Google Play Store Application's Rating Prediction

Milestone: Report

Group 14

Name: Devkumar Patel
Name: Harshil Modi

857-930-8023 (Tel of Devkumar Patel)
857-544-9538 (Tel of Harshil Modi)

patel.devku@northeastern.edu
modi.harsh@northeastern.edu

Percentage of Effort Contributed by Student1: **50%**

Percentage of Effort Contributed by Student2: **50%**

Signature of Student 1: Devkumar Patel

Signature of Student 2: Harshil Modi

Submission Date: **04/23/2023**

Problem Setting

Google Playstore is a product distribution method where there are multiple applications in order to operate them in Android. It is a place where all the application's information is available like the number of installs worldwide, ratings, category, beta versions, updates available, last update details, and many others. Our project aims to find users' ratings based on these factors. This might help companies further invest in future applications like rolling out beta versions and subscriber versions to collect revenue. They can also gather users' feedback for a particular application and based on that they can decide whether that app is successful or not.
Managers of companies can also improve the versions of apps by knowing their customer's preferences and hence increase their revenue. They can also roll out advertisement-based revenue based on the number of installs. There can be numerous challenges like cleaning data due to the presence of null values, presence of a large dataset may increase the time of loading the results, especially during the model selection and implementation phase.

Problem Definition

While there are numerous challenges to address for this project, we have to find out a model which is perfectly suited to our dataset and which gives the highest accuracy in order to predict user ratings by going through multiple steps like data preprocessing, visualization, model exploration, model selection, model implementation, performance evaluation, and interpretation.

Data Sources:

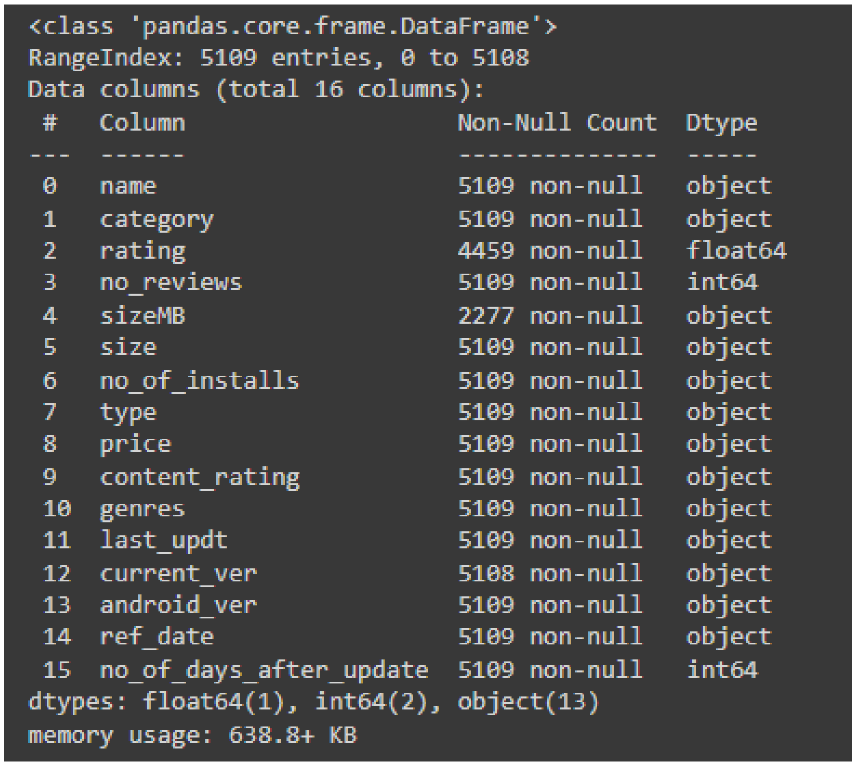
We have obtained the dataset from Kaggle.
https://www.kaggle.com/datasets/lava18/google-play-store-apps

Data Description:

This dataset contains 16 features including-
Name: Provides applications name from the playstore
Category: Defines an application's class like beauty, education, etc.

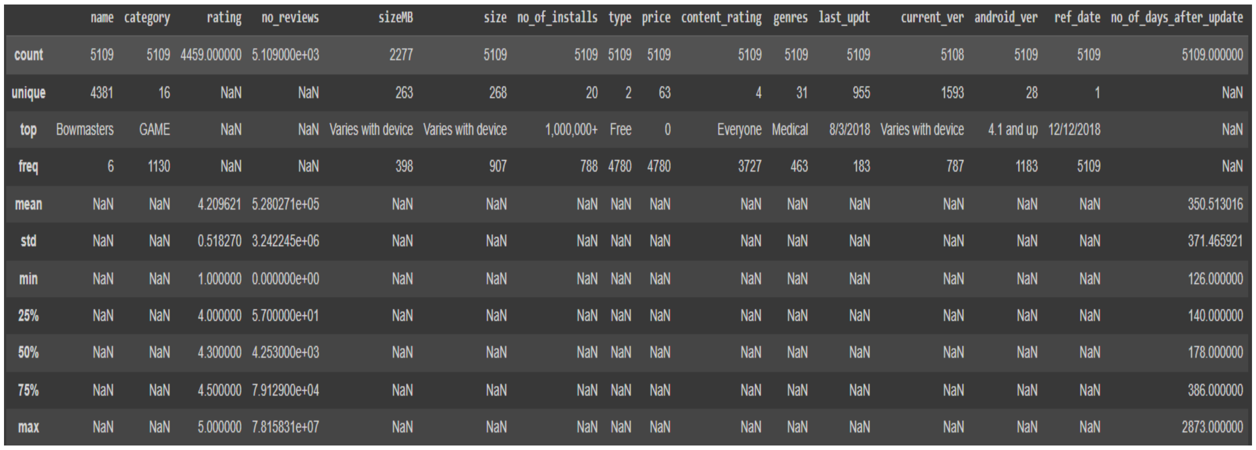
Rating: Showcases the ratings of that app on playstore
No_reviews: Indicates how many reviews are there for that app
sizeMB: Defines the size of the app in MB
size: Initiates size of the app and also shows whether it varies with the device
No_of_installs: Application's number of installs worldwide
type: Whether the application is free or paid
Price: The application's price in Dollar
content_rating: Specifies the age group for which the application is suited to
Genres: Determines classification of apps under various categories
last_updt: Provides information on the application's last update in mm/dd/yy format
current_ver: Application's current version after specific updates
android_ver: Determines the application's functioning on a particular Android version
ref_date: Provides the reference date
no_of_days_after_update: Information about days after which the app has been updated

The dataset has 5109 rows and covers 16 genres of applications.



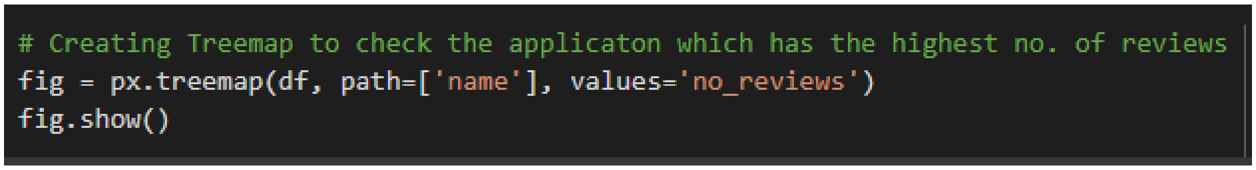
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5109 entries, 0 to 5108
Data columns (total 16 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   name                     5109 non-null   object
 1   category                 5109 non-null   object
 2   rating                   4459 non-null   float64
 3   no_reviews               5109 non-null   int64
 4   sizeMB                   2277 non-null   object
 5   size                     5109 non-null   object
 6   no_of_installs           5109 non-null   object
 7   type                     5109 non-null   object
 8   price                    5109 non-null   object
 9   content_rating           5109 non-null   object
 10  genres                   5109 non-null   object
 11  last_updt                5109 non-null   object
 12  current_ver              5108 non-null   object
 13  android_ver              5109 non-null   object
 14  ref_date                 5109 non-null   object
 15  no_of_days_after_update  5109 non-null   int64
dtypes: float64(1), int64(2), object(13)
memory usage: 638.8+ KB
```

In order to get more idea about the dataset, we have analyzed the distribution of some attributes.



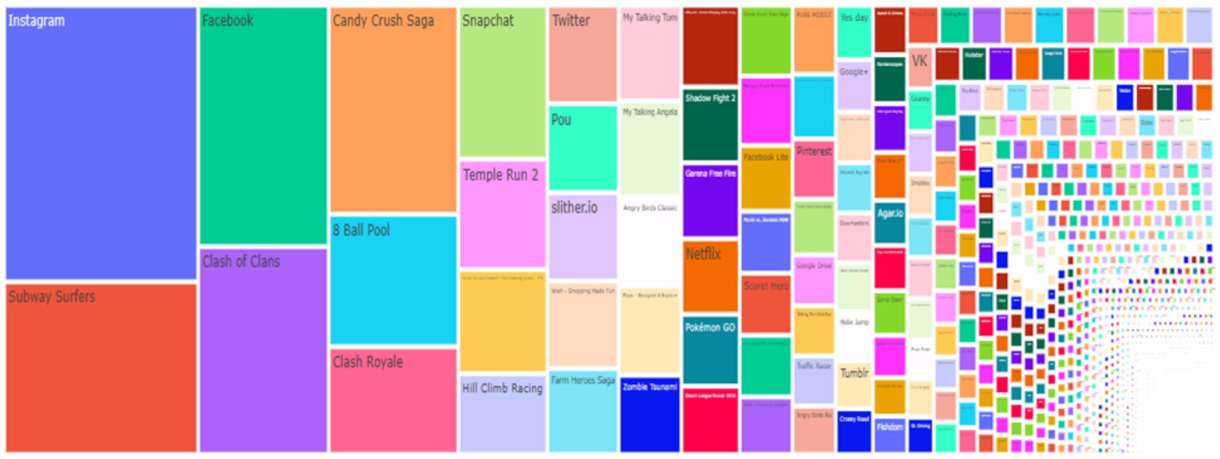
	name	category	rating	no_reviews	sizeMB	size	no_of_installs	type	price	content_rating	genres	last_updt	current_ver	android_ver	ref_date	no_of_days_after_update
count	5109	5109	4459.000000	5.109000e+03	2277	5109	5109	5109	5109	5109	5109	5109	5108	5109	5109	5109.000000
unique	4381	16	NaN	NaN	263	268	20	2	63	4	31	955	1593	28	1	NaN
top	Bowmasters	GAME	NaN	NaN	Varies with device	Varies with device	1,000,000+	Free	0	Everyone	Medical	8/3/2018	Varies with device	4.1 and up	12/12/2018	NaN
freq	6	1130	NaN	NaN	398	907	788	4780	4780	3727	463	183	787	1183	5109	NaN
mean	NaN	NaN	4.209621	5.280271e+05	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	350.513016
std	NaN	NaN	0.518270	3.242245e+06	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	371.465921
min	NaN	NaN	1.000000	0.000000e+00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	126.000000
25%	NaN	NaN	4.000000	5.700000e+01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	140.000000
50%	NaN	NaN	4.300000	4.253000e+03	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	178.000000
75%	NaN	NaN	4.500000	7.912900e+04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	386.000000
max	NaN	NaN	5.000000	7.815831e+07	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2873.000000

Data Exploration:



```python
# Creating Treemap to check the applicaton which has the highest no. of reviews
fig = px.treemap(df, path=['name'], values='no_reviews')
fig.show()
```

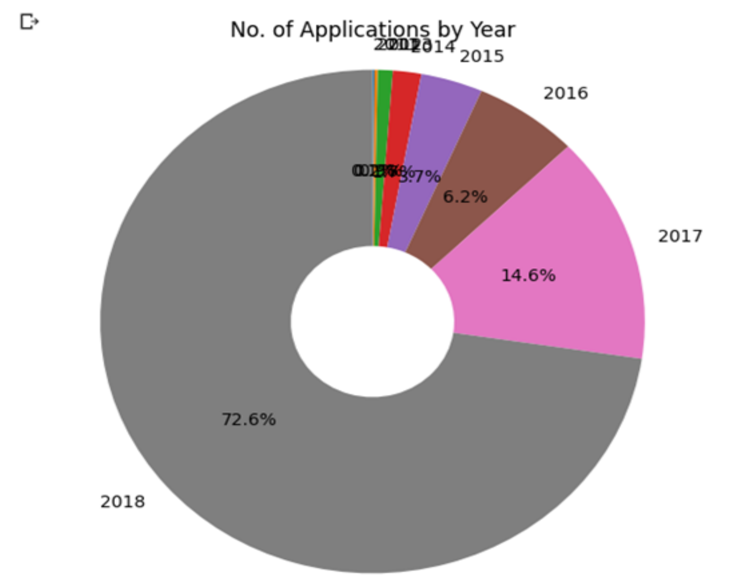
Through our visualization, we found an interesting observation about the gaming category which is leading the market.
From the following treemap, it is clearly seen that Instagram is the most viewed application by users around the globe. Following the list are Subway Surfers, Facebook, Clash of Clans, Candy Crush Saga, and 8 Ball Pool.



```
# abstracting the year from the last update column
df['last_updt'] = pd.to_datetime(df['last_updt'])
df['year_column'] = df['last_updt'].dt.year
```

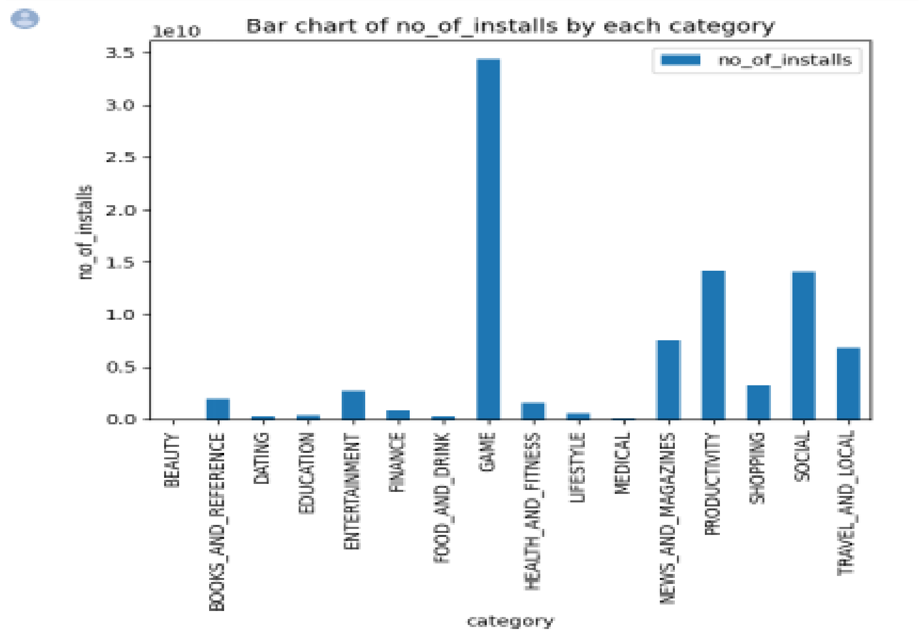
```
# Creating the donut chart which shows the number of applications by year
grouped = df.groupby('year_column')['name'].count()
fig, ax = plt.subplots(figsize=(6, 6))
ax.pie(grouped, labels=grouped.index, autopct='%1.1f%%', startangle=90,
        counterclock=False, wedgeprops={'width': 0.7})
ax.set_title('No. of Applications by Year')
ax.axis('equal')
plt.show()
```

From the donut chart, we can conclude that 73% of applications were developed in 2018. 4 times the combination of previous years. The reason could be due to less development in technology. Also, the number of applications is doubling each year.



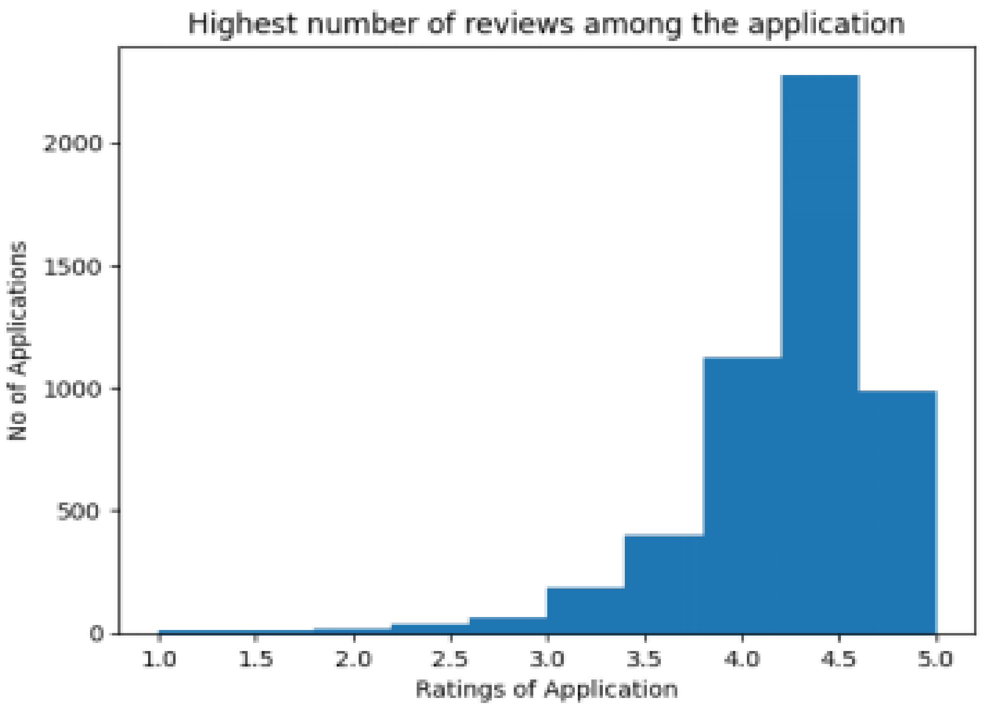
```
# Creating a bar using the group by function
df_groups = df.groupby(['category'])['no_of_installs'].sum()
df_groups.plot(kind = 'bar')
plt.title('Bar chart of no_of_installs by each category')
plt.xlabel('category')
plt.ylabel('no_of_installs')
plt.legend(loc='upper right')
plt.show()
```

The bar chart indicates that the Gaming category is the most booming industry with the highest number of installs. Productivity and social media applications have similar no. of installsSimilarly for news & magazines and travel & local applications. Medical, food & drink, and dating have the least preferences.



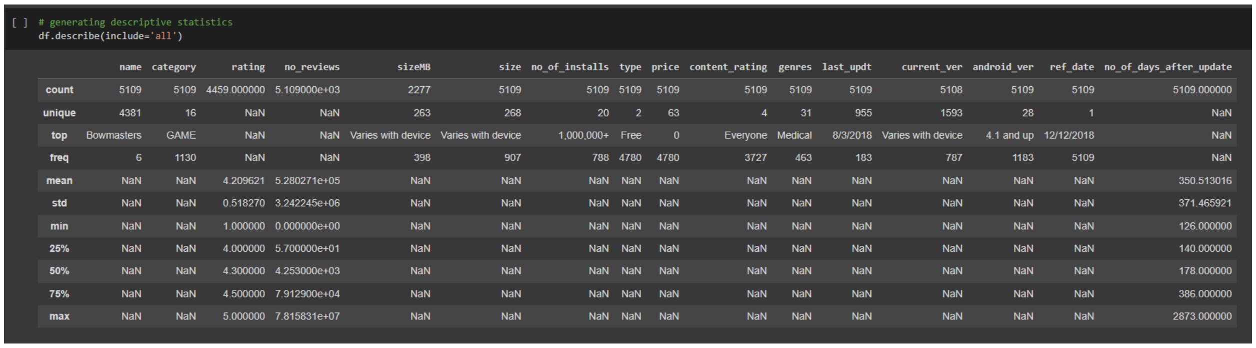
```
column_data = df['rating']

plt.hist(column_data)
plt.xlabel('Ratings of Application')
plt.ylabel('No of Applications')
plt.title('Highest number of reviews among the application')
plt.show()
```

The number of applications built in 2018 was 4 times the number of applications built in a combination of previous years. We could also figure out that people are enjoying the application and its feature as most of our applications within the dataset have been rated 4.5.



Data Mining Tasks:

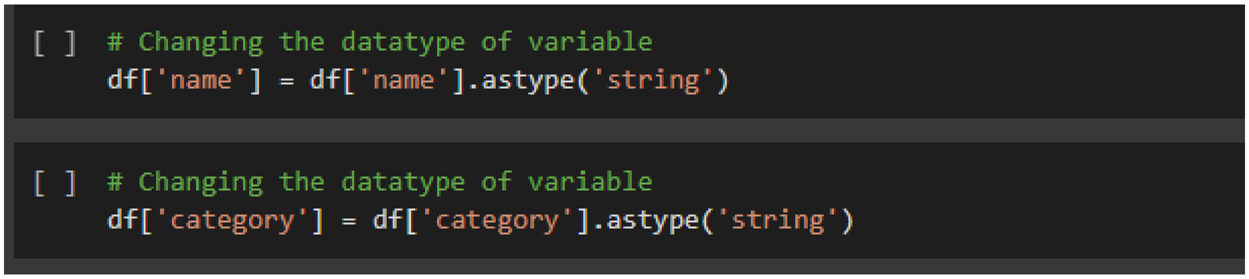
We have collected a raw dataset, which we needed to clean. The below figure shows a summary of the raw dataset used.



```
# generating descriptive statistics
df.describe(include='all')
```

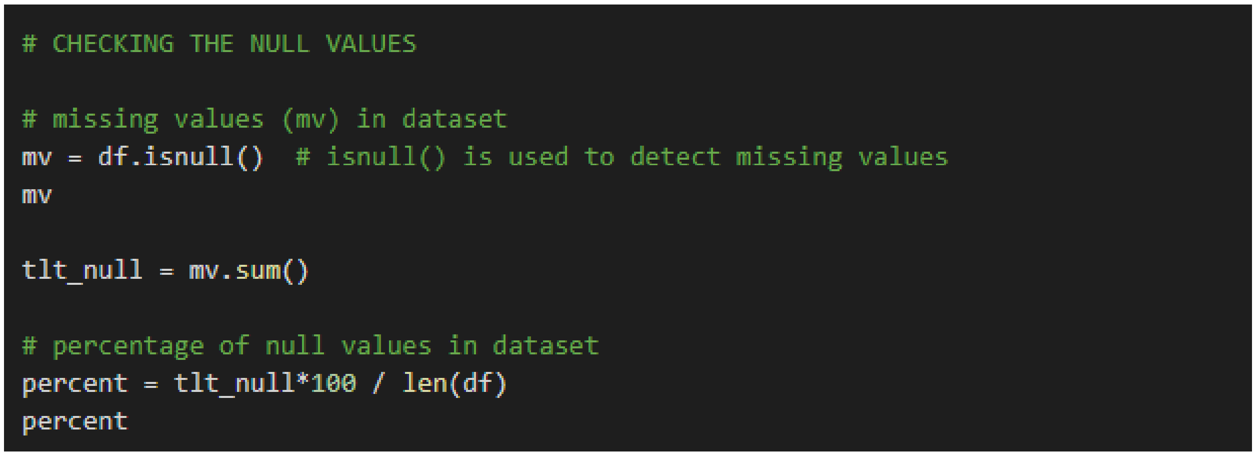
	name	category	rating	no_reviews	sizeMB	size	no_of_installs	type	price	content_rating	genres	last_updt	current_ver	android_ver	ref_date	no_of_days_after_update
count	5109	5109	4459.000000	5.109000e+03	2277	5109	5109	5109	5109	5109	5109	5109	5108	5109	5109	5109.000000
unique	4381	16	NaN	NaN	263	268	20	2	63	4	31	955	1593	28	1	NaN
top	Bowmasters	GAME	NaN	NaN	Varies with device	Varies with device	1,000,000+	Free	0	Everyone	Medical	8/3/2018	Varies with device	4.1 and up	12/12/2018	NaN
freq	6	1130	NaN	NaN	398	907	788	4780	4780	3727	463	183	787	1183	5109	NaN
mean	NaN	NaN	4.209621	5.280271e+05	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	350.513016
std	NaN	NaN	0.518270	3.242245e+06	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	371.465921
min	NaN	NaN	1.000000	0.000000e+00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	126.000000
25%	NaN	NaN	4.000000	5.700000e+01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	140.000000
50%	NaN	NaN	4.300000	4.253000e+03	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	178.000000
75%	NaN	NaN	4.500000	7.912900e+04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	386.000000
max	NaN	NaN	5.000000	7.815831e+07	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2873.000000

In order to the dataset, firstly, we tried changing the data type of the columns which didn’t contain any NULL values.



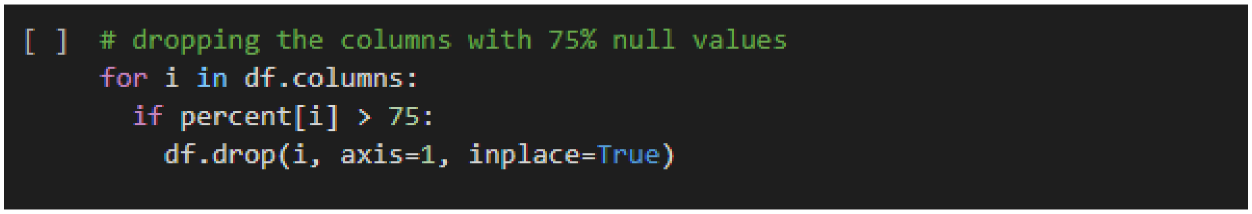
```
[ ]  # Changing the datatype of variable
     df['name'] = df['name'].astype('string')

[ ]  # Changing the datatype of variable
     df['category'] = df['category'].astype('string')
```

Calculating the NULL values within each column in the dataset using isnull().sum() function.

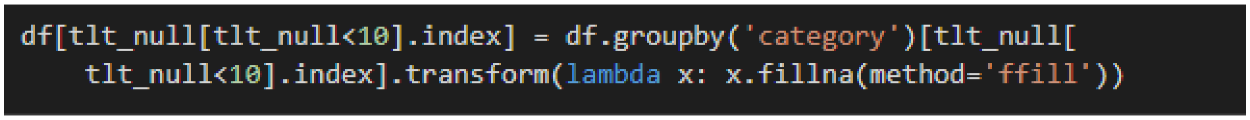


```
# CHECKING THE NULL VALUES

# missing values (mv) in dataset
mv = df.isnull()  # isnull() is used to detect missing values
mv

tlt_null = mv.sum()

# percentage of null values in dataset
percent = tlt_null*100 / len(df)
percent
```

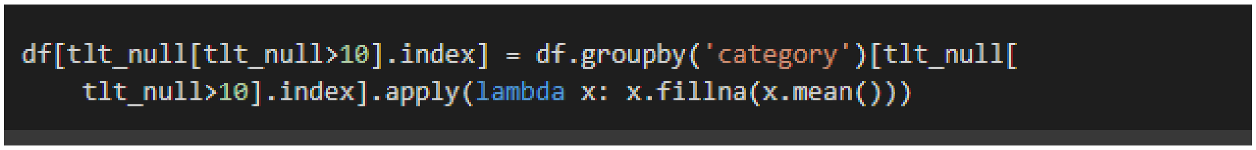
Columns with NULL values greater than 75% were dropped and the rest of the NULL values were replaced by the mean values.



```
[ ]  # dropping the columns with 75% null values
     for i in df.columns:
       if percent[i] > 75:
         df.drop(i, axis=1, inplace=True)
```



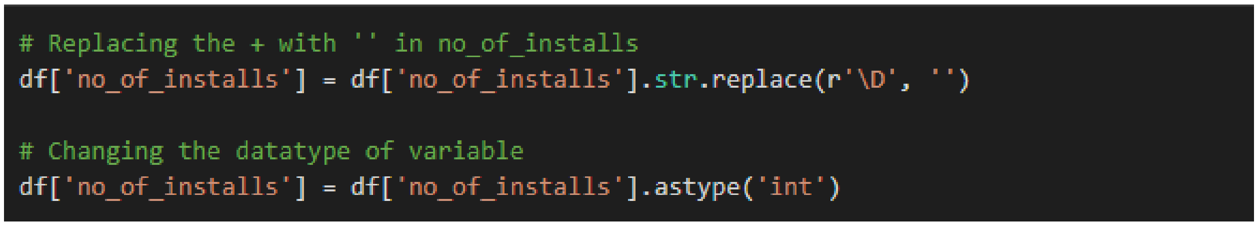
```
df[tlt_null[tlt_null<10].index] = df.groupby('category')[tlt_null[
    tlt_null<10].index].transform(lambda x: x.fillna(method='ffill'))
```



```
df[tlt_null[tlt_null>10].index] = df.groupby('category')[tlt_null[
    tlt_null>10].index].apply(lambda x: x.fillna(x.mean()))
```

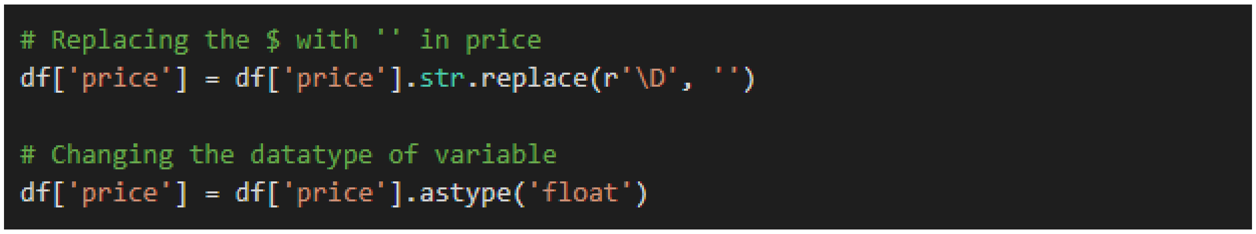
Firstly, we extracted the names of the columns with NULL values greater than 10. The next step was grouping it by category to calculate the mean value then we filled the NULL values with the mean.

There were instances where we had to replace some of the values within the dataset in order to maintain the format of the dataset. For example, the column 'no_of_installs' contained values like 5000+, 1000+, 250+, etc. Thus, we had to remove the '+' to make the column consistent with the integer variable.



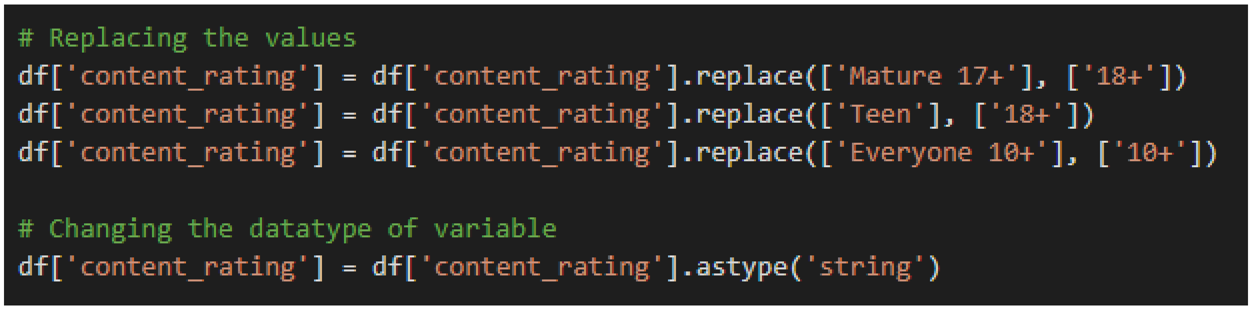
```
# Replacing the + with '' in no_of_installs
df['no_of_installs'] = df['no_of_installs'].str.replace(r'\D', '')

# Changing the datatype of variable
df['no_of_installs'] = df['no_of_installs'].astype('int')
```

Similarly, in the 'price' column we had to remove '$' to make use of the feature variable.



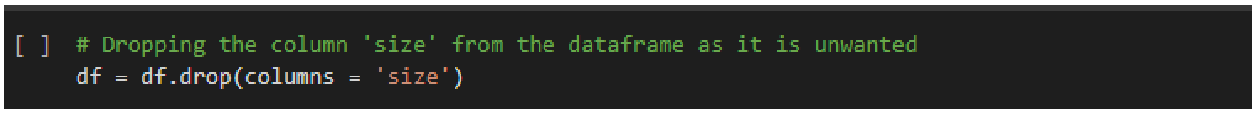
```
# Replacing the $ with '' in price
df['price'] = df['price'].str.replace(r'\D', '')

# Changing the datatype of variable
df['price'] = df['price'].astype('float')
```

Same with the 'content_rating' column. We had to replace some of the categories with the column.



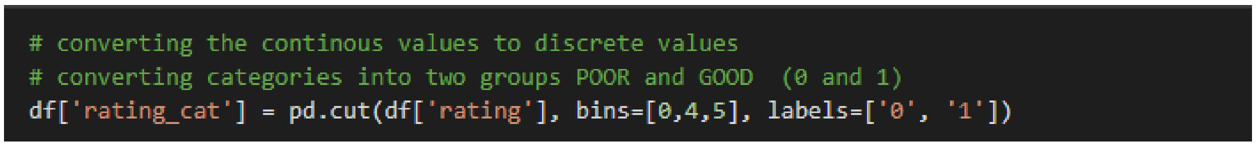
```
# Replacing the values
df['content_rating'] = df['content_rating'].replace(['Mature 17+'], ['18+'])
df['content_rating'] = df['content_rating'].replace(['Teen'], ['18+'])
df['content_rating'] = df['content_rating'].replace(['Everyone 10+'], ['10+'])

# Changing the datatype of variable
df['content_rating'] = df['content_rating'].astype('string')
```

We had to remove some of the unwanted columns within the dataset.



```
[ ]  # Dropping the column 'size' from the dataframe as it is unwanted
     df = df.drop(columns = 'size')
```

Finally, converted the outcome variable into categorical to make a classification model. We had categorized ratings into two (0: BAD and 1: GOOD).



```
# converting the continous values to discrete values
# converting categories into two groups POOR and GOOD  (0 and 1)
df['rating_cat'] = pd.cut(df['rating'], bins=[0,4,5], labels=['0', '1'])
```

Data Mining Models:

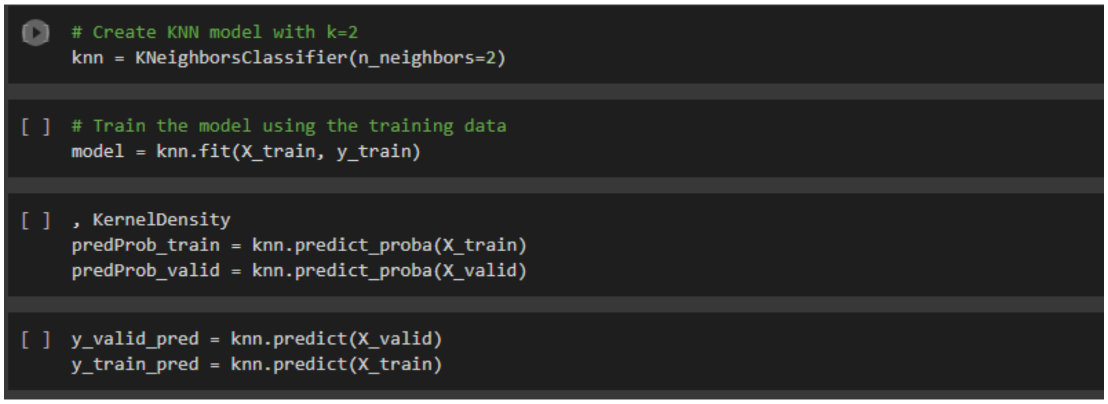
Initially, we tried the prediction model for the rating variable. But the algorithms we used didn't provide that accurate results.

Thus, we shifted to the classification model in order to get better accuracy of our model.

We had taken ‘no_reviews’, ‘sizeMB’, ‘no_of_installs’, ‘price’, and ‘no_of_days_after_update’ as feature variables and ‘rating_cat’ as the output variable. The dataset is divided into 60% training and 40% testing.

After the preprocessing step, we used the following algorithms on our clean dataset for the classification model -

1. K - Nearest Neighbour:
   K - Nearest Neighbour classifies the data point based on its neighbor.

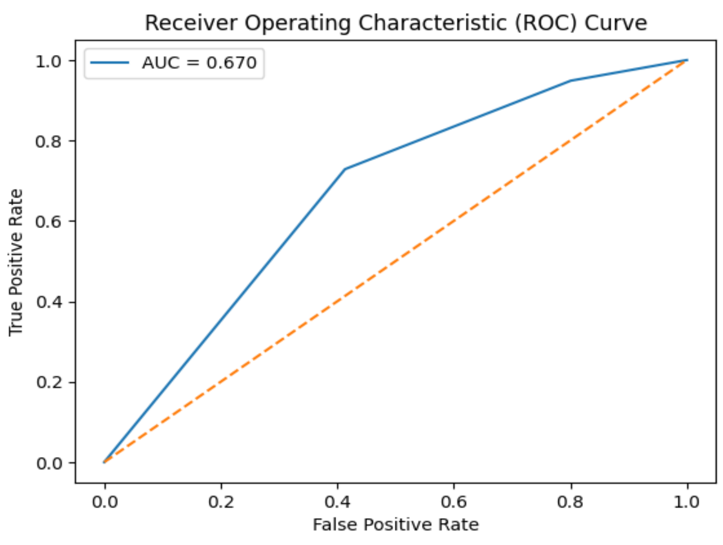


```
# Create KNN model with k=2
knn = KNeighborsClassifier(n_neighbors=2)
```

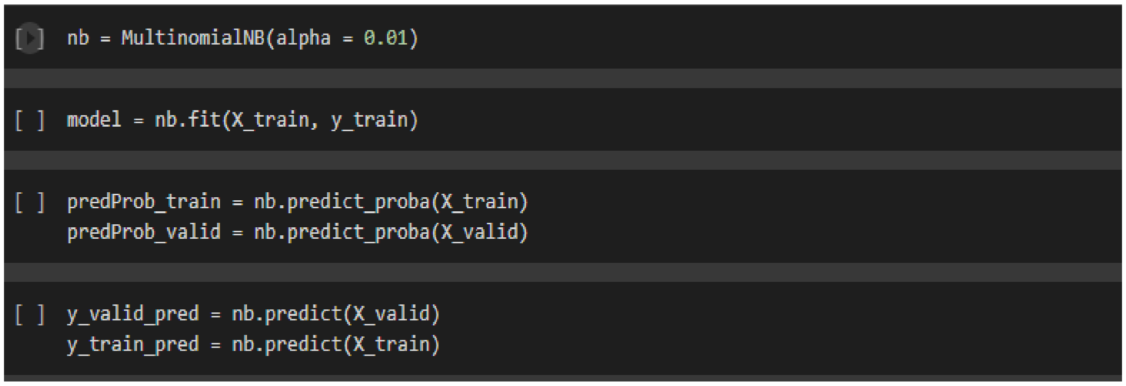
```
[ ] # Train the model using the training data
    model = knn.fit(X_train, y_train)
```

```
[ ] , KernelDensity
    predProb_train = knn.predict_proba(X_train)
    predProb_valid = knn.predict_proba(X_valid)
```

```
[ ] y_valid_pred = knn.predict(X_valid)
    y_train_pred = knn.predict(X_train)
```



2. Naive Bayes:
   Naive Bayes uses probability theory to predict the likelihood of a given input feature.

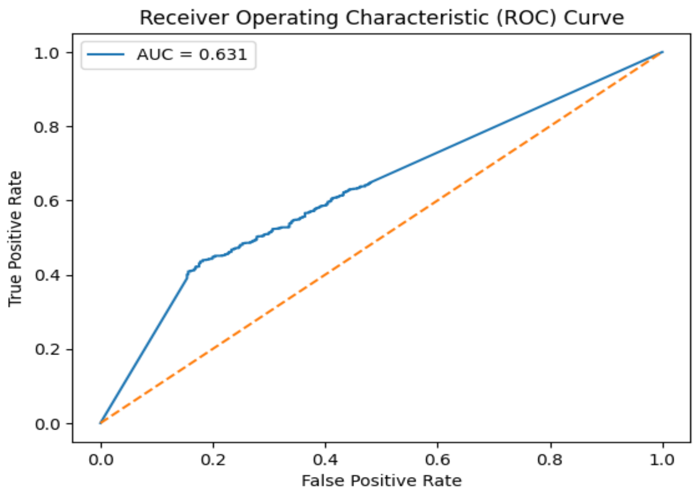


```
nb = MultinomialNB(alpha = 0.01)
```

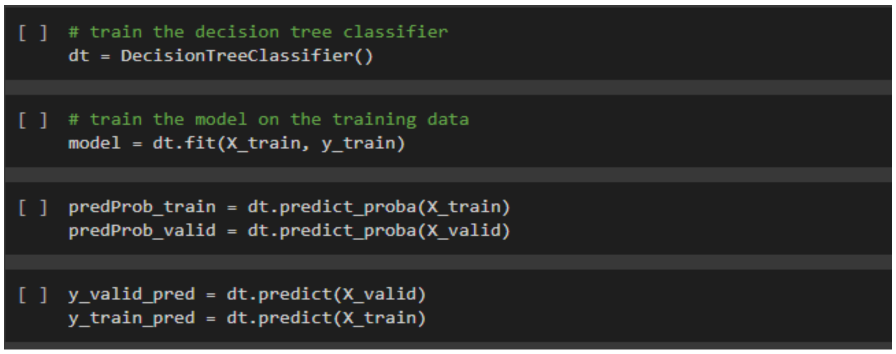
```
model = nb.fit(X_train, y_train)
```

```
predProb_train = nb.predict_proba(X_train)
predProb_valid = nb.predict_proba(X_valid)
```

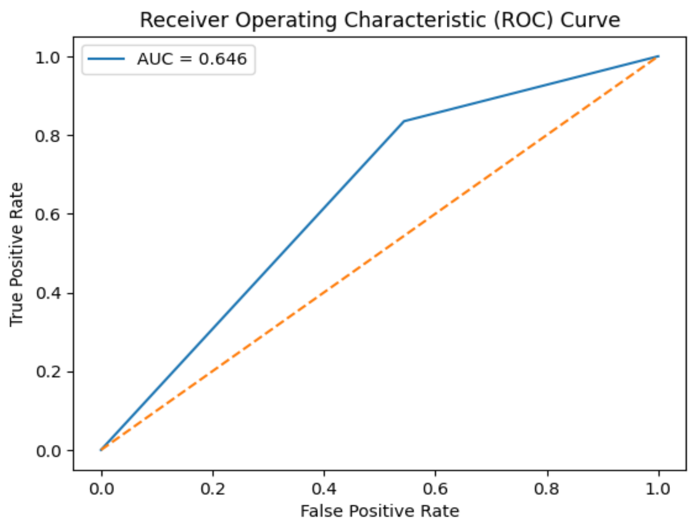
```
y_valid_pred = nb.predict(X_valid)
y_train_pred = nb.predict(X_train)
```



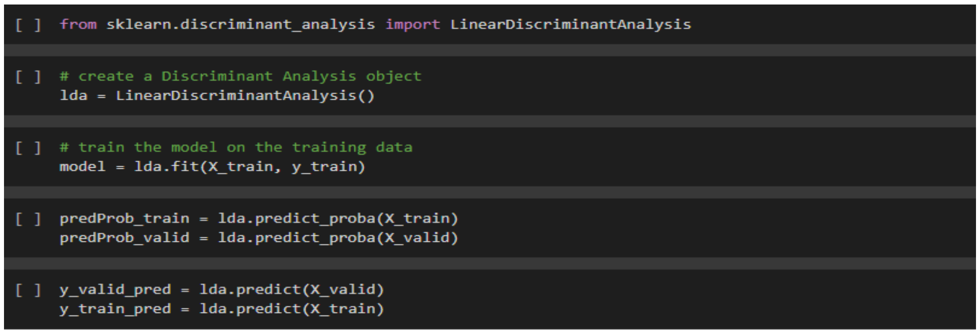
3. CART (Classification and Regression Tree):
   The classification tree splits the dataset into smaller subsets based on the values of the input features.



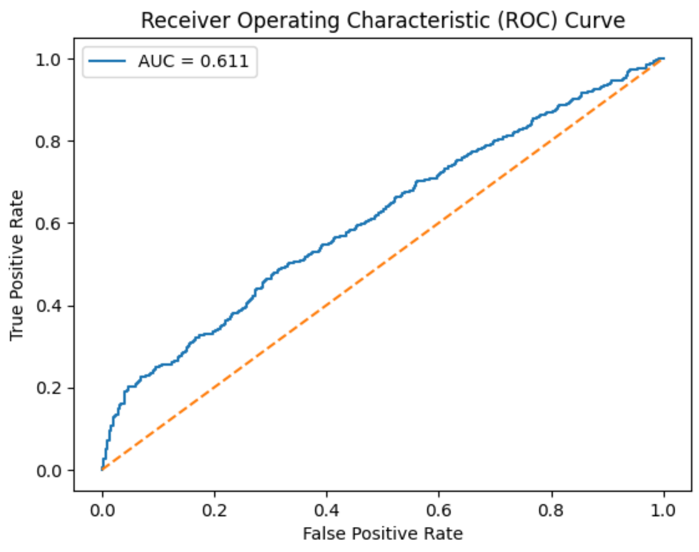
```
[ ] # train the decision tree classifier
    dt = DecisionTreeClassifier()

[ ] # train the model on the training data
    model = dt.fit(X_train, y_train)

[ ] predProb_train = dt.predict_proba(X_train)
    predProb_valid = dt.predict_proba(X_valid)

[ ] y_valid_pred = dt.predict(X_valid)
    y_train_pred = dt.predict(X_train)
```



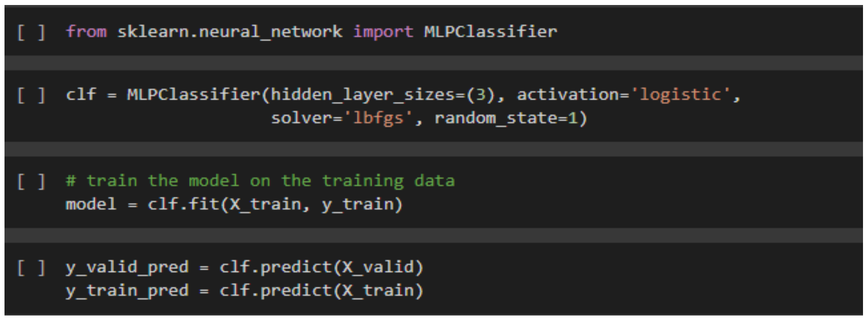
4. Discriminant Analysis:
   Discriminant Analysis finds the feature which can be best discriminated among the classes.



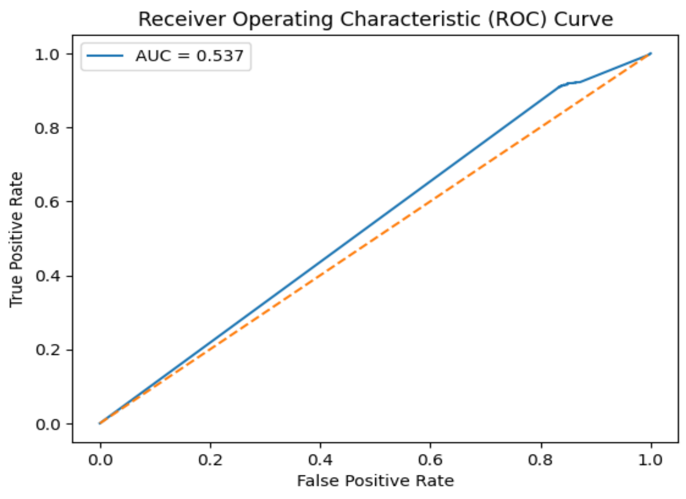
```
[ ] from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

[ ] # create a Discriminant Analysis object
    lda = LinearDiscriminantAnalysis()

[ ] # train the model on the training data
    model = lda.fit(X_train, y_train)

[ ] predProb_train = lda.predict_proba(X_train)
    predProb_valid = lda.predict_proba(X_valid)

[ ] y_valid_pred = lda.predict(X_valid)
    y_train_pred = lda.predict(X_train)
```



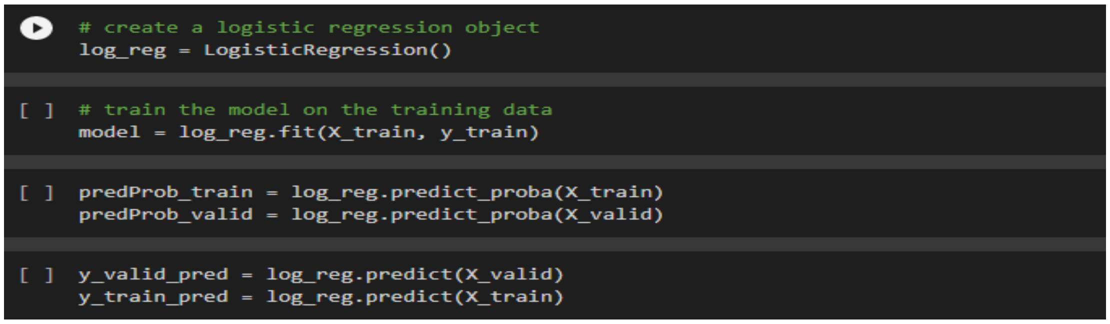
5. Neural Network:
   Neural Network learns from the data by adjusting the weights and biases of the neurons.



```
[ ] from sklearn.neural_network import MLPClassifier

[ ] clf = MLPClassifier(hidden_layer_sizes=(3), activation='logistic',
                        solver='lbfgs', random_state=1)

[ ] # train the model on the training data
    model = clf.fit(X_train, y_train)

[ ] y_valid_pred = clf.predict(X_valid)
    y_train_pred = clf.predict(X_train)
```



6. Logistic Regression:
   The algorithm works by estimating the coefficients of the input variables that maximize the likelihood of the observed data.

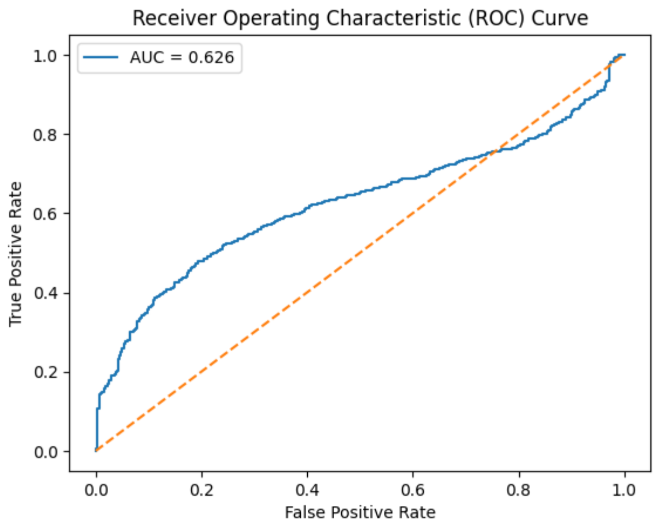


```
# create a logistic regression object
log_reg = LogisticRegression()
```

```
[ ] # train the model on the training data
model = log_reg.fit(X_train, y_train)
```

```
[ ] predProb_train = log_reg.predict_proba(X_train)
predProb_valid = log_reg.predict_proba(X_valid)
```

```
[ ] y_valid_pred = log_reg.predict(X_valid)
y_train_pred = log_reg.predict(X_train)
```

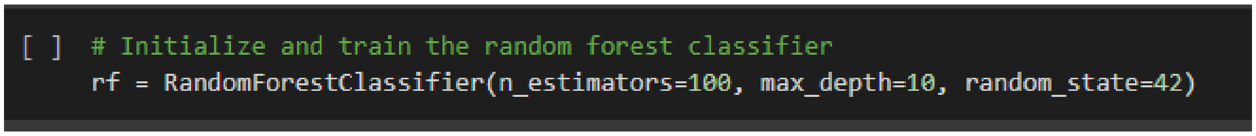


7. Random Forest - Classifier:
   This algorithm builds multiple trees and combined their predictions to produce an outcome.

We then trained all the models to our dataset to check which algorithm classifies the output variable correctly.
Out of all the algorithms used, Random Forest - Classifier shows the best accuracy.

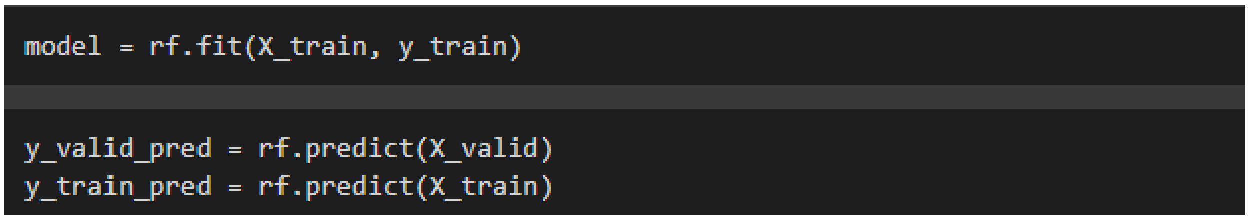
Let me take it deep into the Random Forest Algorithm.

There are certain hyperparameters in the Random Forest Classifier that are responsible for changing the model's behavior like n_estimators,max_depth, and random_state.



```
[ ]  # Initialize and train the random forest classifier
     rf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
```

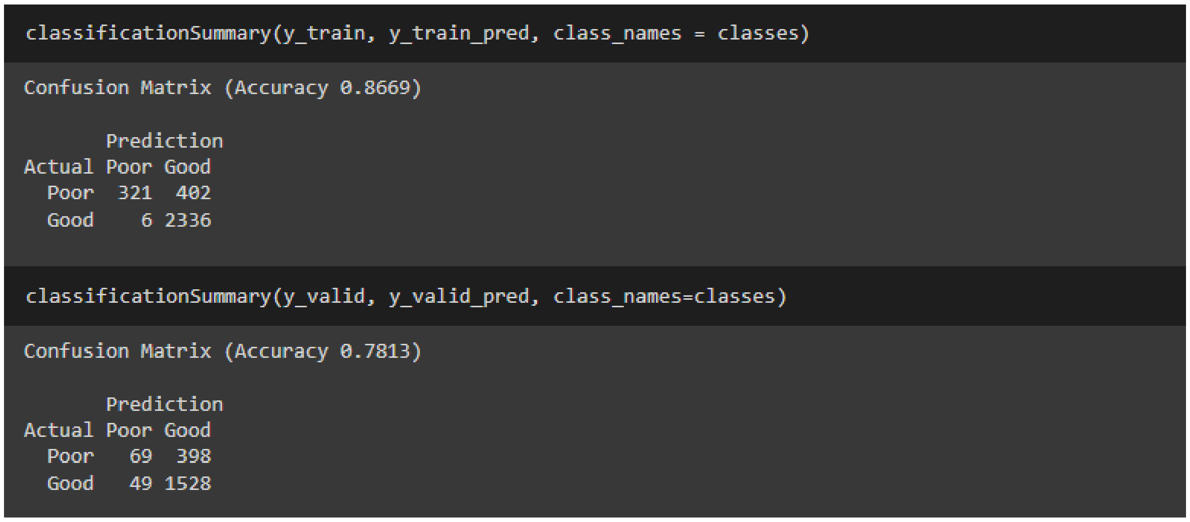
The model is trained using fit() method and further used for prediction using predict() method.



```
model = rf.fit(X_train, y_train)
```

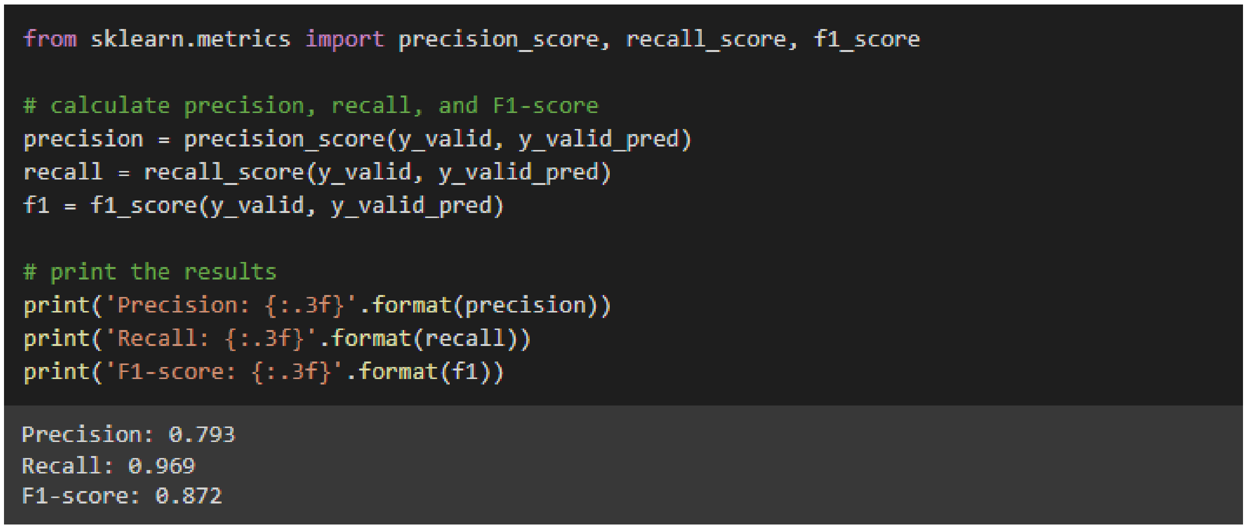
```
y_valid_pred = rf.predict(X_valid)
y_train_pred = rf.predict(X_train)
```

The training data's actual labels are given as y_train argument, while their anticipated labels are represented by y_train_pred. A list of the class names for the classification problem is contained in the class_names argument.



```
classificationSummary(y_train, y_train_pred, class_names = classes)

Confusion Matrix (Accuracy 0.8669)

       Prediction
Actual Poor Good
  Poor  321  402
  Good    6 2336

classificationSummary(y_valid, y_valid_pred, class_names=classes)

Confusion Matrix (Accuracy 0.7813)

       Prediction
Actual Poor Good
  Poor   69  398
  Good   49 1528
```

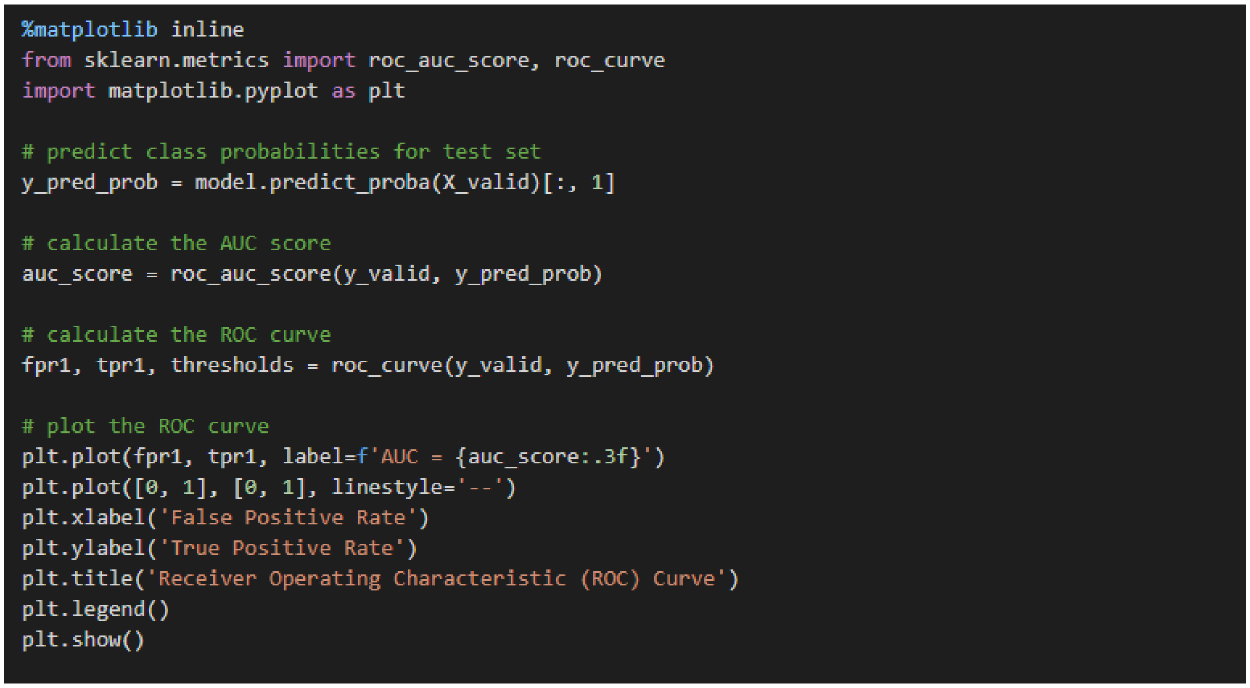
The precision_score, recall_score, and f1_score functions from the sklearn.metrics module are used to calculate and output the precision, recall, and F1-score of a classification model on the validation data.



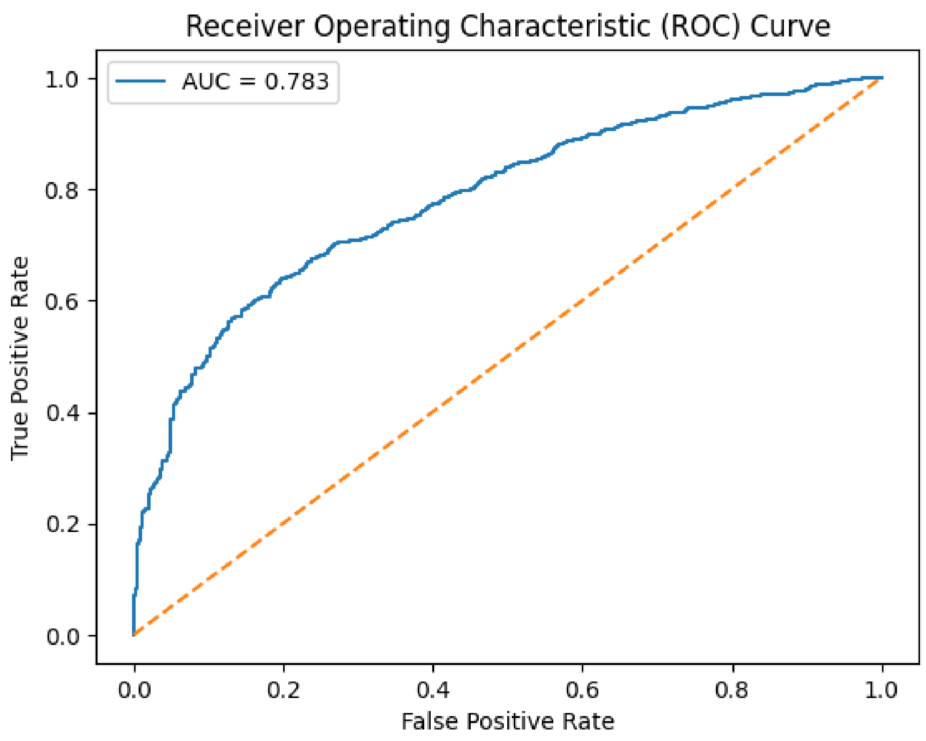
```python
from sklearn.metrics import precision_score, recall_score, f1_score

# calculate precision, recall, and F1-score
precision = precision_score(y_valid, y_valid_pred)
recall = recall_score(y_valid, y_valid_pred)
f1 = f1_score(y_valid, y_valid_pred)

# print the results
print('Precision: {:.3f}'.format(precision))
print('Recall: {:.3f}'.format(recall))
print('F1-score: {:.3f}'.format(f1))
```

```
Precision: 0.793
Recall: 0.969
F1-score: 0.872
```

It uses the matplotlib.pyplot library for visualization along with the roc_auc_score and roc_curve functions from the sklearn.metrics module in scikit-learn to plot the receiver operating characteristic (ROC) curve of a binary classification model on the validation data.



```python
%matplotlib inline
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# predict class probabilities for test set
y_pred_prob = model.predict_proba(X_valid)[:, 1]

# calculate the AUC score
auc_score = roc_auc_score(y_valid, y_pred_prob)

# calculate the ROC curve
fpr1, tpr1, thresholds = roc_curve(y_valid, y_pred_prob)

# plot the ROC curve
plt.plot(fpr1, tpr1, label=f'AUC = {auc_score:.3f}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

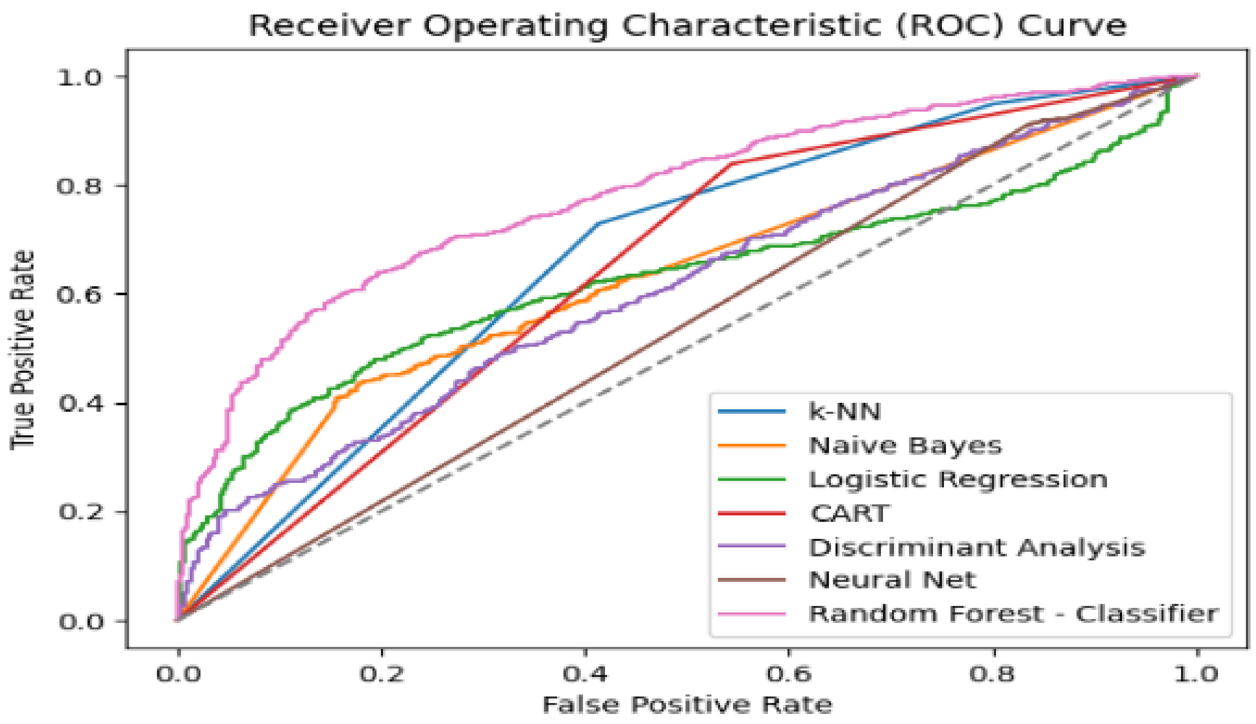


This figure gives us the best visualization of the Random Forest Classifier Model's accuracy.

Performance Evaluation:

When samples are projected to be positive, precision is the percentage of real positives.
Recall counts the percentage of samples that actually match the predictions of being positive.
The F1-score is a balanced indicator between precision and recall because it is the harmonic mean of both. An excellent F1 score means the model has both exceptional precision and recall.

Models Names	**Precision**	**Recall**	**F1-Score**
k-NN	85.6%	72.9%	78.7%
Naive Bayes	89.6%	39.4%	54.7%
Logistic Regression	77.3%	99.7%	87.1%
CART	83.9%	83.8%	83.9%
Discriminant Analysis	77.3%	99.8%	87.1%
Neural Network	77.2%	99.8%	87%
Random Forest	79.3%	96.9%	87.2%



The above figure shows the ROC Curve of the all the algorithms used. We can clearly see that Random Forest - Classifier has the best curve and thus provides the best accuracy.

Project Results:

It is identified that the random forest classifier gives an accuracy of 78.13% which is the highest among all models. An accuracy of 86.69% on the training data indicates that the model is able to correctly classify 86.69% of the samples in the training set. However, an accuracy of 78.13% on the testing data suggests that the model is not performing as well on new, unseen data. Also, it indicates that the model is fitting the noise and the variations in the training data. Due to adaptation to the old data, the model is unable to perform well on the new one.
The difference in performance between the training and testing sets is an indication of overfitting, which occurs when a model learns to fit the noise in the training data rather than the underlying pattern, leading to a poor generalization of new data.

Impact of the Project Outcomes:

The impact can be largely on the application developers who have the option to improvise regularly based on the customer’s feedback, number of downloads, and revenue generated. It can also provide a proper insight into the recommendation algorithm on the playstore which may further boost the user’s rating for an application. After initializing a user’s rating from a particular model based on accuracy, a company can invest further in the upcoming versions and also launch paid services for certain things. This may increase the revenue.

Suggestions:

Even though the model is 78% accurate, to address overfitting, one is to use regularization techniques such as:

- Use more data for training
- Use data augmentation techniques to generate additional training samples
- L1 and L2 regularization