intel.
Introducing
Intel® Unnati Industrial Training – Summer 2023
An Intel® Unnati Community Initiative

INTEL UNNATI SUMMER INDUSTRIAL TRAINING REPORT

ON

SMART MOBILE PHONE PRICE PREDICTION USING MACHINE LEARNING

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

SREE VENKATESWARA COLLEGE OF ENGINEERING

(Approved by AICTE, New Delhi and Affiliated to Jawaharlal Nehru Technological University – Anantapur)

GOLDEN NAGAR, NH5 BYPASS ROAD, NORTH RAJUPALEM, KODAVALURU (V&M), SPSR NELLORE

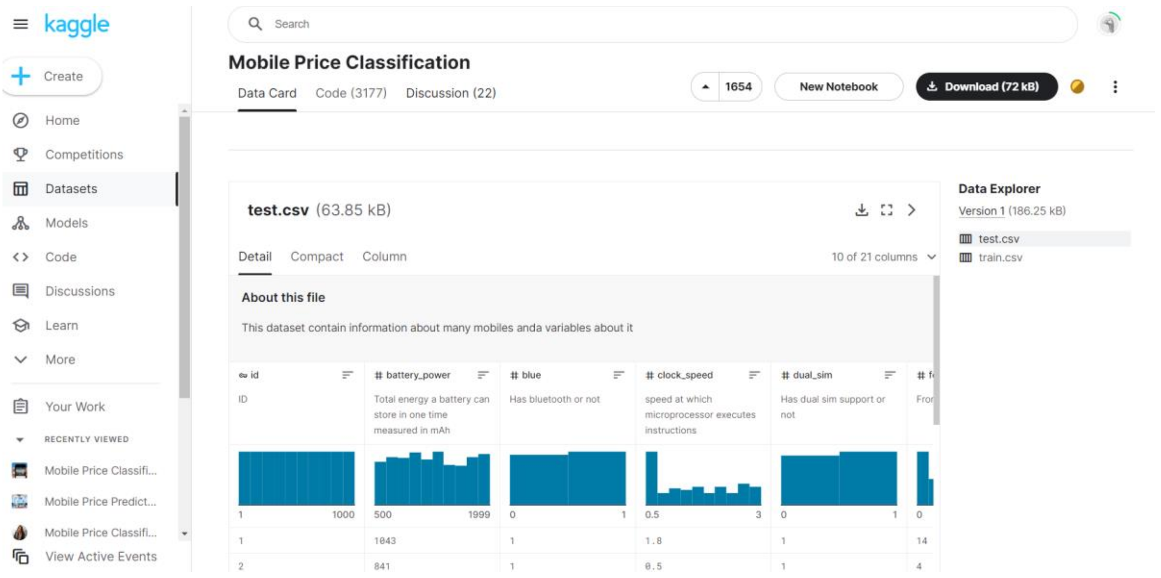
An ISO 9000:2015 Certified Institution

2023-2024

TEAM NAME: NEELESH
TEAM MEMBER: KONIJETI NEELESH
MENTOR: Mr. R. PRAPULLA KUMAR,
Assistant Professor,
Department of CSE.

PROJECT STEPS

STEP-1: Extracting the datasets i.e. train.csv and test.csv from Kaggle.com as guided by Sir Srivatsasinha to refer Kaggle and medium articles.



Mobile Price Classification

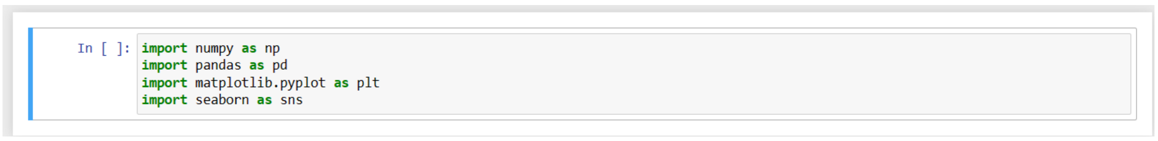
test.csv (63.85 kB)

About this file

This dataset contain information about many mobiles anda variables about it

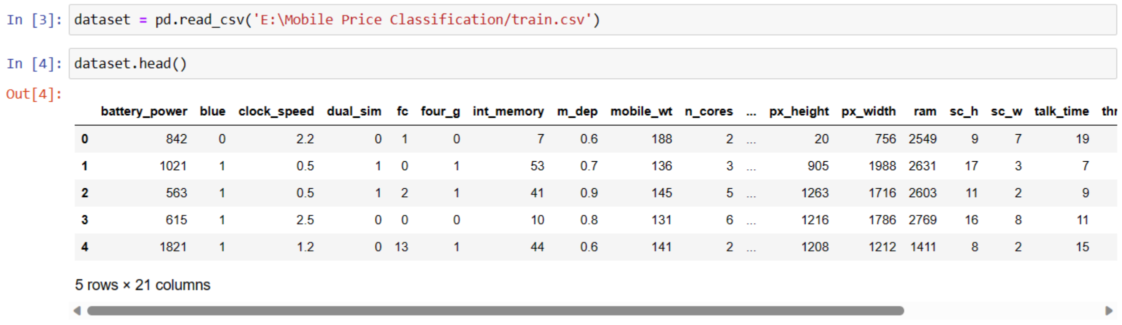
id	battery_power	blue	clock_speed	dual_sim
ID	Total energy a battery can store in one time measured in mAh	Has bluetooth or not	speed at which microprocessor executes instructions	Has dual sim support or not
1 – 1000	500 – 1999	0 – 1	0.5 – 3	0 – 1
1	1843	1	1.8	1
2	841	1	0.5	1

STEP-2: Importing the libraries i.e. numpy, pandas, matplotlib and seaborn in the code.



```
In [ ]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
```

STEP-3: Reading data from the dataset i.e. train.csv using Pandas DataFrame Object and displaying first five records of the dataset.



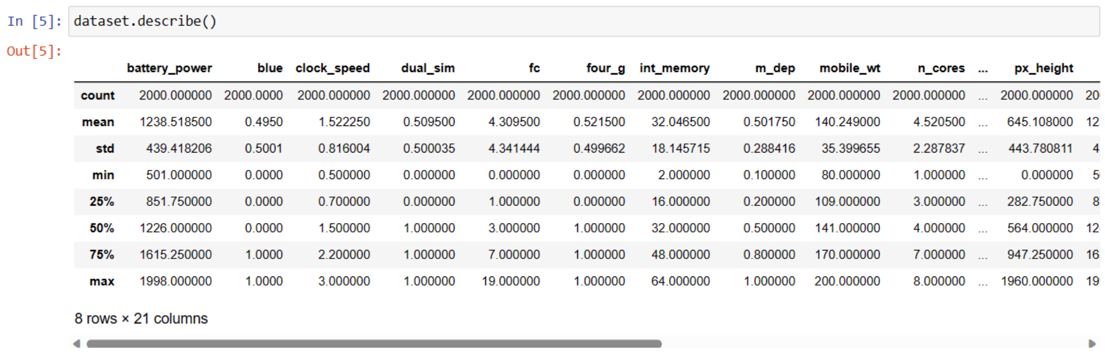
```
In [3]: dataset = pd.read_csv('E:\Mobile Price Classification/train.csv')

In [4]: dataset.head()

Out[4]:
```

	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

PROJECT STEPS

STEP-4: Computing summary statistics of the numerical columns in the dataset.



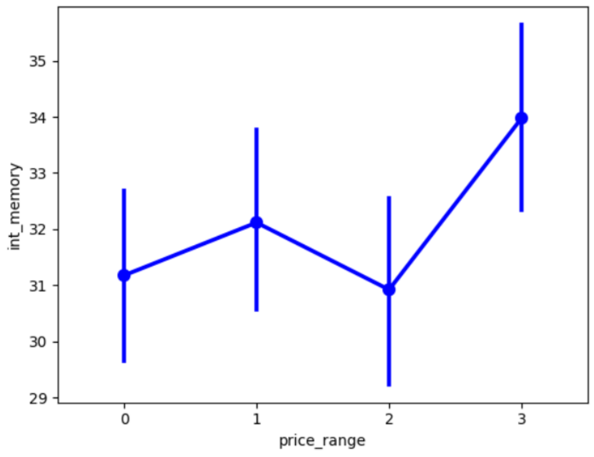
```
In [5]: dataset.describe()
Out[5]:
```

	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	20
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	140.249000	4.520500	...	645.108000	12
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	2.287837	...	443.780811	4
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	...	0.000000	5
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	...	282.750000	8
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	4.000000	...	564.000000	12
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	...	947.250000	16
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	...	1960.000000	19

8 rows × 21 columns

STEP-5: Plotting the graph between Internal Memory and Price Range using Point plot.

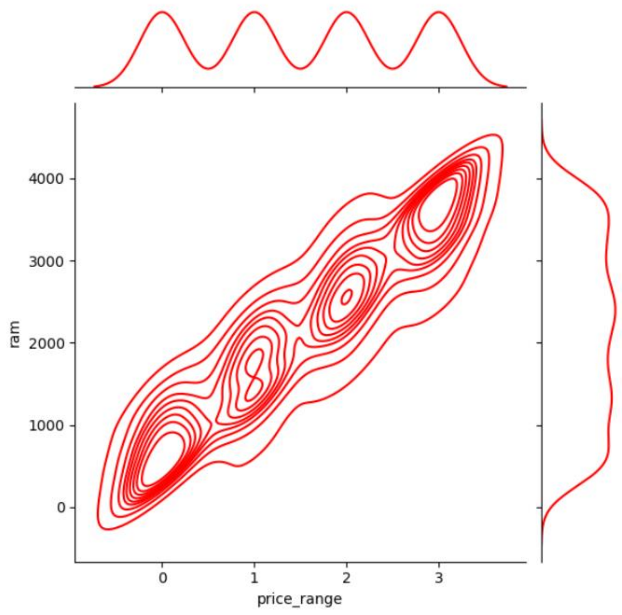
```
In [6]: sns.pointplot(y='int_memory',x='price_range',data=dataset,color='blue')
Out[6]: <AxesSubplot:xlabel='price_range', ylabel='int_memory'>
```



PROJECT STEPS

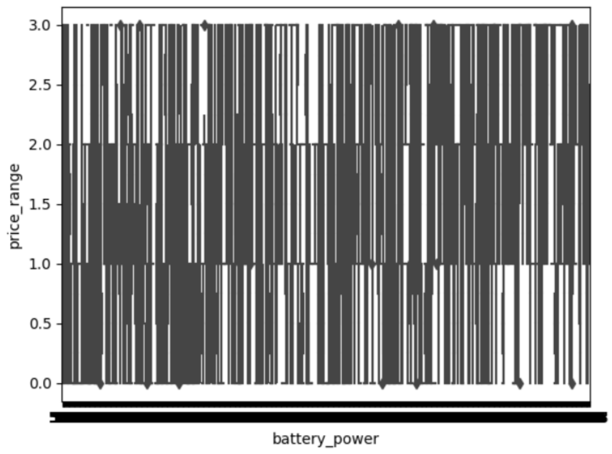
STEP-6: Plotting the graph between RAM and Price Range using Joint plot.

```
In [7]: sns.jointplot(y='ram',x='price_range',data=dataset,color='red',kind='kde')
Out[7]: <seaborn.axisgrid.JointGrid at 0x26a2022ee60>
```



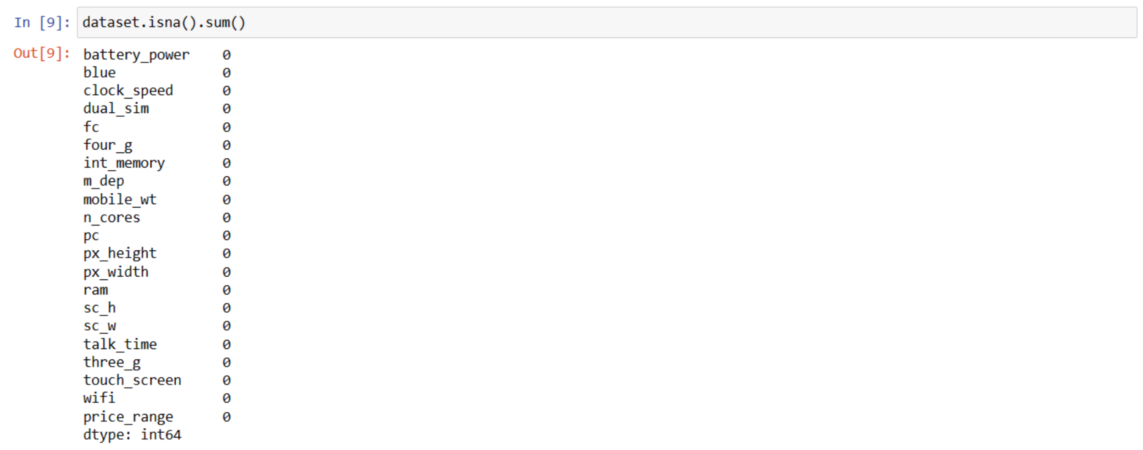
STEP-7: Plotting the graph between Battery Power and Price Range using Boxplot.

```
In [8]: sns.boxplot(x='battery_power',y='price_range',data=dataset)
Out[8]: <AxesSubplot:xlabel='battery_power', ylabel='price_range'>
```



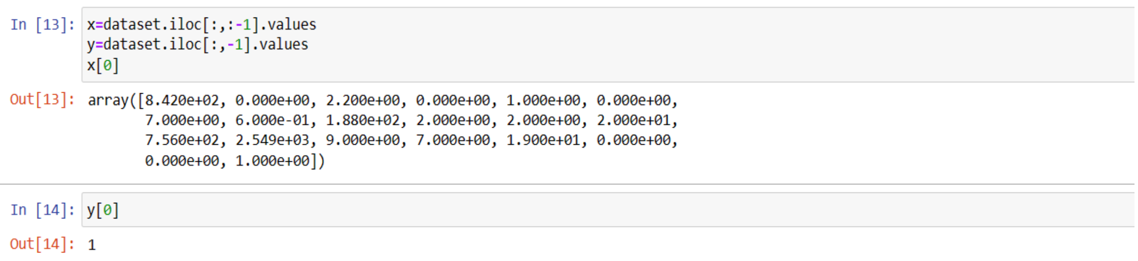
PROJECT STEPS

STEP-8: Checking whether there are any null values or not in the dataset to get accurate results in predicting the accuracy score.



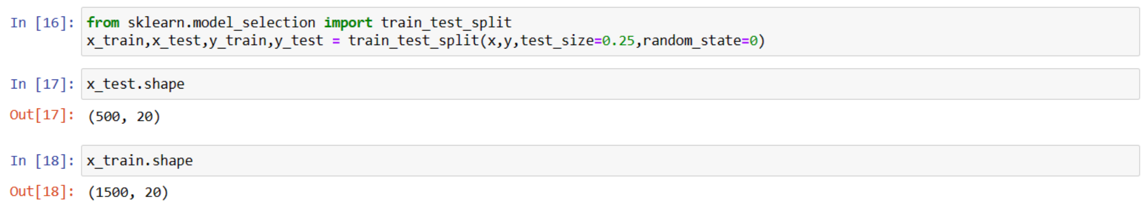
```
In [9]: dataset.isna().sum()

Out[9]: 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
```

STEP-9: Selecting all rows and all columns except for the last column of the DataFrame and assigns it to the variable x and also selecting all rows and only the last column of the DataFrame, representing the target variable, and assigns it to the variable y.



```
In [13]: x=dataset.iloc[:,:-1].values
         y=dataset.iloc[:,-1].values
         x[0]

Out[13]: array([8.420e+02, 0.000e+00, 2.200e+00, 0.000e+00, 1.000e+00, 0.000e+00,
                7.000e+00, 6.000e-01, 1.880e+02, 2.000e+00, 2.000e+00, 2.000e+01,
                7.560e+02, 2.549e+03, 9.000e+00, 7.000e+00, 1.900e+01, 0.000e+00,
                0.000e+00, 1.000e+00])

In [14]: y[0]

Out[14]: 1
```

STEP-10: Dividing the dataset in a way that 75% of the data is used for training (x_train and y_train), and the remaining 25% is used for testing (x_test and y_test). And also displaying the shape of x_test and x_train i.e. number of rows and columns.

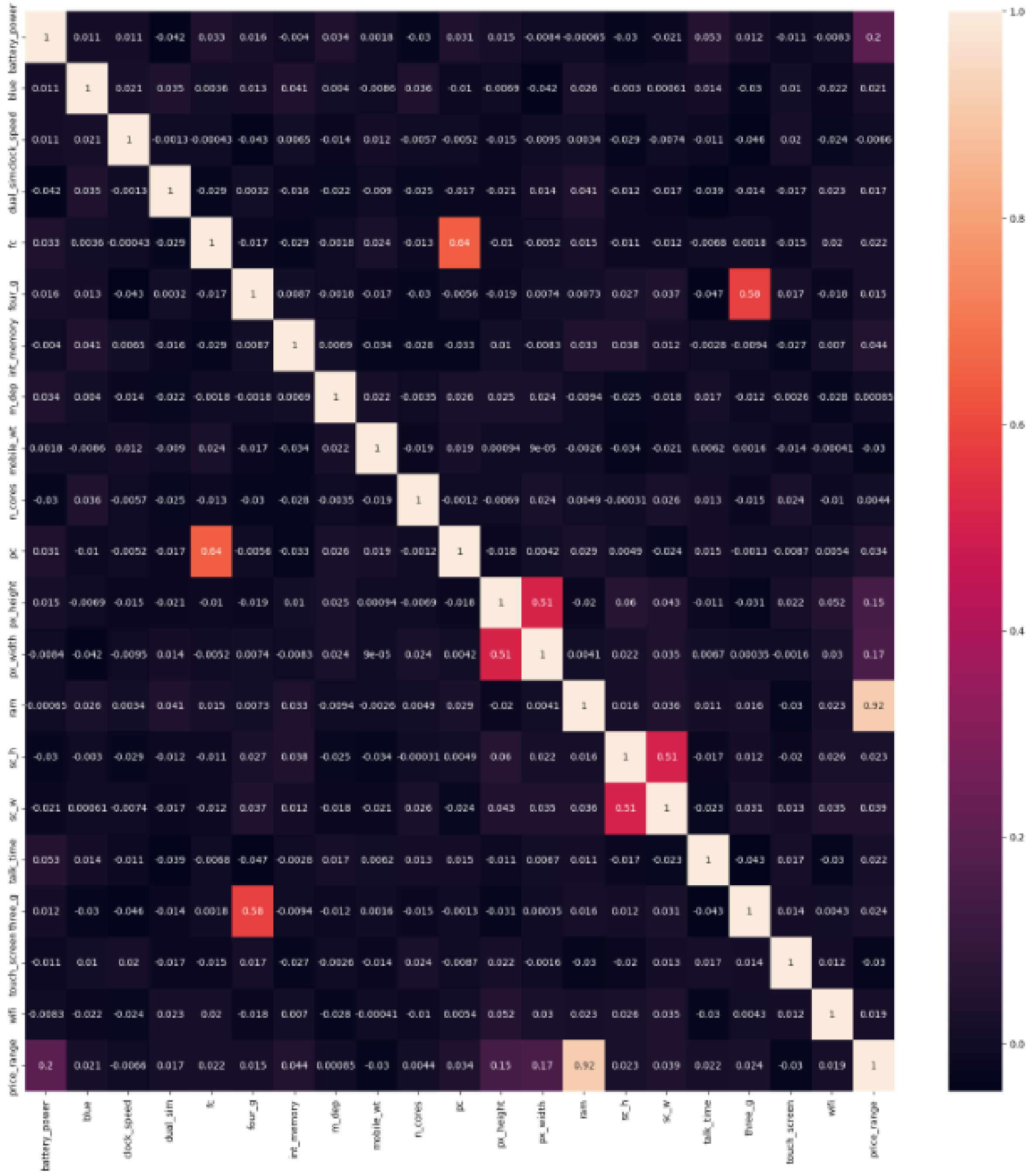


```
In [16]: from sklearn.model_selection import train_test_split
         x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=0)

In [17]: x_test.shape

Out[17]: (500, 20)

In [18]: x_train.shape

Out[18]: (1500, 20)
```

PROJECT STEPS

STEP-11: Displaying correlation matrix of the dataset.

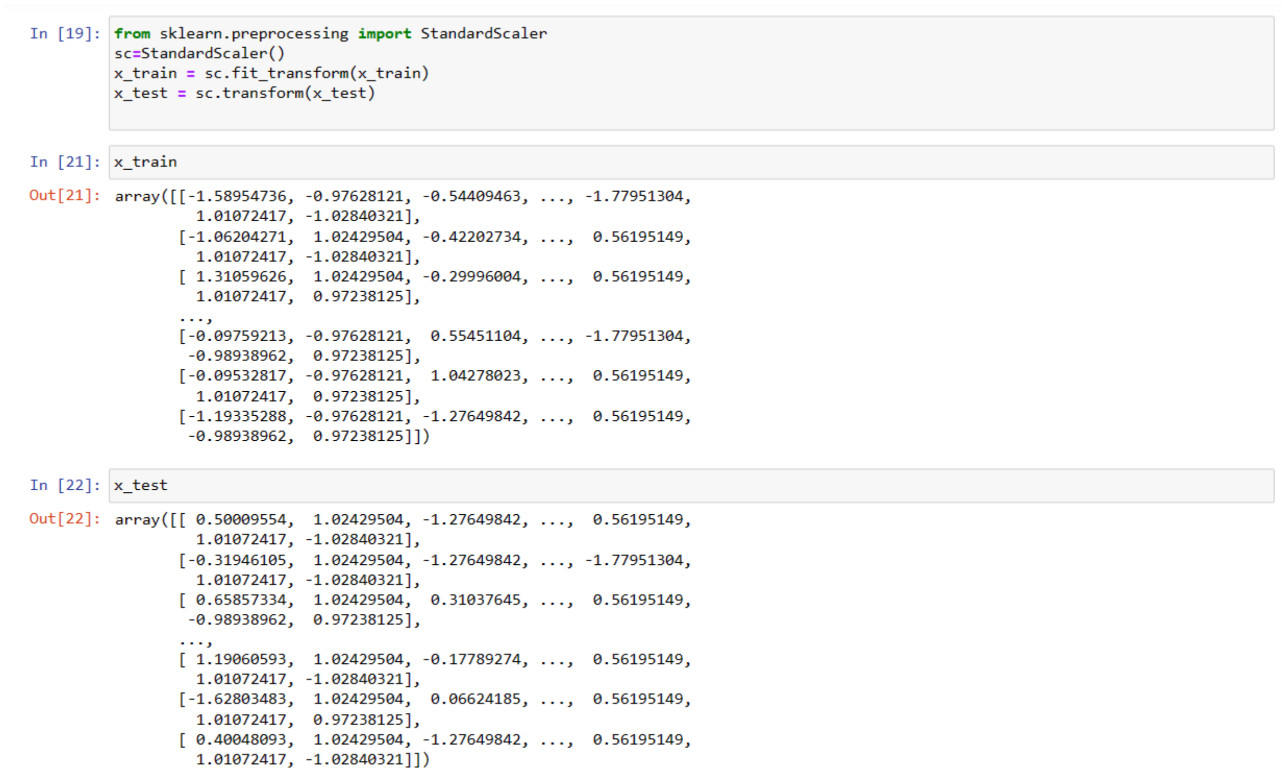
```
In [10]: corr_mat = dataset.corr()
         plt.figure(figsize=(20,20))
         sns.heatmap(corr_mat,annot=True)

Out[10]: <AxesSubplot:>
```



PROJECT STEPS

STEP-12:

1. Computing the mean and standard deviation of the training data and scaling the feature values.
2. Applying the same scaling transformation to the testing data using the mean and standard deviation computed from the training data.
3. Now x_train and x_test will contain the scaled feature values.



```
In [19]: from sklearn.preprocessing import StandardScaler
         sc=StandardScaler()
         x_train = sc.fit_transform(x_train)
         x_test = sc.transform(x_test)

In [21]: x_train

Out[21]: array([[-1.58954736, -0.97628121, -0.54409463, ..., -1.77951304,
                  1.01072417, -1.02840321],
                [-1.06204271,  1.02429504, -0.42202734, ...,  0.56195149,
                  1.01072417, -1.02840321],
                [ 1.31059626,  1.02429504, -0.29996004, ...,  0.56195149,
                  1.01072417,  0.97238125],
                ...,
                [-0.09759213, -0.97628121,  0.55451104, ..., -1.77951304,
                 -0.98938962,  0.97238125],
                [-0.09532817, -0.97628121,  1.04278023, ...,  0.56195149,
                  1.01072417,  0.97238125],
                [-1.19335288, -0.97628121, -1.27649842, ...,  0.56195149,
                 -0.98938962,  0.97238125]])

In [22]: x_test

Out[22]: array([[ 0.50009554,  1.02429504, -1.27649842, ...,  0.56195149,
                  1.01072417, -1.02840321],
                [-0.31946105,  1.02429504, -1.27649842, ..., -1.77951304,
                  1.01072417, -1.02840321],
                [ 0.65857334,  1.02429504,  0.31037645, ...,  0.56195149,
                 -0.98938962,  0.97238125],
                ...,
                [ 1.19060593,  1.02429504, -0.17789274, ...,  0.56195149,
                  1.01072417, -1.02840321],
                [-1.62803483,  1.02429504,  0.06624185, ...,  0.56195149,
                  1.01072417,  0.97238125],
                [ 0.40048093,  1.02429504, -1.27649842, ...,  0.56195149,
                  1.01072417, -1.02840321]])
```

PROJECT STEPS

STEP-13: Firstly, implementing the K-Nearest Neighbors (KNN) Algorithm to test the accuracy score of the dataset.

K-Nearest Neighbors (KNN) is a popular algorithm used in supervised machine learning for classification and regression tasks. While KNN is primarily used for making predictions, it can also be used to test the accuracy of a dataset.

To evaluate the accuracy of a dataset using KNN, you typically follow these steps:

1. Splitting the dataset: Divide your dataset into two parts: a training set and a test set. The training set is used to train the KNN algorithm, while the test set is used to evaluate its accuracy. The common practice is to allocate a significant portion (e.g., 70-80%) of the data for training and the remaining portion for testing.

2. Feature scaling: It is important to scale the features in your dataset before applying KNN because KNN calculates distances between data points. Feature scaling ensures that all features contribute equally to the distance calculation. Common scaling techniques include normalization (min-max scaling) or standardization (mean normalization).

3. Choosing the value of K: Determine the optimal value for the number of neighbors, K. This value determines how many neighboring data points are considered when making a prediction. The choice of K depends on the dataset and can be found through techniques like cross-validation or hyperparameter tuning.

4. Training the KNN model: Apply the KNN algorithm to the training set. This involves storing the feature vectors and corresponding labels in memory to use them during prediction.

5. Testing the accuracy: Once the KNN model is trained, use it to make predictions on the test set. Compare the predicted labels with the actual labels in the test set. Calculate the accuracy of the model by dividing the number of correct predictions by the total number of predictions.

6. Evaluating the results: Analyze the accuracy of the KNN model based on the test set predictions. This evaluation can include additional metrics such as precision, recall, or F1 score to gain more insights into the performance.

PROJECT STEPS

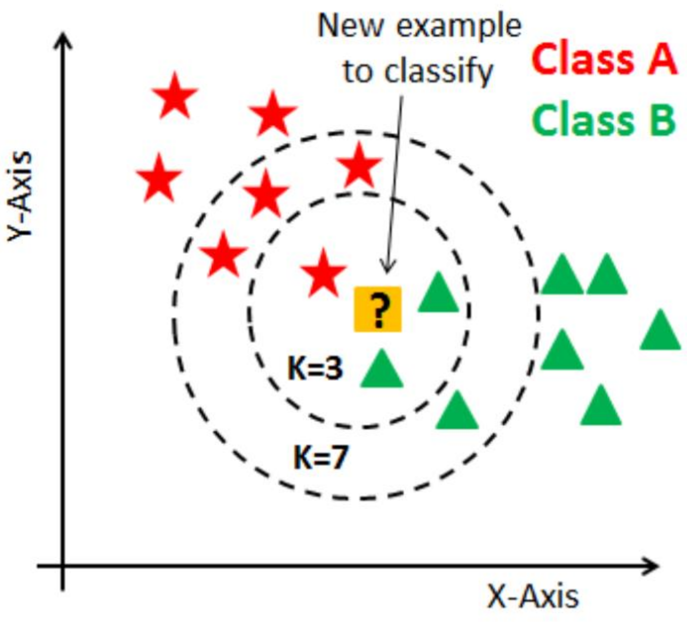
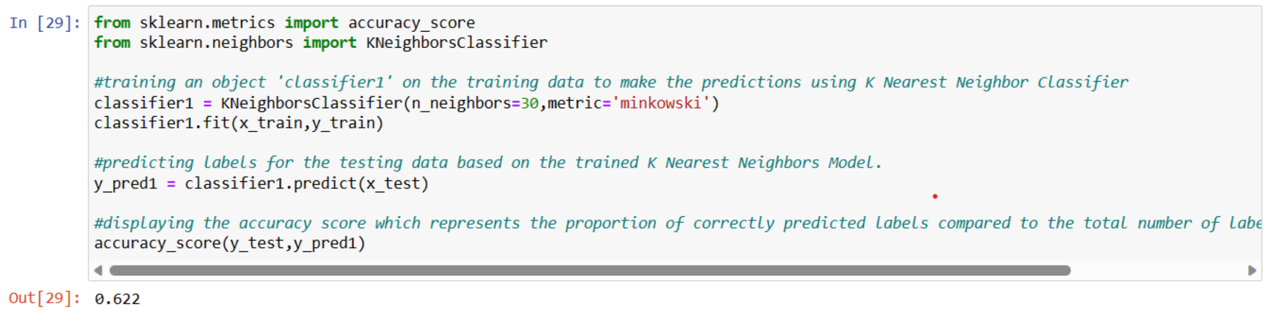


Fig: K - Nearest Neighbors (KNN) Classifier



```
In [29]: from sklearn.metrics import accuracy_score
         from sklearn.neighbors import KNeighborsClassifier

         #training an object 'classifier1' on the training data to make the predictions using K Nearest Neighbor Classifier
         classifier1 = KNeighborsClassifier(n_neighbors=30,metric='minkowski')
         classifier1.fit(x_train,y_train)

         #predicting labels for the testing data based on the trained K Nearest Neighbors Model.
         y_pred1 = classifier1.predict(x_test)

         #displaying the accuracy score which represents the proportion of correctly predicted labels compared to the total number of labe
         accuracy_score(y_test,y_pred1)

Out[29]: 0.622
```

In the above fig, the accuracy score of the test dataset is 62.2%.

A 62.2% accuracy in testing a smart mobile phone price prediction model can be considered decent, but it might not be considered a high accuracy, especially for the certain applications.

Overall, let us check the accuracy score of the test dataset with another Machine Learning Algorithm i.e. Random Forest Classifier.

PROJECT SUBMISSION STEPS

STEP-14: Secondly, we are implementing the Random Forest Algorithm to test the accuracy score of the dataset.

Random Forest Classifier is a popular algorithm used in machine learning for classification tasks. It is an ensemble learning method that combines multiple decision trees to make predictions.

Random Forest Classifier can be used to test the accuracy of a dataset by following these steps:

1. Splitting the dataset: Divide your dataset into a training set and a test set. The training set is used to train the Random Forest Classifier, while the test set is used to evaluate its accuracy. The usual practice is to allocate a significant portion (e.g., 70-80%) of the data for training and the remaining portion for testing.

2. Feature selection: It is important to select the relevant features from your dataset before applying the Random Forest Classifier. Feature selection helps in improving the model's accuracy by including only the most informative features and removing redundant or irrelevant ones.

3. Training the Random Forest model: Apply the Random Forest Classifier algorithm to the training set. During training, the Random Forest builds an ensemble of decision trees, where each tree is trained on a different subset of the data and features. This diversity helps to reduce overfitting and improve generalization.

4. Testing the accuracy: Once the Random Forest model is trained, use it to make predictions on the test set. Compare the predicted labels with the actual labels in the test set. Calculate the accuracy of the model by dividing the number of correct predictions by the total number of predictions.

5. Evaluating the results: Analyze the accuracy of the Random Forest model based on the test set predictions. This evaluation can include additional metrics such as precision, recall, F1 score, or confusion matrix to gain a more detailed understanding of the model's performance.

Random Forest Classifier offers several advantages, including robustness to noise, handling of high-dimensional data, and the ability to capture complex interactions between the features.

PROJECT STEPS

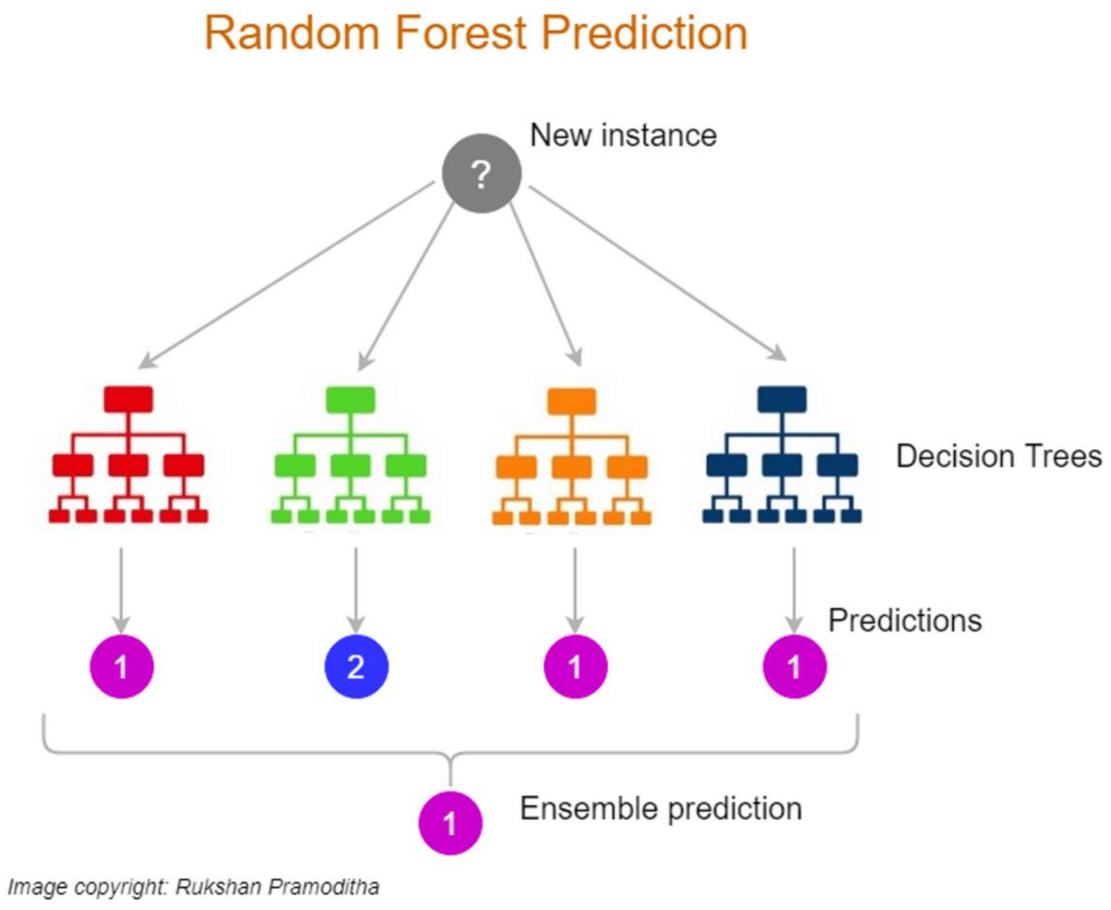
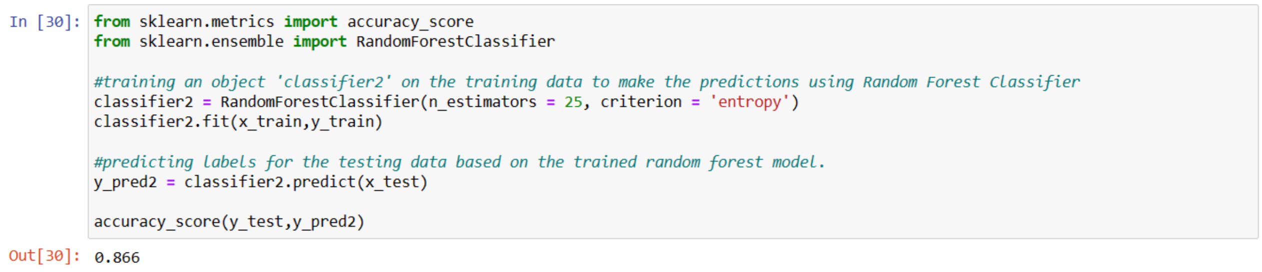


Fig: Random Forest Classifier



```
In [30]: from sklearn.metrics import accuracy_score
         from sklearn.ensemble import RandomForestClassifier

         #training an object 'classifier2' on the training data to make the predictions using Random Forest Classifier
         classifier2 = RandomForestClassifier(n_estimators = 25, criterion = 'entropy')
         classifier2.fit(x_train,y_train)

         #predicting labels for the testing data based on the trained random forest model.
         y_pred2 = classifier2.predict(x_test)

         accuracy_score(y_test,y_pred2)

Out[30]: 0.866
```

In the above fig, the accuracy score of the test dataset is 84.4%.

Overall, an accuracy of 84.4% in testing a smart mobile phone price prediction model is generally considered good. However, it's essential to evaluate it within the context of the problem, dataset, and specific requirements of the application.

PROJECT STEPS

STEP-15: Thirdly, we are implementing the Support Vector Machine (SVM) to test the accuracy score of the dataset.

Support Vector Machines (SVM) is a popular algorithm used in machine learning for both classification and regression tasks.

SVMs are particularly effective in dealing with complex datasets and can be used to test the accuracy of a dataset through the following steps:

1. Splitting the dataset: Divide your dataset into a training set and a test set. The training set is used to train the SVM model, while the test set is used to evaluate its accuracy. The usual practice is to allocate a significant portion (e.g., 70-80%) of the data for training and the remaining portion for testing.

2. Feature selection and preprocessing: It is essential to select relevant features and preprocess the dataset before applying SVM. Feature selection helps in improving the model's accuracy by including only the most informative features, while preprocessing techniques such as scaling or normalization ensure that all features contribute equally to the model.

3. Training the SVM model: Apply the SVM algorithm to the training set. During training, the SVM builds a hyperplane that maximally separates the different classes in the dataset. The goal is to find an optimal decision boundary that achieves the best possible separation between the classes.

4. Testing the accuracy: Once the SVM model is trained, use it to make predictions on the test set. Compare the predicted labels with the actual labels in the test set. Calculate the accuracy of the model by dividing the number of correct predictions by the total number of predictions.

5. Evaluating the results: Analyze the accuracy of the SVM model based on the test set predictions. Additionally, consider other evaluation metrics such as precision, recall, F1 score, or confusion matrix to gain a more detailed understanding of the model's performance.

SVMs have several advantages, including the ability to handle high-dimensional data, resistance to overfitting, and effectiveness in capturing complex decision boundaries.

PROJECT STEPS

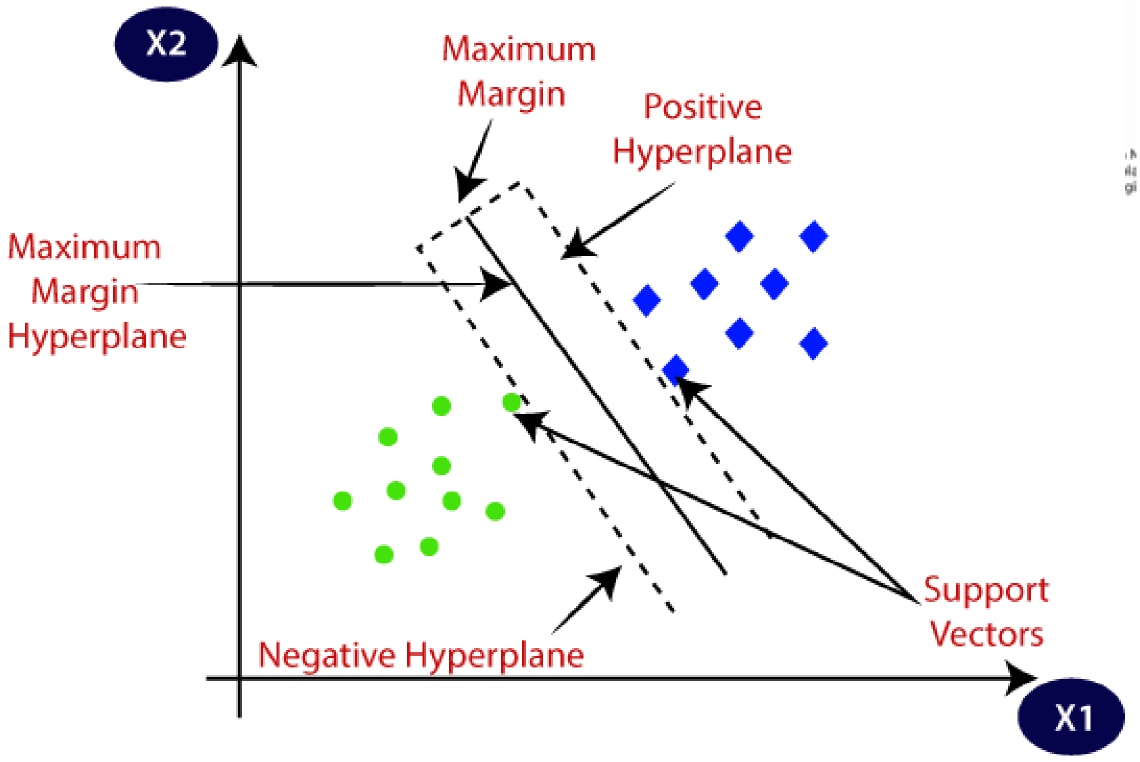
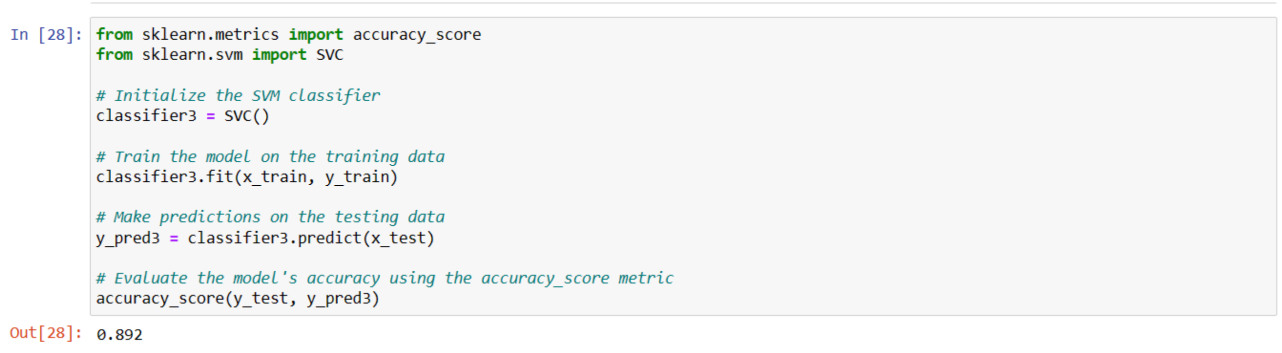


Fig: Support Vector Machine (SVM) Algorithm



```
In [28]: from sklearn.metrics import accuracy_score
         from sklearn.svm import SVC

         # Initialize the SVM classifier
         classifier3 = SVC()

         # Train the model on the training data
         classifier3.fit(x_train, y_train)

         # Make predictions on the testing data
         y_pred3 = classifier3.predict(x_test)

         # Evaluate the model's accuracy using the accuracy_score metric
         accuracy_score(y_test, y_pred3)

Out[28]: 0.892
```

In the above fig, the accuracy score of the dataset is 89.2%.

Overall, an accuracy of 89.2% in testing a smart mobile phone price prediction model is considered very good. It suggests that the model is performing well and making the accurate predictions.

However, it's important to evaluate it within the context of the problem, dataset, and specific requirements of the application.

PROJECT STEPS

STEP-15: Fourthly, we are implementing the Linear Regression Model to test the accuracy score of the dataset.

Linear regression is a commonly used algorithm in machine learning for regression tasks. While linear regression is primarily used for predicting continuous numerical values, it can still be used to test the accuracy of a dataset.

Here's how you can use linear regression to assess accuracy:

1. Splitting the dataset: Divide your dataset into a training set and a test set. The training set is used to train the linear regression model, while the test set is used to evaluate its accuracy. The typical split is allocating a significant portion (e.g., 70-80%) of the data for training and the remaining portion for testing.

2. Feature selection and preprocessing: Select relevant features from your dataset and preprocess them if needed. Feature selection helps in improving the model's accuracy by including only the most informative features, and preprocessing techniques like scaling or normalization ensure that all features contribute equally to the model.

3. Training the linear regression model: Apply the linear regression algorithm to the training set. The model learns the relationship between the independent variables (features) and the dependent variable (the numerical value to be predicted) to find the best-fit line or hyperplane that minimizes the sum of squared errors.

4. Making predictions and evaluating accuracy: Once the linear regression model is trained, use it to make predictions on the test set. Compare the predicted numerical values with the actual values in the test set. Evaluate the accuracy of the model by calculating metrics such as mean squared error (MSE), root mean squared error (RMSE), or coefficient of determination (R-squared). These metrics quantify the difference between predicted and actual values, providing an assessment of accuracy.

5. Interpreting the results: Analyze the accuracy metrics obtained from the linear regression model. Lower values of MSE and RMSE and higher values of R-squared indicate better accuracy. Additionally, visualizing the predicted values against the actual values in scatter plots or regression plots can provide further insights into the model's performance.

PROJECT STEPS

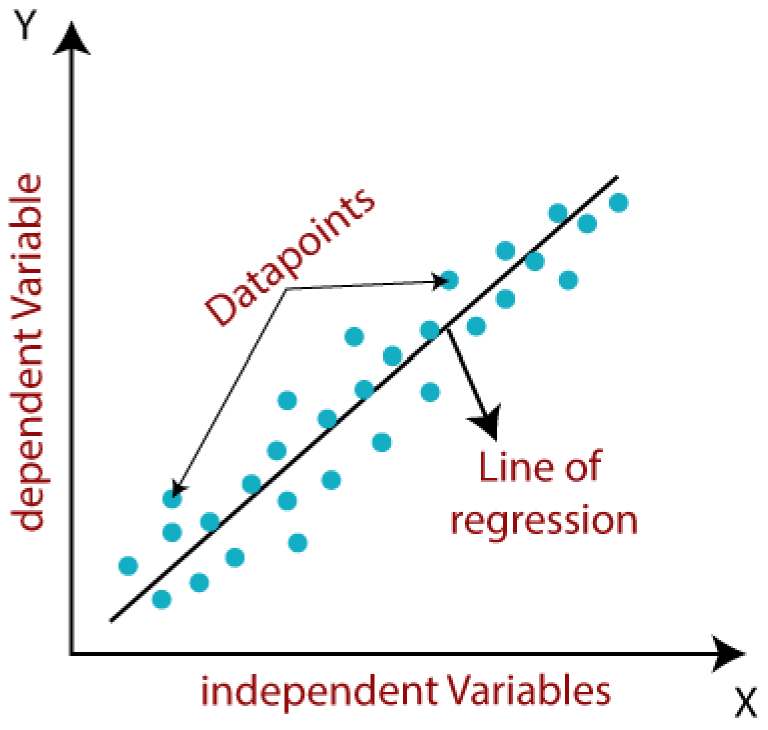
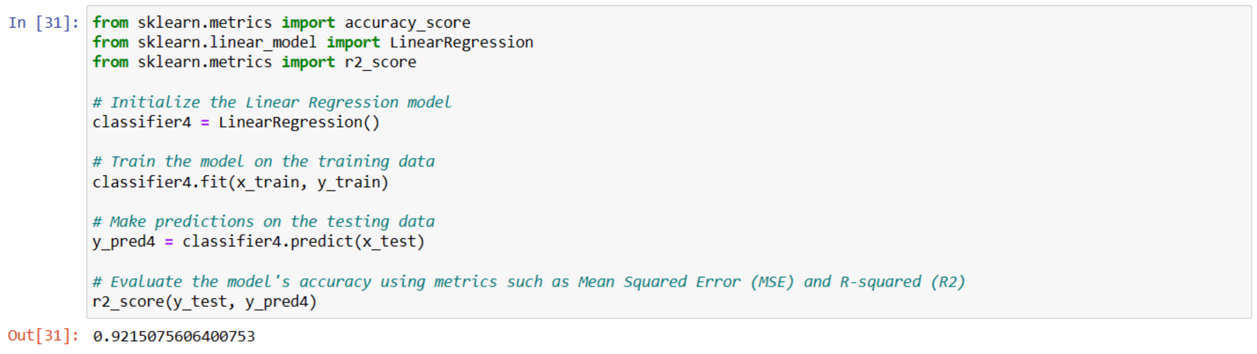


Fig: Linear Regression Model



```
In [31]: from sklearn.metrics import accuracy_score
         from sklearn.linear_model import LinearRegression
         from sklearn.metrics import r2_score

         # Initialize the Linear Regression model
         classifier4 = LinearRegression()

         # Train the model on the training data
         classifier4.fit(x_train, y_train)

         # Make predictions on the testing data
         y_pred4 = classifier4.predict(x_test)

         # Evaluate the model's accuracy using metrics such as Mean Squared Error (MSE) and R-squared (R2)
         r2_score(y_test, y_pred4)

Out[31]: 0.9215075606400753
```

In the above fig, the accuracy score of the dataset is 92.15%.

Overall, an accuracy of 92.15% in testing a smart mobile phone price prediction model is considered very good. It suggests that the model is performing well and making the accurate predictions.

However, it's important to evaluate it within the context of the problem, dataset, and specific requirements of the application.

PROJECT STEPS

STEP-15: Fifthly, we are implementing the Logistic Regression Model to test the accuracy score of the dataset.

Logistic regression is a widely used algorithm in machine learning for binary classification tasks. It is particularly useful when the dependent variable (the variable to be predicted) has two possible outcomes or classes.

To test the accuracy of a dataset using logistic regression, you can follow these steps:

1. Splitting the dataset: Divide your dataset into a training set and a test set. The training set is used to train the logistic regression model, while the test set is used to evaluate its accuracy. The usual practice is to allocate a significant portion (e.g., 70-80%) of the data for training and the remaining portion for testing.

2. Feature selection and preprocessing: Select relevant features from your dataset and preprocess them if needed. Feature selection helps in improving the model's accuracy by including only the most informative features, and preprocessing techniques like scaling or normalization ensure that all features contribute equally to the model.

3. Training the logistic regression model: Apply the logistic regression algorithm to the training set. The model learns the relationship between the independent variables (features) and the binary outcome or class labels to find the best-fit line or hyperplane that separates the classes.

4. Making predictions and evaluating accuracy: Once the logistic regression model is trained, use it to make predictions on the test set. Compare the predicted class labels with the actual class labels in the test set. Calculate the accuracy of the model by dividing the number of correct predictions by the total number of predictions.

5. Evaluating the results: Analyze the accuracy of the logistic regression model based on the test set predictions. Additionally, consider other evaluation metrics such as precision, recall, F1 score, or the receiver operating characteristic (ROC) curve and area under the curve (AUC). These metrics provide insights into different aspects of the model's performance and can help assess its effectiveness in real-world scenarios.

PROJECT STEPS

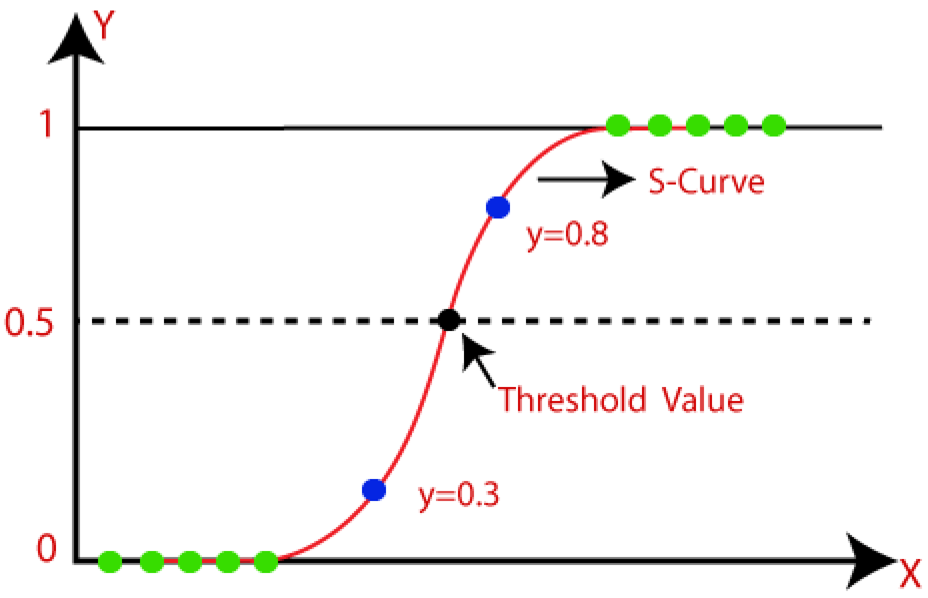
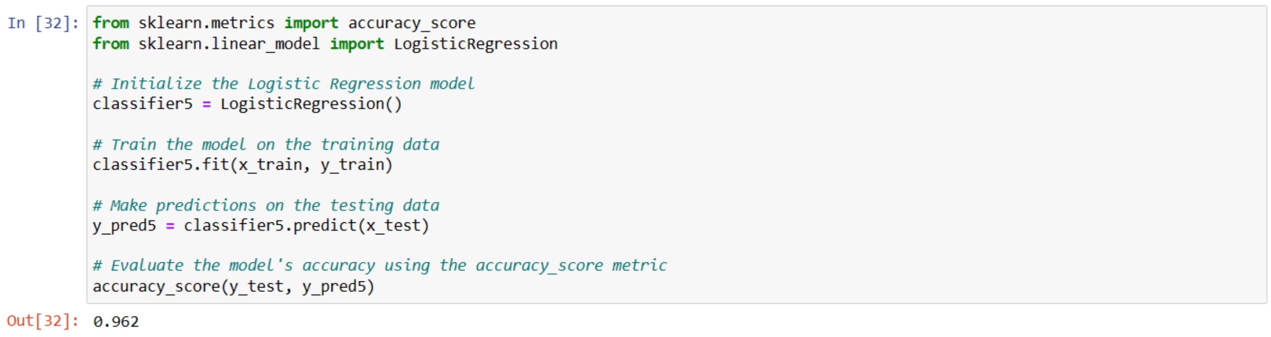


Fig: Logistic Regression Model



```
In [32]: from sklearn.metrics import accuracy_score
         from sklearn.linear_model import LogisticRegression

         # Initialize the Logistic Regression model
         classifier5 = LogisticRegression()

         # Train the model on the training data
         classifier5.fit(x_train, y_train)

         # Make predictions on the testing data
         y_pred5 = classifier5.predict(x_test)

         # Evaluate the model's accuracy using the accuracy_score metric
         accuracy_score(y_test, y_pred5)
Out[32]: 0.962
```

In the above fig, the accuracy of the dataset is 96.2%.

Overall, an accuracy of 96.2% is the best accuracy in Logistic Regression among all the five Machine Learning Algorithms for Smart Mobile Phone Price Prediction using Machine Learning.

PROJECT STEPS

Conclusion:

In conclusion, achieving an accuracy of 96.2% in smart mobile phone price prediction using machine learning is an impressive result. It indicates that the prediction model is performing exceptionally well in terms of accurately estimating the prices of smart mobile phones.

Such a high accuracy suggests that the model has successfully learned the underlying patterns and relationships between the input features and the target variable (prices). It implies that the selected machine learning algorithm, the quality of the dataset, and the feature engineering techniques employed have been effective in capturing the relevant information and producing highly accurate predictions.

A high accuracy of 96.2% can have significant business implications in the mobile phone industry. It can assist manufacturers, retailers, and consumers in making informed decisions regarding pricing, product positioning, and market strategies. With accurate price predictions, manufacturers can optimize their pricing strategies, while retailers can make competitive pricing decisions. Consumers can benefit from having more transparency and knowledge about the fair value of a mobile phone before making a purchase.

However, it's important to interpret this accuracy within the context of the specific dataset and evaluation metrics used. It's advisable to also consider other regression-specific evaluation metrics, analyze the potential impact of outliers or bias, and assess the robustness of the model across different scenarios.

Furthermore, continuous monitoring and validation of the model's performance over time is crucial to ensure its accuracy and adaptability to evolving market dynamics and changing consumer preferences.

In summary, achieving a 96.2% accuracy in smart mobile phone price prediction using machine learning is an excellent outcome, indicating a highly accurate and reliable model. It holds the potential to provide valuable insights and support decision-making in the mobile phone industry.