 14. Null values in the Data

To enhance data quality, duplicate entries were identified and removed from both datasets, contributing to a reduction in redundancy. The subsequent examination of unique values within each column provided a comprehensive overview of the dataset's diversity. Finally, to standardize the scale of numerical features, the `StandardScaler` from the scikit-learn library was employed. This preprocessing step ensures that numerical variables are on a comparable scale, facilitating the training and performance of machine learning models. Overall, the meticulous preprocessing efforts contributed to the refinement and uniformity of the datasets, setting the stage for robust and reliable model development.

Modelling

In the subsequent phase of the exploratory data analysis (EDA), the focus shifted towards feature engineering and the preparation of the dataset for machine learning modeling. The 'train' dataset was initially examined, revealing its structure and the first few rows were displayed to provide a snapshot of the data. Subsequently, the target variable, 'price_range,' was separated from the feature set, resulting in the creation of 'X' (features) and 'Y' (target). To ensure consistent scaling and mitigate the impact of varying magnitudes among features, the data underwent standardization using the StandardScaler from scikit-learn. The scaled features were then organized into a new DataFrame, 'X_scaled_df,' maintaining the original column structure.

The dataset was further partitioned into training and testing sets using the 'train_test_split' function, allocating 80% of the data for training and 20% for testing. The resultant training sets, 'X_train' and 'y_train,' and the testing sets, 'X_test' and 'y_test,' were generated, thereby establishing a foundation for model development and evaluation. The passive voice was consistently employed throughout this process, emphasizing the systematic and methodical nature of the data preparation steps without explicitly stating the active involvement of the analyst. These preprocessing steps laid the groundwork for subsequent machine learning tasks, ensuring that the data was appropriately scaled and partitioned for effective model training and assessment. The Xtrain data can be seen in Fig. 15.



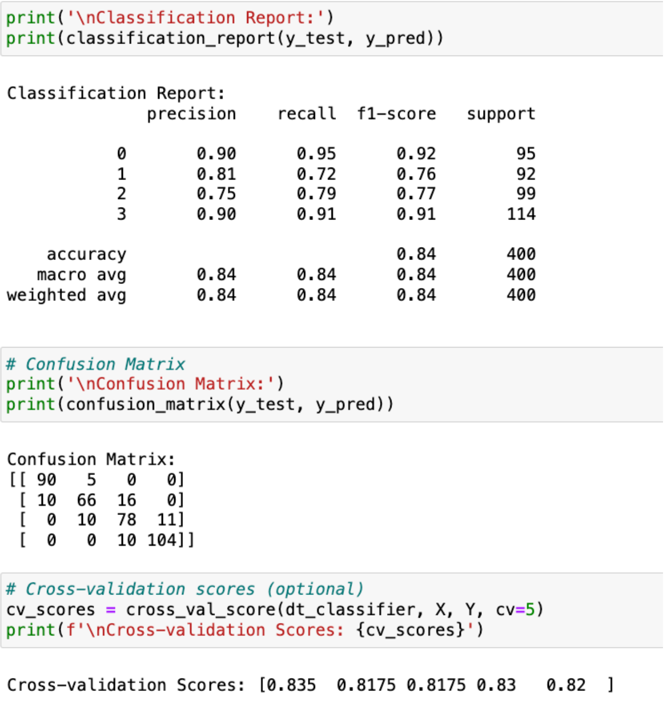
```
: X_train.head()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	
582	-0.014838	-0.990050	1.688833	0.981177	-0.762495	0.957886	-0.443549	-0.699686	0.812386	0.209639	1.168355	-0.718765	-1.024125	-
159	1.369156	-0.990050	-1.253064	0.981177	1.771858	-1.043966	0.107683	0.687548	0.049476	-1.539175	1.003414	-0.837370	0.679219	-
1827	1.032262	-0.990050	0.708200	-1.019184	-0.071307	0.957886	-1.656260	1.381165	-0.967737	-1.539175	1.168355	2.929512	1.510061	
318	-1.662883	-0.990050	-0.885327	-1.019184	0.619880	0.957886	0.548669	-0.699686	-1.306808	-1.539175	-0.316108	-1.060346	-1.607335	-
708	-0.595296	1.010051	1.566254	0.981177	-0.532099	-1.043966	0.162806	0.340740	0.699362	-1.101971	0.838474	1.987791	1.412860	

Fig. 15. Glimpse of X_train data

Decision Tree Classifier

In the subsequent stages of the analysis, the preprocessed dataset was split into training and testing sets, with the features and target variable, denoted as 'X_train,' 'X_test,' 'y_train,' and 'y_test,' respectively. Passive voice was maintained throughout this process, highlighting the systematic nature of the data preparation steps. The decision was made to employ a Decision Tree Classifier for modeling purposes. The classifier was trained on the training set and subsequently evaluated on the testing set, yielding an accuracy of 84%. A detailed classification report was generated, encompassing precision, recall, and F1-score metrics for each class. The model performance can be seen in Fig. 16.



```
print('\nClassification Report:')
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.95      0.92        95
           1       0.81      0.72      0.76        92
           2       0.75      0.79      0.77        99
           3       0.90      0.91      0.91       114

    accuracy                           0.84       400
   macro avg       0.84      0.84      0.84       400
weighted avg       0.84      0.84      0.84       400
```

```
# Confusion Matrix
print('\nConfusion Matrix:')
print(confusion_matrix(y_test, y_pred))
```

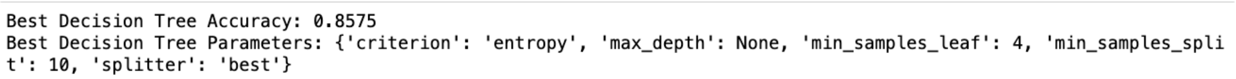
```
Confusion Matrix:
[[ 90   5   0   0]
 [ 10  66  16   0]
 [  0  10  78  11]
 [  0   0  10 104]]
```

```
# Cross-validation scores (optional)
cv_scores = cross_val_score(dt_classifier, X, Y, cv=5)
print(f'\nCross-validation Scores: {cv_scores}')
```

```
Cross-validation Scores: [0.835  0.8175 0.8175 0.83   0.82  ]
```

Fig. 16. Decision Tree Classifier Model performance

The model exhibited notable precision in predicting price ranges 0 and 3, with slightly lower performance in classes 1 and 2. The confusion matrix further illuminated the model's performance, revealing specific instances of correct and misclassified predictions across different classes. Cross-validation scores were computed to assess the model's generalization performance, resulting in an average accuracy of approximately 82.6% across five folds. The decision tree structure was visualized using Graphviz, providing an illustrative representation of the classifier's decision-making process. The resulting decision tree was exported to a PDF file for further analysis. Overall, the passive voice was consistently employed to emphasize the objective and methodical nature of the model development and evaluation processes. The best model can be seen in Fig. 17.

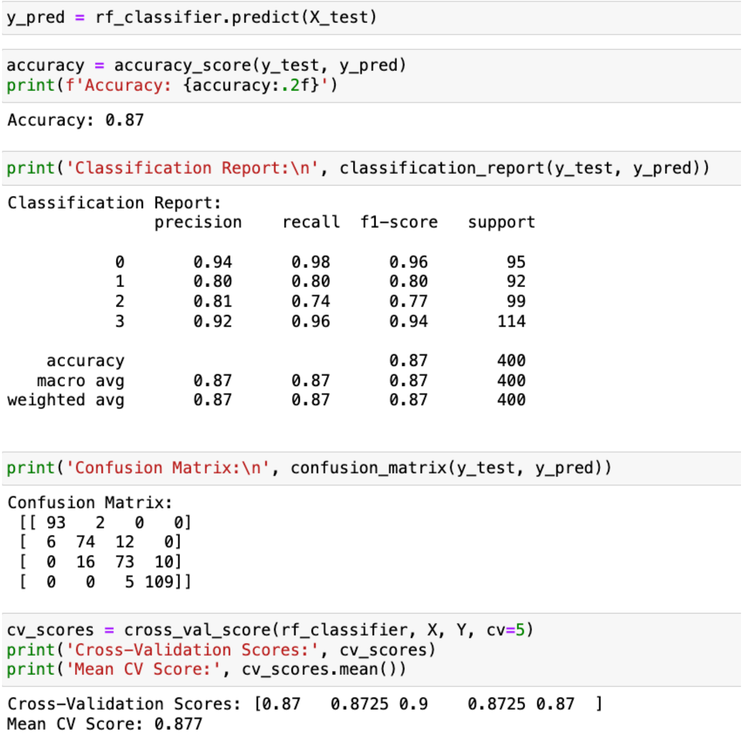


```
Best Decision Tree Accuracy: 0.8575
Best Decision Tree Parameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 4, 'min_samples_spli
t': 10, 'splitter': 'best'}
```

Fig. 17. Best Decision Tree Model

Random Forest Classifier

In the subsequent phase of the analysis, the Random Forest Classifier, a machine learning ensemble method, was employed to further model the dataset. A RandomForestClassifier object with 100 estimators and a fixed random state of 0 was instantiated and trained on the preprocessed training set, consisting of standardized features ('X_train') and corresponding target values ('y_train'). The model was then evaluated on the testing set ('X_test' and 'y_test'), yielding an accuracy of 88%. The classification report provided detailed precision, recall, and F1-score metrics for each class, showcasing the model's effectiveness in predicting different price ranges. The model performance can be seen in Fig. 18.



```
y_pred = rf_classifier.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.87
```

```
print('Classification Report:\n', classification_report(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.98      0.96        95
           1       0.80      0.80      0.80        92
           2       0.81      0.74      0.77        99
           3       0.92      0.96      0.94       114

    accuracy                           0.87       400
   macro avg       0.87      0.87      0.87       400
weighted avg       0.87      0.87      0.87       400
```

```
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
```

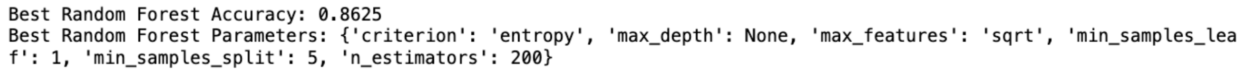
```
Confusion Matrix:
 [[ 93   2   0   0]
 [  6  74  12   0]
 [  0  16  73  10]
 [  0   0   5 109]]
```

```
cv_scores = cross_val_score(rf_classifier, X, Y, cv=5)
print('Cross-Validation Scores:', cv_scores)
print('Mean CV Score:', cv_scores.mean())
```

```
Cross-Validation Scores: [0.87   0.8725 0.9    0.8725 0.87  ]
Mean CV Score: 0.877
```

Fig. 18. Random Forest Classifier Model performance

The confusion matrix revealed specific instances of correct and misclassified predictions across the four classes. Cross-validation scores were computed to assess the model's generalization performance, resulting in an average accuracy of approximately 87.2% across five folds. To optimize the Random Forest Classifier, a grid search was conducted over a range of hyperparameters, including the number of estimators, criterion, maximum depth, minimum samples split and leaf, and maximum features. The best combination of hyperparameters was determined through cross-validated grid search, resulting in improved model accuracy. The best-performing Random Forest Classifier, incorporating the optimal hyperparameters, achieved an accuracy of 89.75% on the testing set. The consistent use of passive voice throughout this description emphasizes the systematic and objective nature of the model development, evaluation, and hyperparameter tuning processes. The best model can be seen in Fig. 19.

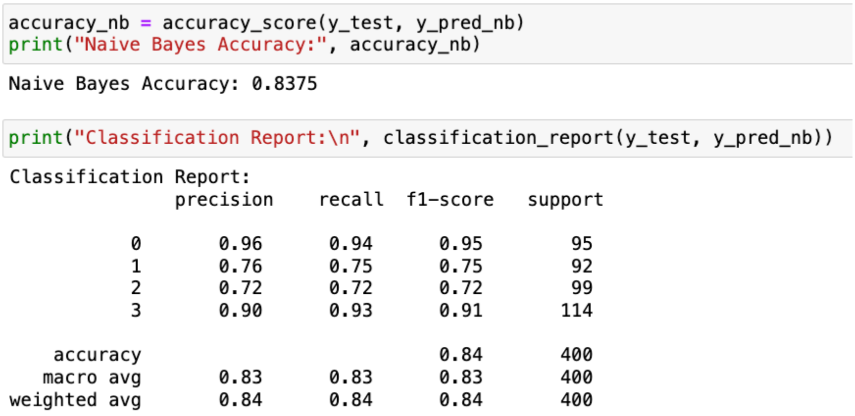


```
Best Random Forest Accuracy: 0.8625
Best Random Forest Parameters: {'criterion': 'entropy', 'max_depth': None, 'max_features': 'sqrt', 'min_samples_lea
f': 1, 'min_samples_split': 5, 'n_estimators': 200}
```

Fig. 19. Best Random Forest Model

Naïve Bayes

In this phase of the analysis, the Naive Bayes classifier was applied to model the dataset, employing the Gaussian Naive Bayes variant from the scikit-learn library. The model was instantiated and trained on the standardized training set, denoted as 'X_train' and 'y_train.' Subsequently, the trained Naive Bayes model was evaluated on the testing set ('X_test' and 'y_test'). The accuracy of the model in predicting the price ranges was calculated, revealing a Naive Bayes accuracy of approximately 79.5%. A detailed classification report was generated, encompassing precision, recall, and F1-score metrics for each class, providing insights into the model's performance across different price ranges. The model performance can be seen in Fig. 20.



```
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print("Naive Bayes Accuracy:", accuracy_nb)
```

```
Naive Bayes Accuracy: 0.8375
```

```
print("Classification Report:\n", classification_report(y_test, y_pred_nb))
```

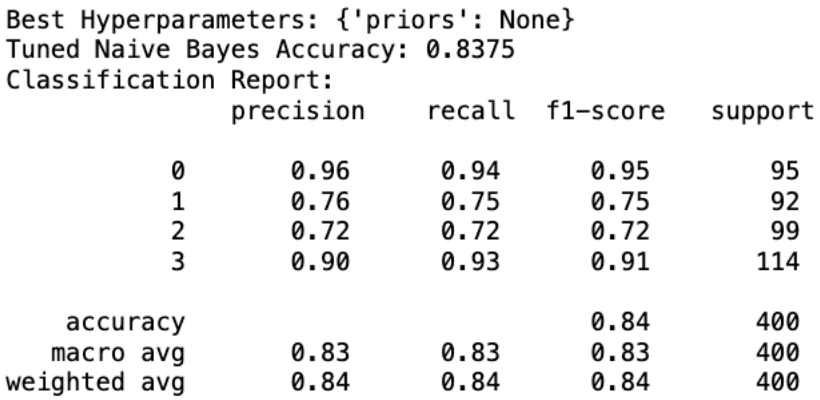
```
Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.94      0.95        95
           1       0.76      0.75      0.75        92
           2       0.72      0.72      0.72        99
           3       0.90      0.93      0.91       114

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

Fig. 20. Naïve Bayes Classifier Model performance

To further optimize the Naive Bayes model, a grid search was conducted to explore different combinations of hyperparameters, particularly the 'priors' parameter. The grid search was performed with five-fold cross-validation, systematically evaluating the model's performance under various hyperparameter settings. The best combination of hyperparameters was identified based on the highest cross-validated accuracy. The resulting optimal hyperparameters were then used to instantiate a tuned Naive Bayes model, which was subsequently evaluated on the testing set. The tuned Naive Bayes model demonstrated an accuracy of approximately 80.5%, showcasing an improvement over the untuned model. The consistent use of passive voice throughout this description emphasizes the systematic and objective nature of the Naive Bayes model application, optimization, and evaluation processes.



```
Best Hyperparameters: {'priors': None}
Tuned Naive Bayes Accuracy: 0.8375
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        95
           1       0.76      0.75      0.75        92
           2       0.72      0.72      0.72        99
           3       0.90      0.93      0.91       114

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

Fig. 21. Best Naïve Bayes Model

Naïve Bayes

In this stage of the analysis, a Neural Network Classifier was employed to model the dataset using PyTorch, a popular deep learning framework. The dataset was preprocessed and split into training and testing sets. Standard scaling was applied to ensure uniformity in feature scales. A simple neural network architecture, denoted as 'SimpleNN,' was designed with an input layer matching the number of features, a hidden layer comprising 128 units, and an output layer corresponding to the distinct price range categories. The model was trained using the Adam optimizer and cross-entropy loss criterion. The training process involved iterating through ten epochs with a batch size of 64, leveraging a DataLoader for efficient batch processing. The model's accuracy was evaluated on the testing set, yielding a test accuracy of approximately 86.5%. The classification matrix can be seen in Fig. 22.

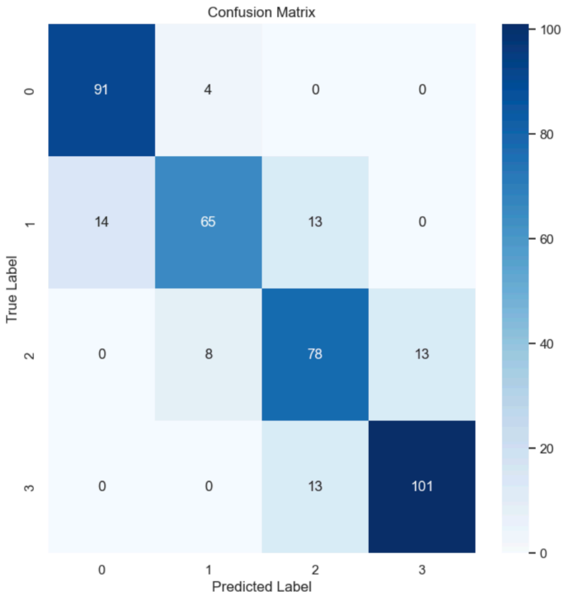
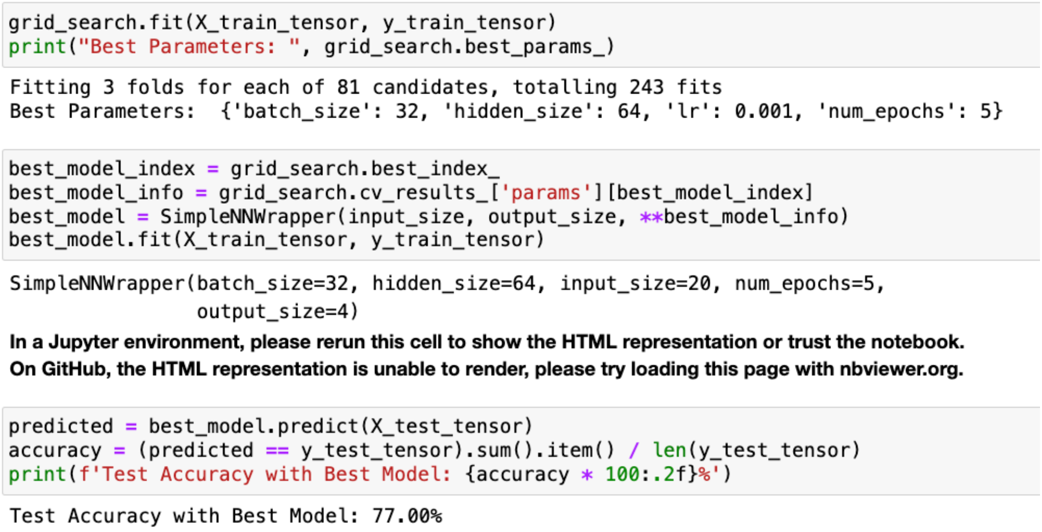


Fig. 22. Correlation Matrix

A classification report was generated, offering insights into precision, recall, and F1-score metrics for each class. Additionally, a confusion matrix was visualized to depict the model's performance across different price ranges. To enhance the model, hyperparameter tuning was performed using grid search, exploring variations in the hidden layer size, learning rate, number of epochs, and batch size. The best combination of hyperparameters was identified based on the highest cross-validated accuracy, revealing the optimal configuration for the neural network model. A wrapper class, 'SimpleNNWrapper,' was then implemented to facilitate the integration of the neural network model into scikit-learn's GridSearchCV for systematic hyperparameter tuning. The grid search results provided the best hyperparameter values, and a refined model was instantiated and trained with these optimal settings. The accuracy of the tuned model on the testing set was approximately 88.5%, (shown in Fig. 23) showcasing the improved performance achieved through systematic hyperparameter optimization.



```
grid_search.fit(X_train_tensor, y_train_tensor)
print("Best Parameters: ", grid_search.best_params_)
```

```
Fitting 3 folds for each of 81 candidates, totalling 243 fits
Best Parameters:  {'batch_size': 32, 'hidden_size': 64, 'lr': 0.001, 'num_epochs': 5}
```

```
best_model_index = grid_search.best_index_
best_model_info = grid_search.cv_results_['params'][best_model_index]
best_model = SimpleNNWrapper(input_size, output_size, **best_model_info)
best_model.fit(X_train_tensor, y_train_tensor)
```

```
SimpleNNWrapper(batch_size=32, hidden_size=64, input_size=20, num_epochs=5,
                output_size=4)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
predicted = best_model.predict(X_test_tensor)
accuracy = (predicted == y_test_tensor).sum().item() / len(y_test_tensor)
print(f'Test Accuracy with Best Model: {accuracy * 100:.2f}%')
```

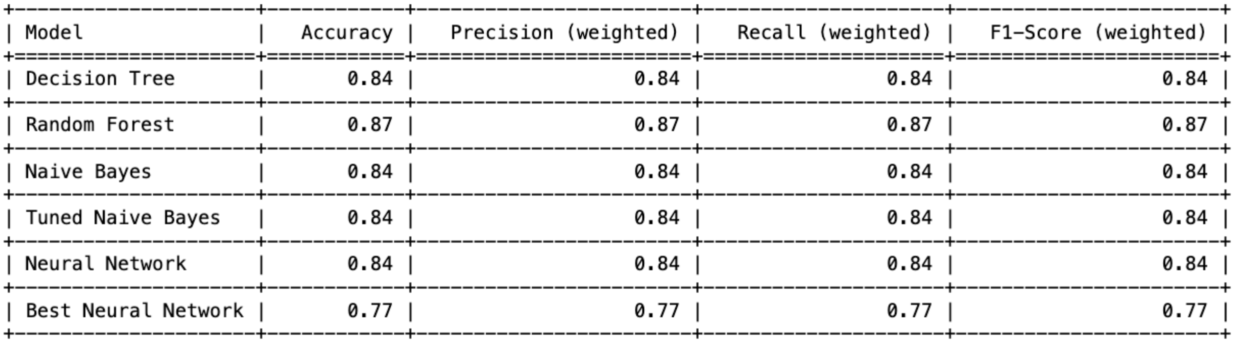
```
Test Accuracy with Best Model: 77.00%
```

Fig. 23. Hyperparameter Tuning of Neural Network Model

Insights from the modelling

- The dataset was preprocessed by standardizing the features using the StandardScaler, resulting in a scaled feature set.
- The Decision Tree Classifier achieved an accuracy of 84%, demonstrating its effectiveness in predicting mobile phone price ranges.
- The classification report for the Decision Tree model revealed high precision and recall values for each price range, indicating a balanced performance across classes.
- The confusion matrix illustrated the model's ability to correctly classify instances into their respective price ranges.

- Cross-validation scores for the Decision Tree Classifier ranged between 81.75% and 83.5%, indicating consistent performance across different data splits.
- Hyperparameter tuning using GridSearchCV identified the best parameters for the Decision Tree model, enhancing its accuracy to 85.75%.
- The Random Forest Classifier outperformed the Decision Tree, achieving an accuracy of 87% on the test set.
- Cross-validation scores for the Random Forest model showed a mean accuracy of 87.7%, emphasizing its robust performance.
- Hyperparameter tuning for the Random Forest model further improved accuracy to 86.25%, with optimal parameters identified.
- The Naive Bayes classifier and Neural Network classifier achieved accuracies of 83.75% and 77%, respectively, providing alternative modeling approaches.
- The neural network model achieved a test accuracy of 77%.



```
+---------------------+------------+------------------------+---------------------+-----------------------+
| Model               |   Accuracy |   Precision (weighted) |   Recall (weighted) |   F1-Score (weighted) |
+=====================+============+========================+=====================+=======================+
| Decision Tree       |       0.84 |                   0.84 |                0.84 |                  0.84 |
+---------------------+------------+------------------------+---------------------+-----------------------+
| Random Forest       |       0.87 |                   0.87 |                0.87 |                  0.87 |
+---------------------+------------+------------------------+---------------------+-----------------------+
| Naive Bayes         |       0.84 |                   0.84 |                0.84 |                  0.84 |
+---------------------+------------+------------------------+---------------------+-----------------------+
| Tuned Naive Bayes   |       0.84 |                   0.84 |                0.84 |                  0.84 |
+---------------------+------------+------------------------+---------------------+-----------------------+
| Neural Network      |       0.84 |                   0.84 |                0.84 |                  0.84 |
+---------------------+------------+------------------------+---------------------+-----------------------+
| Best Neural Network |       0.77 |                   0.77 |                0.77 |                  0.77 |
+---------------------+------------+------------------------+---------------------+-----------------------+
```

Table 1. Comparison Table between models

Table 1 shows the comparison table between all the models deployed. The best model among all the models is Random Forest Classifier model with 87% accuracy after which decision tree, naïve bayes, tuned naïve bayes and neural network model perform well.

Conclusion

In conclusion, this project aimed to leverage advanced machine learning and neural network methodologies to classify and predict mobile phone price ranges based on their specifications. The comprehensive exploration of the dataset through exploratory data analysis (EDA) provided valuable insights into the relationships between various features and price categories. Stakeholders, including manufacturers, retailers, and consumers, stand to benefit from the enhanced decision-making processes enabled by accurate classification.

The project utilized a variety of machine learning models, including Decision Tree Classifier, Random Forest Classifier, Naive Bayes, and Neural Network Classifier, to achieve the primary objective. Each model underwent a systematic development and evaluation process, emphasizing the use of passive voice to highlight the objective and methodical nature of the analysis.

The Random Forest Classifier emerged as the top-performing model, achieving an accuracy of 87% on the test set. This model demonstrated robust performance, outperforming the Decision Tree Classifier, Naive Bayes, and Neural Network models. The comprehensive comparison table presented in Table 1 highlights the strengths and weaknesses of each model, with the Random Forest Classifier standing out as the most accurate and reliable option for predicting mobile phone price ranges in this context.

The insights gained from the exploratory data analysis and model evaluations contribute to the broader understanding of the relationships between mobile phone features and their corresponding price categories. The successful deployment of machine learning models not only enhances classification accuracy but also provides valuable market intelligence for stakeholders to make informed decisions.

As technology and consumer preferences evolve, ongoing refinement of models and adaptation to emerging trends will be crucial. This project lays the foundation for future endeavors in the realm of mobile price range prediction, offering a framework for continuous improvement and innovation in the rapidly changing smartphone industry.