

Capstone Project-3 Mobile Price Range Prediction

Team Members

Sachin S Panchal Shreedarsh M Ranjith K Rohan A G Kshipra Parihar

Contents:



- 1. Data Exploration
- 2. Data Pre-processing
- **3.Data Cleaning**
- **4.Univariate Analysis**
- **5.**Multivariate Analysis
- **6.Outlier Treatment**
- 7. Feature Engineering
- 8. Standardization
- 9.K nearest neighbours model
- 10.Logistic regression model
- 11.XGBoost Classifier
- 12. Hyper parameter tuning
- 13. Model Explainability
- 14.Conclusion

Mobile Price Range Prediction



In the competitive mobile phone market companies want to understand sales data of mobile phones and factors which drive the prices. The objective is to find out some relation between features of a mobile phone(eg:- RAM, Internal Memory, etc) and its selling price. In this problem, we do not have to predict the actual price but a price range indicating how high the price is.



Attribute Information



Battery_power - Total energy a battery can store in one time measured 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 mega pixels

Four_g - Has 4G or not

Int_memory - Internal Memory in Gigabytes

M_dep - Mobile Depth in cm

Mobile_wt - Weight of mobile phone

N_cores - Number of cores of processor

Pc - Primary Camera mega pixels

Px_height - Pixel Resolution Height

Px_width - Pixel Resolution Width
Ram - Random Access Memory in Mega Bytes

Sc_h - Screen Height of mobile in cm

Sc_w - Screen Width of mobile in cm

Talk_time - longest time that a single battery charge will last when you are

Three_g - Has 3G or not

Touch_screen - Has touch screen or not

Wifi - Has wifi or not

Price_range - This is the target variable with value of O(low cost), I(medium cost), I(medium cost), I(medium cost)

2(high cost) and 3(very high cost).



Exploring the dataset

ba	ttery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	 px_height	px_width	ram	sc_h	SC_W	talk_time	three_g	touch_screen	wifi	price_range
0	842	0	2.2	0	1	0	7	0.6	188	2	 20	756	2549	9	7	19	0	0	1	1
1	1021	1	0.5	1	0	1	53	0.7	136	3	 905	1988	2631	17	3	7	1	1	0	2
2	563	1	0.5	1	2	1	41	0.9	145	5	 1263	1716	2603	11	2	9	1	1	0	2
3	615	1	2.5	0	0	0	10	8.0	131	6	 1216	1786	2769	16	8	11	1	0	0	2
4	1821	1	1.2	0	13	1	44	0.6	141	2	 1208	1212	1411	8	2	15	1	1	0	1

5 rows × 21 columns

df.tail()

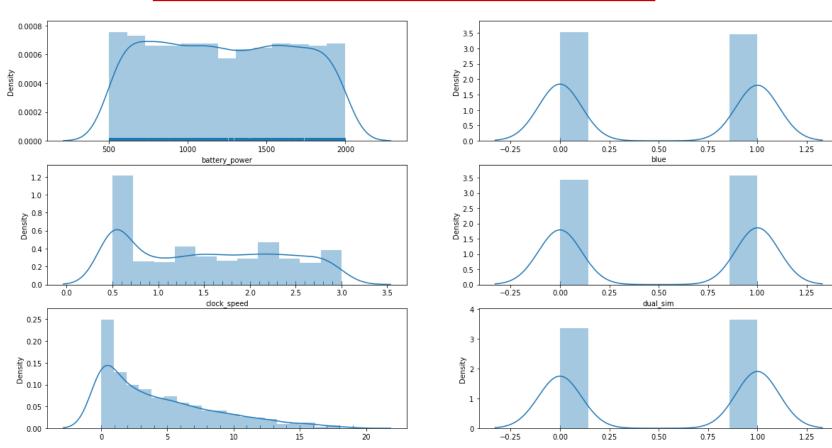
•		battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	 px_height	px_width	ram	sc_h	SC_W	talk_time	three_g	touch_screen	wifi	price_rar	nge
1	995	794	1	0.5	1	0	1	2	8.0	106	6	 1222	1890	668	13	4	19	1	1	0		0
1	996	1965	1	2.6	1	0	0	39	0.2	187	4	 915	1965	2032	11	10	16	1	1	1		2
1	997	1911	0	0.9	1	1	1	36	0.7	108	8	 868	1632	3057	9	1	5	1	1	0		3
1	998	1512	0	0.9	0	4	1	46	0.1	145	5	 336	670	869	18	10	19	1	1	1		0
1	999	510	1	2.0	1	5	1	45	0.9	168	6	 483	754	3919	19	4	2	1	1	1		3

5 rows × 21 columns

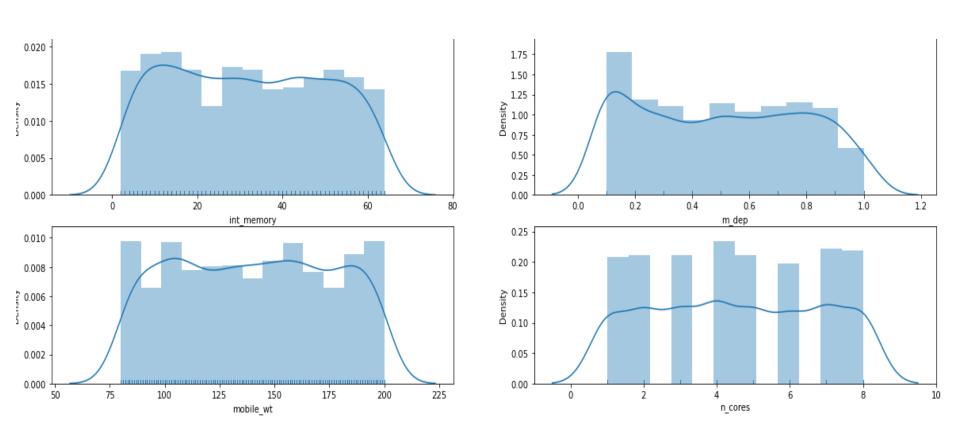
[45] df.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 2000 entries, 0 to 1999 Data columns (total 21 columns): Non-Null Count Dtype battery power 2000 non-null int64 2000 non-null int64 2000 non-null float64 clock_speed dual sim 2000 non-null int64 4 fc 2000 non-null four_g 2000 non-null 6 int memory 2000 non-null int64 7 m_dep 2000 non-null float64 8 mobile wt 2000 non-null 9 n_cores 2000 non-null 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 14 sc h 2000 non-null int64

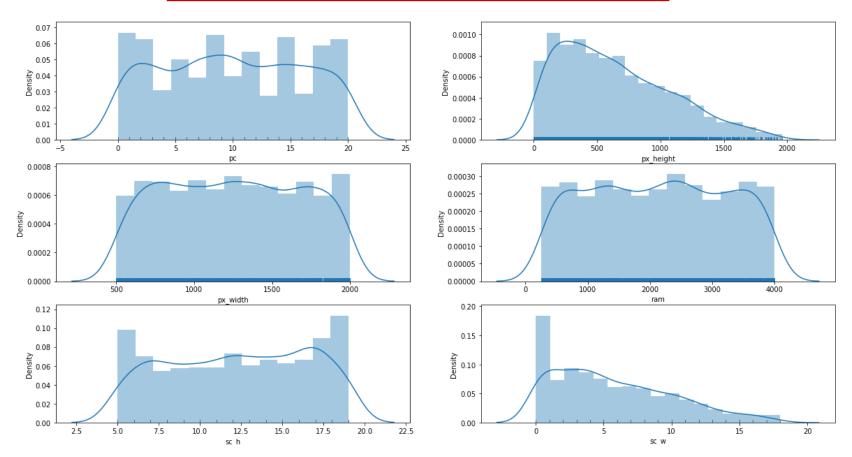




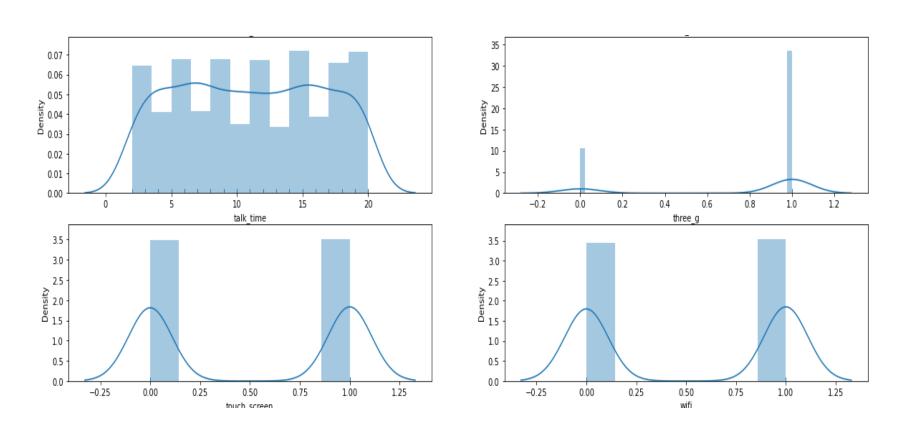




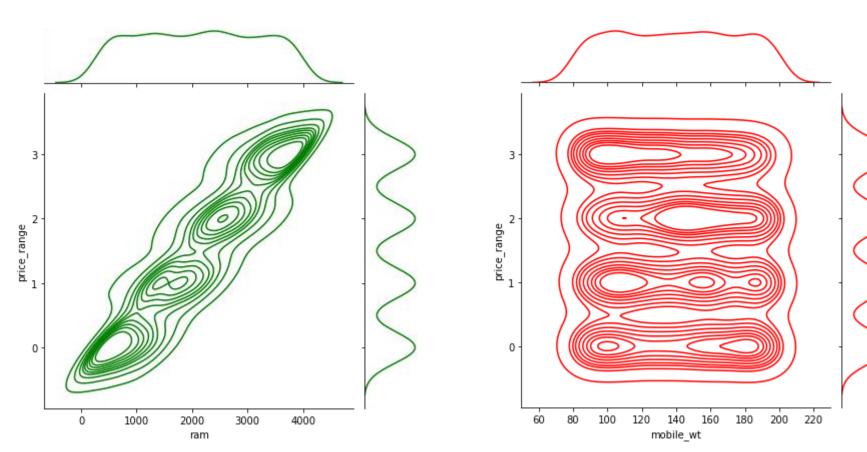




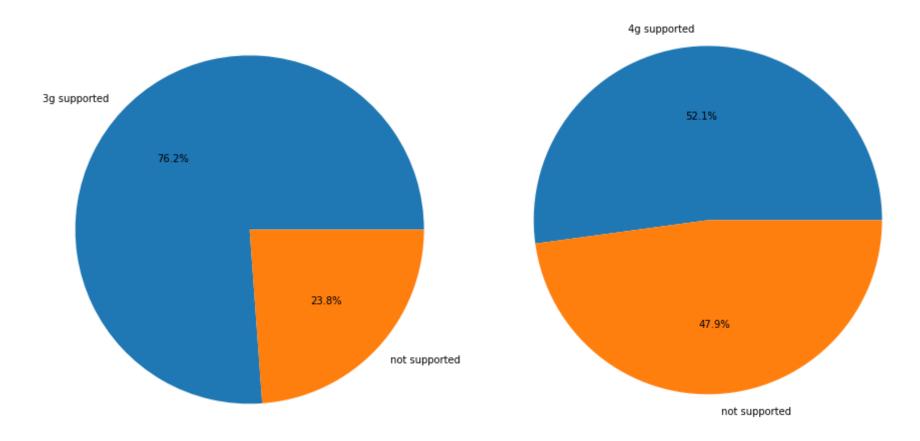




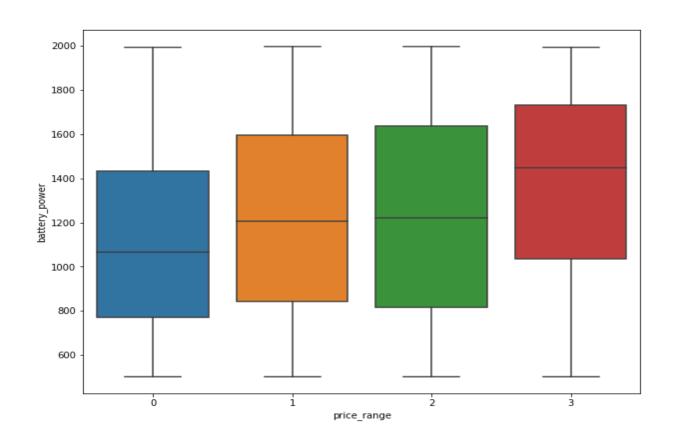




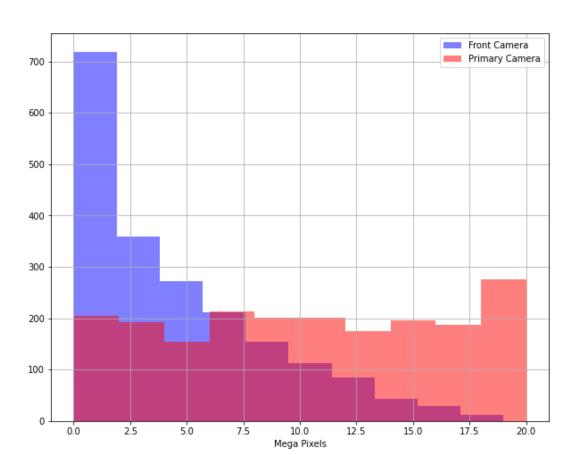






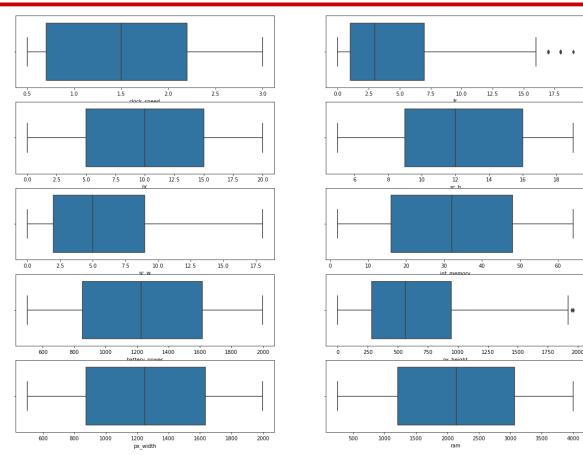








Outliers detection and treatment



Correlation Heatmap

ΔΙ

battery_power	1	0.011	0.011	-0.042	0.033	0.016	-0.004	0.034	0.0018	-0.03	0.031	0.015	-0.0084	0.00065	-0.03	-0.021	0.053	0.012	-0.011	-0.0083	0.2
blue -	0.011	1	0.021	0.035	0.0036	0.013	0.041	0.004	-0.0086	0.036	-0.01	-0.0069	-0.042	0.026	-0.003	0.00061	0.014	-0.03	0.01	-0.022	0.021
clock_speed -	0.011	0.021	1	-0.0013	-0.00043	-0.043	0.0065	-0.014	0.012	-0.0057	-0.0052	-0.015	-0.0095	0.0034	-0.029	-0.0074	-0.011	-0.046	0.02	-0.024	-0.0066
dual_sim -	-0.042	0.035	-0.0013	1	-0.029	0.0032	-0.016	-0.022	-0.009	-0.025	-0.017	-0.021	0.014	0.041	-0.012	-0.017	-0.039	-0.014	-0.017	0.023	0.017
fc -	0.033	0.0036	-0.00043	-0.029	1	-0.017	-0.029	-0.0018	0.024	-0.013	0.64	-0.01	-0.0052	0.015	-0.011	-0.012	-0.0068	0.0018	-0.015	0.02	0.022
four_g -	0.016	0.013	-0.043	0.0032	-0.017	1	0.0087	-0.0018	-0.017	-0.03	-0.0056	-0.019	0.0074	0.0073	0.027	0.037	-0.047	0.58	0.017	-0.018	0.015
int_memory	-0.004	0.041	0.0065	-0.016	-0.029	0.0087	1	0.0069	-0.034	-0.028	-0.033	0.01	-0.0083	0.033	0.038	0.012	-0.0028	-0.0094	-0.027	0.007	0.044
m_dep -	0.034	0.004	-0.014	-0.022	-0.0018	-0.0018	0.0069	1	0.022	-0.0035	0.026	0.025	0.024	-0.0094	-0.025	-0.018	0.017	-0.012	-0.0026	-0.028	0.00085
mobile_wt	0.0018	-0.0086	0.012	-0.009	0.024	-0.017	-0.034	0.022	1	-0.019	0.019	0.00094	9e-05	-0.0026	-0.034	-0.021	0.0062	0.0016	-0.014	0.00041	-0.03
n_cores -	-0.03	0.036	-0.0057	-0.025	-0.013	-0.03	-0.028	-0.0035	-0.019	1	-0.0012	-0.0069	0.024	0.0049	-0.00031	0.026	0.013	-0.015	0.024	-0.01	0.0044
pc -	0.031	-0.01	-0.0052	-0.017	0.64	-0.0056	-0.033	0.026	0.019	-0.0012	1	-0.018	0.0042	0.029	0.0049	-0.024	0.015	-0.0013	-0.0087	0.0054	0.034
px_height -	0.015	-0.0069	-0.015	-0.021	-0.01	-0.019	0.01	0.025	0.00094	-0.0069	-0.018	1	0.51	-0.02	0.06	0.043	-0.011	-0.031	0.022	0.052	0.15
px_width -	-0.0084	-0.042	-0.0095	0.014	-0.0052	0.0074	-0.0083	0.024	9e-05	0.024	0.0042	0.51	1	0.0041	0.022	0.035	0.0067	0.00035	-0.0016	0.03	0.17
ram -	-0.00065	0.026	0.0034	0.041	0.015	0.0073	0.033	-0.0094	-0.0026	0.0049	0.029	-0.02	0.0041	1	0.016	0.036	0.011	0.016	-0.03	0.023	0.92
sc_h ·	-0.03	-0.003	-0.029	-0.012	-0.011	0.027	0.038	-0.025	-0.034	0.00031	0.0049	0.06	0.022	0.016	1	0.51	-0.017	0.012	-0.02	0.026	0.023
sc_w -	-0.021	0.00061	-0.0074	-0.017	-0.012	0.037	0.012	-0.018	-0.021	0.026	-0.024	0.043	0.035	0.036	0.51	1	-0.023	0.031	0.013	0.035	0.039
talk_time -	0.053	0.014	-0.011	-0.039	-0.0068	-0.047	-0.0028	0.017	0.0062	0.013	0.015	-0.011	0.0067	0.011	-0.017	-0.023	1	-0.043	0.017	-0.03	0.022
three_g -	0.012	-0.03	-0.046	-0.014	0.0018	0.58	-0.0094	-0.012	0.0016	-0.015	-0.0013	-0.031	0.00035	0.016	0.012	0.031	-0.043	1	0.014	0.0043	0.024
touch screen	-0.011	0.01	0.02	-0.017	-0.015	0.017	-0.027	-0.0026	-0.014	0.024	-0.0087	0.022	-0.0016	-0.03	-0.02	0.013	0.017	0.014	1	0.012	-0.03
wifi -		-0.022		0.023	0.02	-0.018		-0.028					0.03	0.023	0.026	0.035	-0.03		0.012	1	0.019
price range -	0.2		-0.0066		0.022	0.015		0.00085		0.0044	0.034	0.15	0.17	0.92	0.023	0.039	0.022	0.024	-0.03	0.019	1
price_range		— , —		,	بار.			— , —			<u>.</u>			-,-	e,				_		, ,
	tery_power	plue	ock_speed	dual_sim	-	four_g	ıt_memory	m_dep	mobile_wt	n_cores		px_height	px_width	men	×,	» S	talk_time	three_g	uch_screen	wifi	rice_range

- 1.0

- 0.8

. 0 6

- 0.0

Standardization



Standardization is an important technique that is mostly performed as a preprocessing step before many Machine Learning models, to standardize the range of features of input data set.

Satandardization

```
[68] scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.fit_transform(X_test)
```

Al

Machine Learning Modelling

1. K Nearest Neighbours

- •K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- •K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- •K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- •K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- •K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.

▼ 1.K Nearest Neighbors

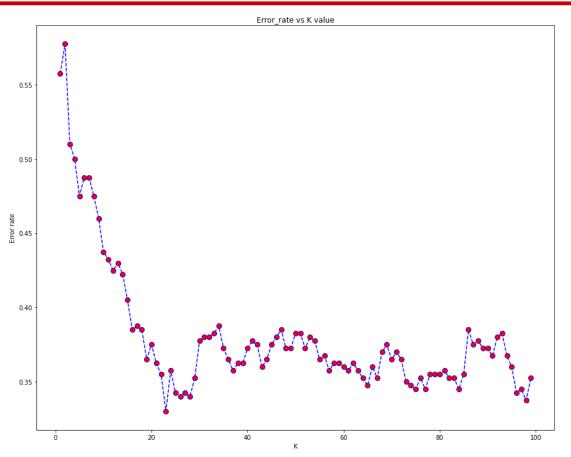
[] from sklearn.neighbors import KNeighborsClassifier knn = KNeighborsClassifier(n_neighbors=10) knn.fit(X_train,y_train)

KNeighborsClassifier(n_neighbors=10)

- knn.score(X_test,y_test)
- 0.5625



Elbow method for least error rate





Machine Learning Modelling

2. Logistic Regression

- •Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- •Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- •Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
- •In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

```
[ ] from sklearn.linear_model import LogisticRegression
    logmodel = LogisticRegression()
    logmodel.fit(X_train, y_train)

    LogisticRegression()

[ ] logmodel.score(X_test,y_test)
    0.9125

[ ] y_pred_train2 = logmodel.predict(X_train)
    y_pred_test2 = logmodel.predict(X_test)
```



Machine Learning Modelling

3.XGBoost Classifier

XGBoost is a decision-tree-based ensemble Machine Learning algorithm
that uses a gradient boosting framework. In prediction problems involving
unstructured data (images, text, etc.) artificial neural networks tend to
outperform all other algorithms or frameworks.

▼ XGBoost classifier

```
[ ] from xgboost import XGBClassifier
    xgbmodel = XGBClassifier(random_state=0)
    xgbmodel.fit(X_train,y_train)

XGBClassifier(objective='multi:softprob')

[ ] y_pred_train = xgbmodel.predict(X_train)
    y_pred_test = xgbmodel.predict(X_test)

[ ] xgbmodel.score(X_test,y_test)
    0.8925
```



Machine Learning Modelling Hyper-parameter Tuning

Hperparameter tuning for xgboost model

```
n estimators = list(np.arange(5,20,2,dtype='int64'))
                                                            #Number of Trees
    max depth = list(np.arange(10,25,1,dtype='int64'))
                                                            #Max depth of trees
    learning rate = list(np.arange(0.03,0.20,0.01))
                                                            #Learning rate
    gamma = list(np.arange(10,20,1,dtype='int64'))
                                                            #gamma
    subsample = [0.3, 0.4, 0.5, 0.6]
                                                            #subsamples
[ ] #Hyperparameters assigning
    param dict = {'n estimators' : n estimators,
                   'max depth' : max depth,
                   'learning rate' : learning rate,
                   'gamma':gamma,
                   'subsample':subsample}
[ ] #Randomized Search CV
    xgb randomized = RandomizedSearchCV(estimator=xgbmodel, param distributions=param dict, cv=5,scoring = 'accuracy', random state = 0)
    xgb randomized.fit(X train,y train)
    RandomizedSearchCV(cv=5, estimator=XGBClassifier(objective='multi:softprob'),
                       param distributions={'gamma': [10, 11, 12, 13, 14, 15, 16,
                                                      17, 18, 19],
                                             'learning_rate': [0.03, 0.04, 0.05,
                                                               0.06000000000000000005.
```



Machine Learning Modelling Hyper-parameter Tuning

```
[ ] xgb_randomized.best_params_

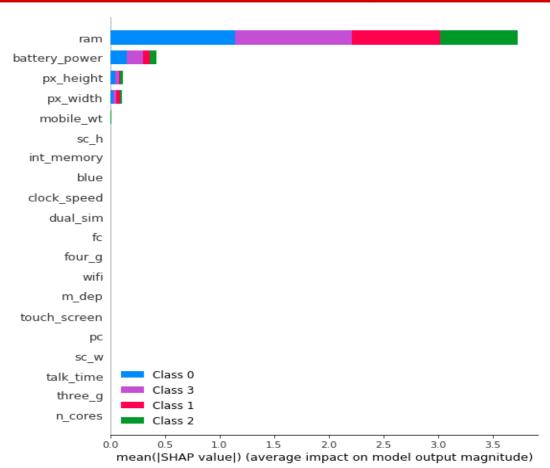
{'gamma': 12,
    'learning_rate': 0.13,
    'max_depth': 13,
    'n_estimators': 17,
    'subsample': 0.6}

[ ] xgb_final_model=xgb_randomized.best_estimator_

[ ] # predicting on both train and test
    y_pred_train3=xgb_final_model.predict(X_train)
    y_pred_test3=xgb_final_model.predict(X_test)
```

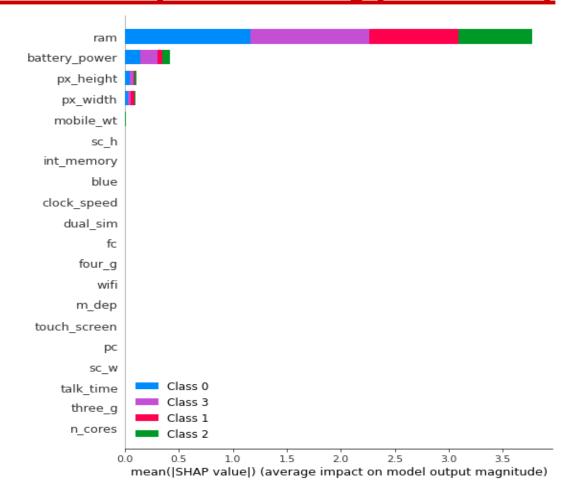
Αl

Model Explainability(Training Set)



Model Explainability(Test Set)





Conclusion



- 1. The 'price range' of the given dataset has equal distribution of the total number of phones in each of the price range with 500 nos.
- 2. It isobserved that 76.2 percent are 3g supported and 27.8 percent are not supported.
- 3. It is observed that 52.1 percent are 4g supported and 47.9 percent are not supported.
- 4. There is only few number of outlier in 'fc' column in feature engineering and we can neglect it as it has negligible amount.
- 5. During Multivariate analysis, in correlation heatmap, we get to see that 'ram is highly correlated with 'price range' thus inferring that 'ram' has high impact on price prediction.
- 6. In K nearest Neighbors classification model, we have got the knn score as 56.25%, accuracy score of 67% for training set and 65% for test set.

Conclusion



- 7. During 'elbow method' we have got the insight that the optimum value of k is 22 with least error rate.
- 8. In Logistic Regression Model, we have got the log score as 91%. accuracy score of 98% for training set and 91% for test set.
- 9. In XGBoost model the score was 89% before hyper parameter tuning.
- 10. RandomizedSearchCV is used for hyperparameter tuning in XGBoost classifier and the accuracy obtained after hyper parameter tuning was 86% for training set and 80% for test set.
- 11. Finally, in the model explaininabilty we have used shap and we got the insight that 'ram', 'battery power', and phone dimensions are the features which is deciding as key factor for the price range prediction.

References



- 1). https://pandas.pydata.org/
- 2). https://matplotlib.org/
- 3). https://seaborn.pydata.org/
- 4). Geek for geeks
- 5). Analytics Vindhya
- 6).XGBoost Documentation



Thank You