How to make apps popular on Google play store: Visual analysis of google play store app features

by

Nupura Walawalkar

A Project Report Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
Master of Science
Supervised by

Professor Erik Golen

School of Information

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

May 2023

The project report "How to make apps popular on Google play store: Visual Analysis of google play store app features" by Nupura Walawalkar has been examined and approved by the following Examination Committee:

Professor Erik Golen
Senior Lecturer
Project Committee Chair

Professor John-Paul Takats
Visiting Lecturer

Dedication

...

Acknowledgments

I am grateful to my family

Abstract

How to make apps popular on Google play store: Visual analysis of google play store app features

Nupura Walawalkar

Supervising Professor: Professor Erik Golen

People frequently use mobile applications for communication, traveling, education, work, and leisure activities. Google Play Store and Apple Appstore are the two major app stores due to many apps and the popularity of Apple and Android devices. There are approximately 2.5 billion Android users base is and 1 billion Apple users as of 2021. The Android app ecosystem has evolved and flourished because android apps are supported on various devices. Many innovative apps are published daily, but only some have become very popular among users over time. Google Play store offers good guidelines for developers to create appealing app store listings. However, it is yet to be determined which parts of the app advertisement persuade users to download and rate the app. It would be valuable to analyze the google play store app advertisements to help the developers understand user preferences while selecting features for an app. Previous work related to this area only evaluated specific aspects of the app store listing, such as app description, pricing, reviews, user-developer dialog in thoughts, and permissions. Still, studying the combination of different parts of an app advertisement would be valuable. This project intends to identify which parts of an app store listing boost ratings and app downloads by analyzing app features using a random forest classifier and performing a sentimental analysis of reviews. The analysis performed in this project will provide an insightful dashboard for developers to understand user preferences for making their apps popular.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Background

The mobile applications market has been one of the fastest-growing segments of the software application market. A study by MarketWatch shows that people in most developed countries spend an average of 70% of their digital minutes on mobile apps (MarketWatch, 2022). People use mobile apps for different purposes, such as banking, shopping, social media, and photography. Many companies provide their services through mobile apps to lure many consumers. Companies use mobile apps to gain a competitive edge over their competitors. They provide their customers with simple, contactless, personalized experiences through their apps. Android OS seems to be the most popular smartphone OS, with a market share of 71.5 % as of Q1 of 2022, outperforming iOS Apple's smartphone OS, which has 27.68 % (MarketWatch, 2022).

The Android OS app store, Google Play Store, provides a good platform and SDKs for developers to deploy apps. Android software development kit offers an easy-to-use integrated development environment for developers to create apps. Android apps are supported on different types of devices. More than 2 million apps are available on Google Play Store for download. About There is intense competition among businesses to publish their apps on the google play store. Time-to-market plays an essential role in consumer engagement and consumer retention, increasing the company revenue by increasing the number of app installations. To remain listed on the app store and increase app installations, app developers must follow Google Play's security guidelines and procedures, such as scanning apps for malicious behavior, before publishing their apps. Google play store often removes malicious apps if there it has low ratings, terrible reviews, or user complaints. New app developers might be interested to know they should design and market their app and which features might lead to it getting featured in the top or trending apps. App users often express their experience using the app, request new features and complain about app updates in reviews.

Recently, researchers have also started taking advantage of the popularity of app markets and published app prototypes to collect feedback from various app users (Zhai et al., 2009). Therefore, researchers and app developers must get insights into the mobile application market and consumer behavior to stay ahead of the competition.

Google play store defines apps into different buckets depending on the minimum number of installations of the app. When an app is newly published or updated, it appears in the list of recently added or updated apps on the Google App store. A few users might install an app when a developer releases it initially. It needs to be clarified how the app advertisement needs to be written to appeal to a broad audience.

1.2 Dataset

To understand this problem, I have used Google play store data provided by Gautham Prakash on Kaggle (Prakash, 2021) was used. The dataset is from June 2021 and contains records for 2.3 million apps. It has 23 features: name, category, type(paid/free), rating, minimum and maximum installs, editor's choice, reviews, size, last updated, and supported android version. In addition, app descriptions and reviews were collected for about 700k apps from this dataset.

1.3 Objective

1. Google Playstore data exploration: I explored the original dataset and extracted additional features such as app descriptions and reviews using the Google play store scraper python package. The data exploration contains bar charts and pie charts explaining the relationship between different attributes. This section also includes how missing data were handled. This section concludes with information about attributes useful for building the model and for sentimental analysis.

2. Tableau dashboard for data visualization: The Tableau dashboard will provide users with a comprehensive overview of different statistics that make apps popular.

3. Train multiple classifier models to identify features of popular apps: Using the features identified in data exploration, I have trained multiple classifier model and selected random forest model to identify which features are essential for improving app installations.

4. Perform sentimental analysis on app reviews to find user opinions about popular and less popular apps: This section includes analyzing the polarity and subjectivity of sentiments in app reviews.

The rest of the report is organized into 4 sections: section 2 for previous work on random forest classifier and sentimental analysis, section 3 for methodology, section 4 for results, and section 5 for concluding remarks.

2 Related Work

Hundreds of apps in different categories, such as games, family, and entertainment, are published daily on the Google Play store. However, only some apps have become popular among users. Companies often promote their apps on social media such as Facebook and Instagram through paid digital marketing campaigns (Play, 2022). These app promotions also help to improve the visibility of apps. Apps become popular when more users can discover newly published apps. Google Play provides guidelines for developers to publish apps and describe them using keywords to make them easily searchable in the Google Play Store (Google, 2021). A striking title, a description focused on what users want from the application, and high-quality graphic images and videos of an app are some strategies for making the app stand out in search results. These guidelines, however, do not ensure that users will install the app.

A wide-ranging study by He Jiang et al. (Jiang et al., 2015) employs crowdsourcing to identify which features of an app store listing are more helpful to users. This paper focuses on improving app descriptions to attract more users to install the app. The authors gathered feedback and scores from participants in their experiment to build an SVM model to identify which features of the app description benefit users. This study only focuses on one aspect of the app advertisement on google play. Several other factors affect app downloads by users. X.Wei et al studied how the Google Play app store has evolved regarding evaluating app permissions (Wei et al., 2012). Another research by Taylor and Martinovic (Taylor & Martinovic, 2017) showed the evolution of permissions required for free and paid apps and how permission-hungry apps affect user decisions to download the app. Users might be hesitant to download apps that require unwarranted licenses. B. Carbunar et al. have studied how app pricing affects app downloads (Carbunar & Potharaju, 2015). Another study provides critical insights into how a good app description with searchable keywords makes them discoverable and influences users to download the app.

Niels Henze and Susanne Boll (Henze & Boll, 2011) analyzed a dataset of 150K game installations and observed 24k published apps to determine the best time of the week to deploy apps that will increase app installations. They observed that apps released on Sunday performed best in the app market. Saad et al. (Saad & Nanath, 2020) performed sentimental analysis using a support vector machine(SVM), Naïve Bayes classifier, and K-nearest neighbor models on app reviews to identify popular apps. P. B. Prakash Reddy et al. (Reddy & Nallabolu, 2020) analyzed apps in each category, size of apps, reviews, and price and identified that the size of apps somewhat impacts downloads.

There is an opportunity for study in which combining these factors helps increase app installations. Random forests classifier is a stable and robust prediction model which provides accurate results by building and merging multiple decision trees. Jingxian Zhang et.al (Jin & Liu, 2010) used a similar approach where they used random forest regression to measure how expensive cars are and to watch brands look based on their websites. M. Nayebi et al. analyzed the marketability of F-Droid apps (open-source android app store) using three machine learning models. They found random forest classifier performed best with an F1 score of 78% to identify marketable release.

While analyzing the app advertisement might help create an excellent technical and business strategy, we should also consider the user's experience with the app (Carreño & Winbladh, 2013). An app user's review describing their experience using the app might intentionally or unintentionally affect app downloads Ware, 2012. Google Play

Store stores a large number of app reviews for each app. These reviews range from compliments to complaints about app updates and features. It can be tricky to filter positive reviews from negative reviews. In such a scenario, the sentimental analysis might be helpful to mine people's opinions and attitudes towards the app.

Sentimental analysis has three types of classification levels: document level, sentence level, and feature level analysis. Sentimental analysis involves subjectivity and polarity analysis of text to determine how positive, neutral, or negative it is. There are two types of sentimental analysis techniques: machine learning and lexicon-based techniques (Agrawal et al., 2021). Machine learning techniques include the Naive Bayes classifier, SVM, neural networks, rule-based classifiers, lexicon-based approaches such as a dictionary-based process, the manual opinion method, and a corpus-based process. Machine learning-based approaches use labeled datasets to predict the sentiment of the text. Lexicon-based method calculates the sentiment orientation of a set of sentences in the document using a dictionary with each labeled as positive, negative, or neutral sentiments along with polarity, parts of speech and subjectivity classifiers, mood, and modality. The sentiment orientation can be positive, negative, or neutral. The overall sentiment in the document is scored using a function that takes the sum or average of positive and negative words in the text.

3 Methodology

This section consists of 5 parts: Data collection, data preprocessing and feature selection, building a random forest model, sentimental analysis, and data visualization using the tableau dashboard.

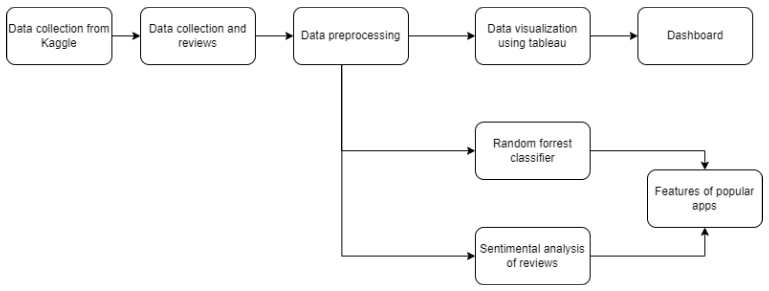

Data collection from Kaggle
Data collection and reviews
Data preprocessing
Data visualization using tableau
Dashboard
Random forrest classifier
Features of popular apps
Sentimental analysis of reviews

Figure 1. Research process

3.1 Data Extraction

The Google Play store dataset provided by G. Prakash on Kaggle contained several fields such as the app names, category of apps, ratings, number of installations, developer details, and android version. The dataset consisted of a single CSV file. The app descriptions and reviews not present in the original dataset were extracted using the google-play-store-scraper python package.

3.2 Data preprocessing

The following table summarizes the features in raw data.

Table 1. Initial feature description

Google play store data initial features			
Field	Description	Type	No. of records (before cleaning)
App Id	Unique identifier of app	String	764466
Category	Type of service provided by the app. There are different categories of apps such as games, tools, entertainment, music, weather etc.	String	764466
Rating	Rating given by users for an app. App rating ranges from 1-5 stars	Numeric	757067
Rating count	Number of ratings given by users who installed the app	Numeric	757067

Continuation of Table 2

Installs	Google play store defines brackets for the number of installations such as 1+ , 5+, 10+, 100+ . These brackets indicate the minimum number of users who installed the app.	String	764431
Maximum Installs	Actual number of installations	Numeric	764431
Minimum Installs	Google play store defines brackets for the number of installations such as 1+ , 5+, 10+, 100+ . These brackets indicate the minimum number of users who installed the app.	Numeric	764466
Free	Indicates whether the app is free or paid	Boolean	764466
Price	Price of app	Numeric	764466
Currency	Currencies supported for paying app's price	String	764426
Size	Size of App in kb, MB and GB	String	764460
Minimum Android Version	Minimum Android version supported by app	String	762006
Developer Id	Developer's unique identifier	String	764463
Developer Website	Developer's web page	String	530958
Developer Email	Developer email	String	764462
Released	Release date of app	String	739810

Continuation of Table 2

Privacy Policy	webpage with details of how app protect user privacy while using the app	String	764466
Last Updated	date when the app was last updated	String	764466
Content Rating	Indicates the type os user the app is suitable for. Content rating can be , Everyone, Teen, Adults only 18+ , Mature 17+, unrated	String	633545
Ad-Supported	Many apps display ads when a user is using the app. The ads help to generate revenue. This flag indicates to user whether the app supports ads.	Boolean	764466
In-app purchases	Indicates whether the app provides in app purchases.	Boolean	764466
Editor Choice	Indicates which apps were chosen by editorial team of google playstore. This flag helps to highlight new and innovative apps on google play store page	Boolean	764466
description	summary of app's features	String	764466
descriptionhtml	summary of app's features	String	764466
description	summary of app's features	String	764466
video	url of youtube video of app. It gives a preview of the app interface	String	72009

Continuation of Table 2			
screen-shots	URL of images of app interface	String	764466

After extracting additional data and combining the datasets, a few columns in raw data contained some valuable data in different formats and garbage data. Categorical and string columns must be converted into a numeric format to build a machine-learning model efficiently.

1. Title Length: The app name column was in string format. It consisted of alphanumeric characters and special characters such as emojis. A new field was created by extracting the number of words in App Name to analyze if it is an important feature in marketing the app.

2. Size: The size of different apps were kilobytes, megabytes, and gigabytes. About 26,692 apps had a 'Varies with device' size. The app size was converted to kilobytes and the 'Varies with device' size was converted to 'NA.'

3. No of videos: The dataset contained a list of URLs of youtube videos created to allow users to preview the app. A new field containing the number of videos created for an app.

4. No of images: The dataset contained a list of URLs of promotional images created to allow users to preview the app. A new field containing the number of images/screenshots created for an app.

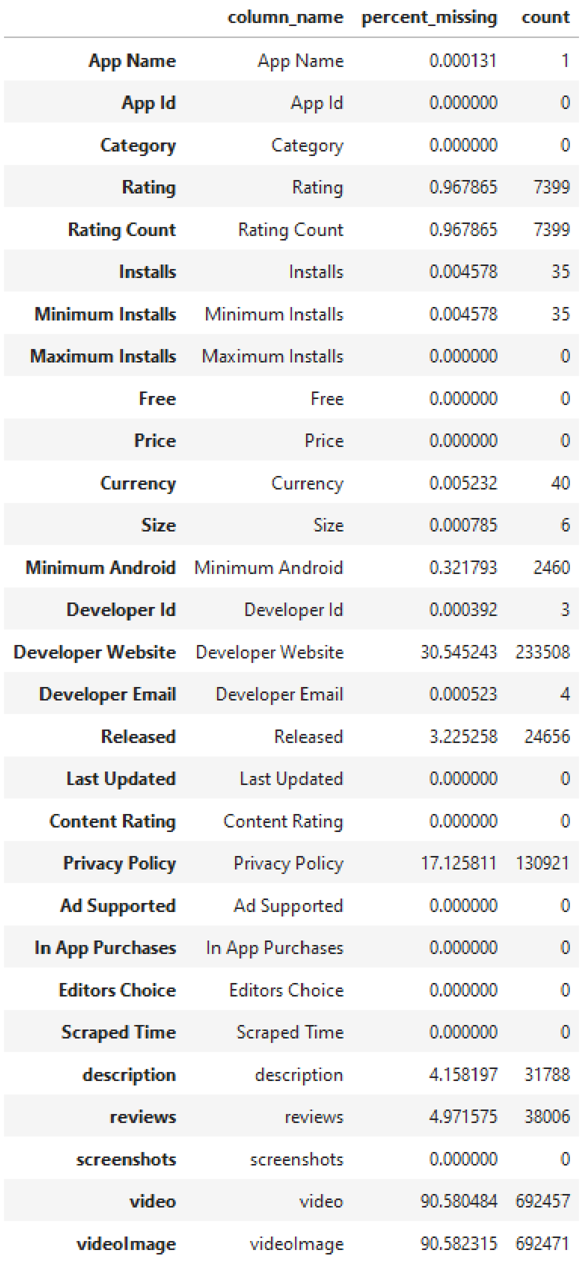
5. Released day of the week: The dataset contained an app's 'Released date.' A new field was created that stated the day of the week based on the Release date. E.g.: If an app was released on '2021-03-02', the released day of the week would be 1.

6. Content rating: The content rating column consisted of categorical values. The values were encoded using one-hot encoding to create dummy/indicator variables representing the different ratings.

7. Installs: Each app is grouped into various categories, such as 10+, 50+,100+, and 500+, depending on the number of maximum installs of the app. If an app had 1500 installs, then it was placed in the 1000+ category. The installs field was converted from string to numeric by removing the '+' sign at the end of the string.

8. Installs bucket: Classification for this dataset will be performed based on the number of installations. For this purpose, a new target variable, "Installs_bucket" was created based on the Installs column. Apps with greater than 50,000 installations were grouped in the "high" class, and apps with less than 50,000 installations were grouped into the "low" class.

3.2.1 Missing value detection and imputation



	column_name	percent_missing	count
App Name	App Name	0.000131	1
App Id	App Id	0.000000	0
Category	Category	0.000000	0
Rating	Rating	0.967865	7399
Rating Count	Rating Count	0.967865	7399
Installs	Installs	0.004578	35
Minimum Installs	Minimum Installs	0.004578	35
Maximum Installs	Maximum Installs	0.000000	0
Free	Free	0.000000	0
Price	Price	0.000000	0
Currency	Currency	0.005232	40
Size	Size	0.000785	6
Minimum Android	Minimum Android	0.321793	2460
Developer Id	Developer Id	0.000392	3
Developer Website	Developer Website	30.545243	233508
Developer Email	Developer Email	0.000523	4
Released	Released	3.225258	24656
Last Updated	Last Updated	0.000000	0
Content Rating	Content Rating	0.000000	0
Privacy Policy	Privacy Policy	17.125811	130921
Ad Supported	Ad Supported	0.000000	0
In App Purchases	In App Purchases	0.000000	0
Editors Choice	Editors Choice	0.000000	0
Scraped Time	Scraped Time	0.000000	0
description	description	4.158197	31788
reviews	reviews	4.971575	38006
screenshots	screenshots	0.000000	0
video	video	90.580484	692457
videoImage	videoImage	90.582315	692471

Figure 2. Percentage of missing values in each feature

The Google Playstore data set provided by Kaggle and the additional data extracted using the google playstore scraper contained some missing values as shown in Figure 2. These missing values were distributed across different categories. More than 90% of the apps in the dataset didn't have promotional videos. Promotional videos are not mandatory for publishing the app on the google play store, but the app encourages developers to add a promotional video to improve user engagement. The release date was missing for around 3.22% app. Other textual data, such as the privacy policy of 17% apps and the developer website of 30% apps, is missing.

Missing values can result in low-quality data and unreliable data mining results. Finding and imputing missing values in the dataset is necessary to create an efficient and robust data mining model. The authors Makaba et al (Makaba & Dogo, 2019) suggest several strategies for handling missing data. Deleting/Discarding data works if the data point is insignificant; otherwise, it may lead to data loss. Replacing values with mean or median values is more suitable for numerical data, but it doesn't factor in the correlation between different features and can give incorrect results. Replacing values with mode or most frequent values works well with categorical data but also introduces bias in the data. An alternate method is to use k-NN algorithm, which uses feature similarity to predict missing values. It finds the closest neighbors to the observation record with missing data and then imputes missing values based on the records with non-missing values. Although this method is more accurate than previous methods, it is computationally more expensive for large datasets. A recent method called Multivariate imputation by chained equation(MICE) (van Buuren & Groothuis-Oudshoorn, 2011) fills missing values multiple times. This method can effortlessly impute a missing value for different types of variables.

Based on the above information,the data imputation step first started by removing 35 rows with missing 'Installs' and 26,692 records having 'Varies with device' size was removed. The MICE approach imputed missing values in numerical fields such as Rating, Rating Count, No of videos, and reviews. The Iterative Imputer python package imputed 692445 values for no of videos, 38006 for reviews, and 7399 for Rating and Rating count. The final dataset after imputing missing values contained 737739 apps.

3.3 Google play store data exploration

The google play store has different categories of apps. Education category is quite common in this dataset. Books and Reference ,Music and Tools apps also seem to be popular. Tools and casual, arcade, and action gaming apps are the most installed apps.

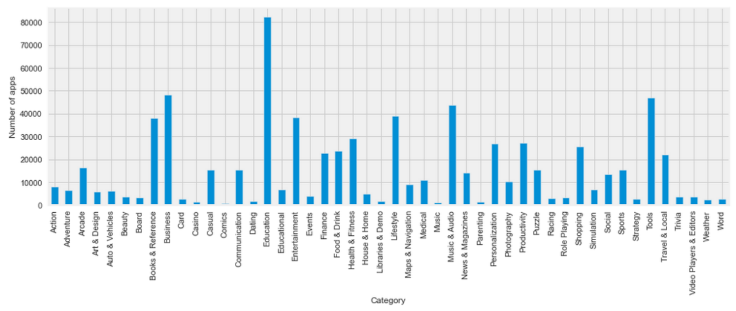


Figure 3. Number of Apps in each Category

Google play store tracks the maximum and minimum installations of apps and groups them depending on the number of minimum installations. Figure 4 shows the distribution of apps according to these groups. The majority of apps have less than 10-50000+ installations. There are only so many apps that have higher than 50000 installations.

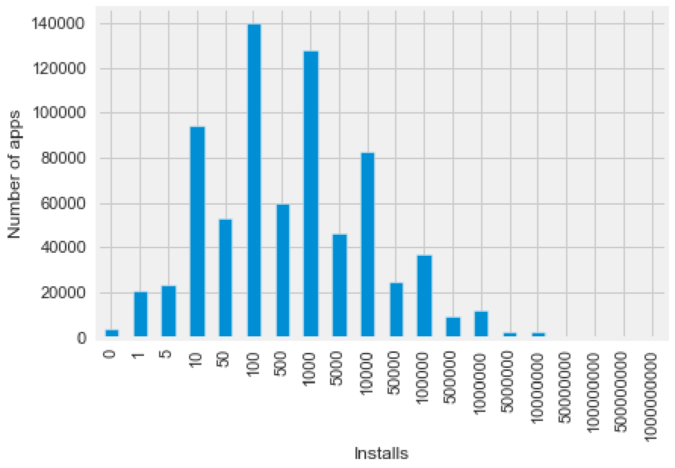


Figure 4. Distribution of app installations

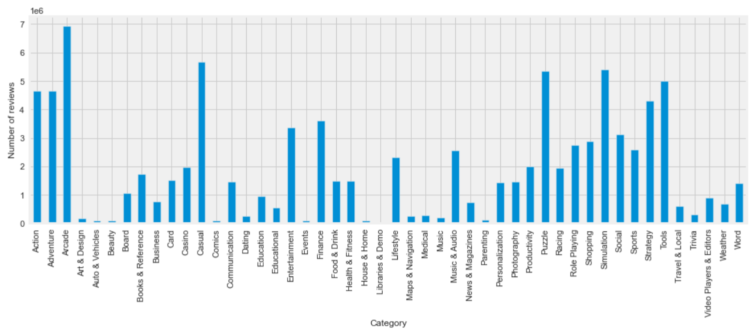


Figure 5. Average number of reviews for apps in each category

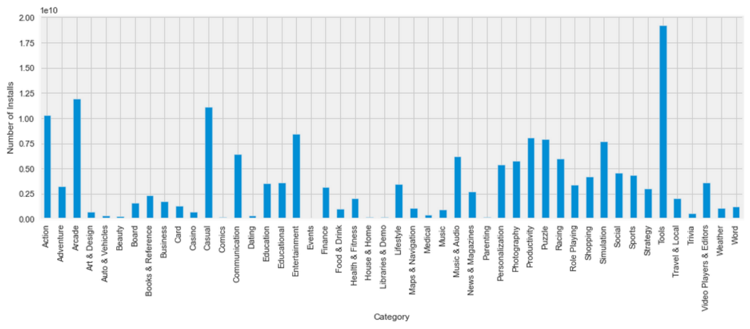


Figure 6. Average app installations in each category

Although Tools apps have the most installed, gaming apps like action, adventure, arcade, and puzzle games are the best-performing apps because they get more user responses through ratings and reviews. Maps and navigation, medical, music, art and design, and events are the least-performing apps. Since reviews are essential for developers to gauge user satisfaction, users mention their experience with the app's different features and rate it. These reviews often help other users decide whether to install the app. Further analysis from users' points of view about different app features, such as app name, sizing and pricing strategy, ad support, and in-app purchases, might help to understand what influences users to install an app.

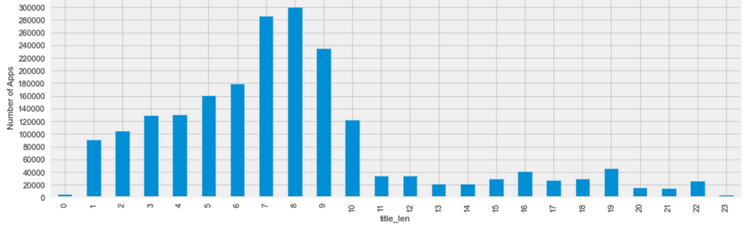


Figure 7. Effect of length of App Name on Installs

The Google play store suggests that developers keep app names as short and easily searchable as possible. Figure 7 indicates that app names having eight or fewer words have more installations than apps with longer names.

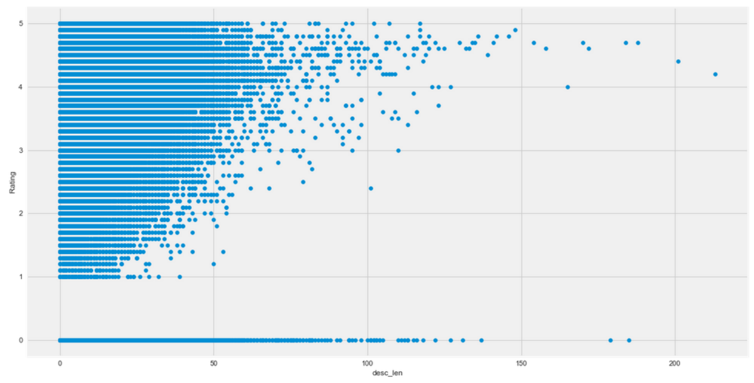


Figure 8. Effect of length of App Description on Installs

Figure 8 indicates that users highly rated apps with shorter descriptions (less than 50 sentences). Very few apps with lengthy descriptions received ratings of more than three stars. App descriptions that listed the features and permissions in a structured format required by apps received good ratings.

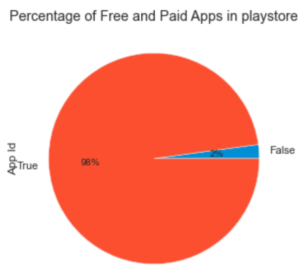


Figure 9. Percentage of Free /Paid Apps

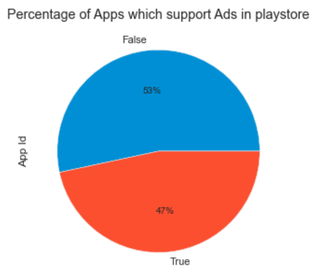


Figure 10. Percentage of Apps which support ads

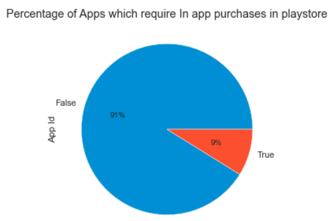


Figure 11. Percentage of Apps with In-app purchases

The primary motivation of developers behind increasing app installations is to increase their customer base and revenue. Developers can earn revenue by making paid apps, displaying ads, or asking users to purchase items or services while using their apps. Figures 9,10, and 11 show that Free apps dominate the Android app market—about 47% of apps in the dataset support ads. The users seem to be more tolerant of apps that display ads than paid apps and apps that require In-app purchases. It is easier for users to install the app, explore the features, and then pay for services as required. Figures 12 and 13 show that apps supported and apps with in-app purchases have higher ratings than other apps.

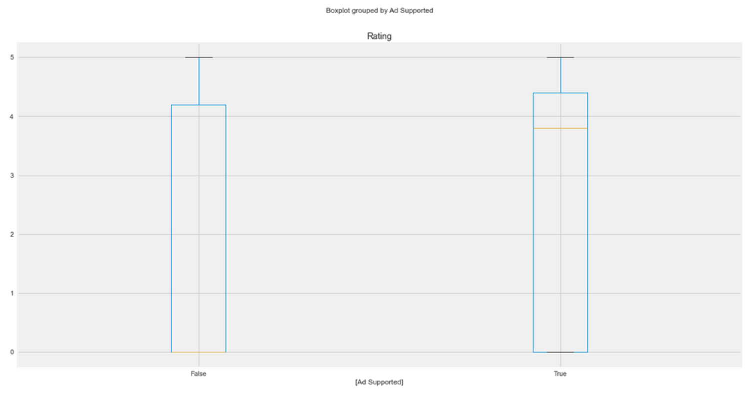


Figure 12. Rating of Ad-supported apps

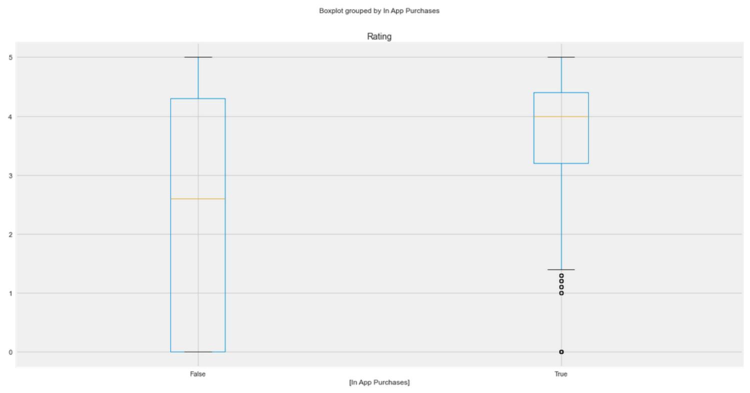


Figure 13. Rating of Ad-supported apps

Google Play Store has a feature called 'Editors Choice' for some apps. There are about 309 Editors' choice apps out of 737k apps. New apps with innovative features are added to the editor's choice apps. These apps are featured on the Google Play Store and recommended to users. Figure 14 shows that editor's choice apps received a high rating of 4.3.

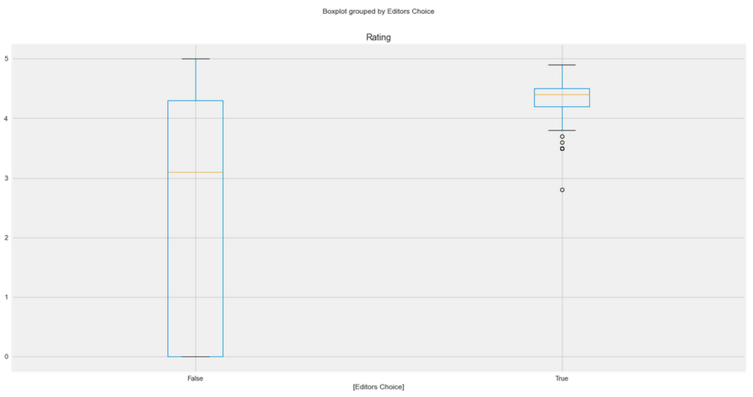


Figure 14. Rating of Editors choice apps

Apart from the cost of the app, users also look at the size of the app. Mobile devices have a limited amount of space. Bulky apps can affect the performance of the other apps on the mobile device. Users might install bulky apps with good graphics and plenty of innovative features. Figure 15 shows that lightweight apps have a higher rating than bulky apps, with few exceptions. The average app size in the dataset is 200MB.

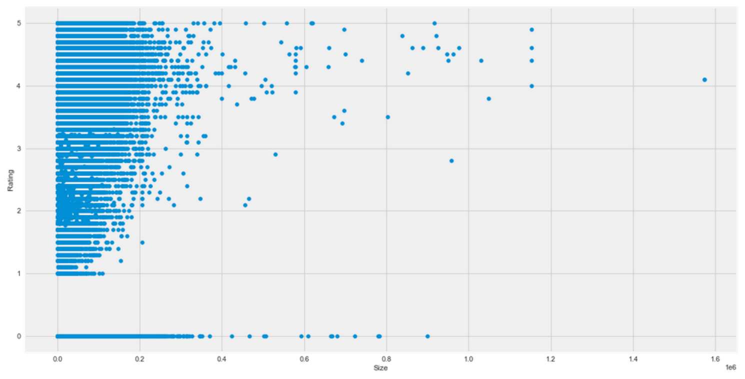


Figure 15. Rating of App according to size

Figure 16 shows the relationship between essential variables such as Size, Price, Rating, and the number of reviews. The popular apps are in orange, and the less popular apps are in blue. The size and price of the app are negatively correlated with the rating. Apps rating decreases with an increase in size or price. The number of reviews is positively correlated with ratings for popular apps. Size and price have a weak negative correlation for less popular apps.

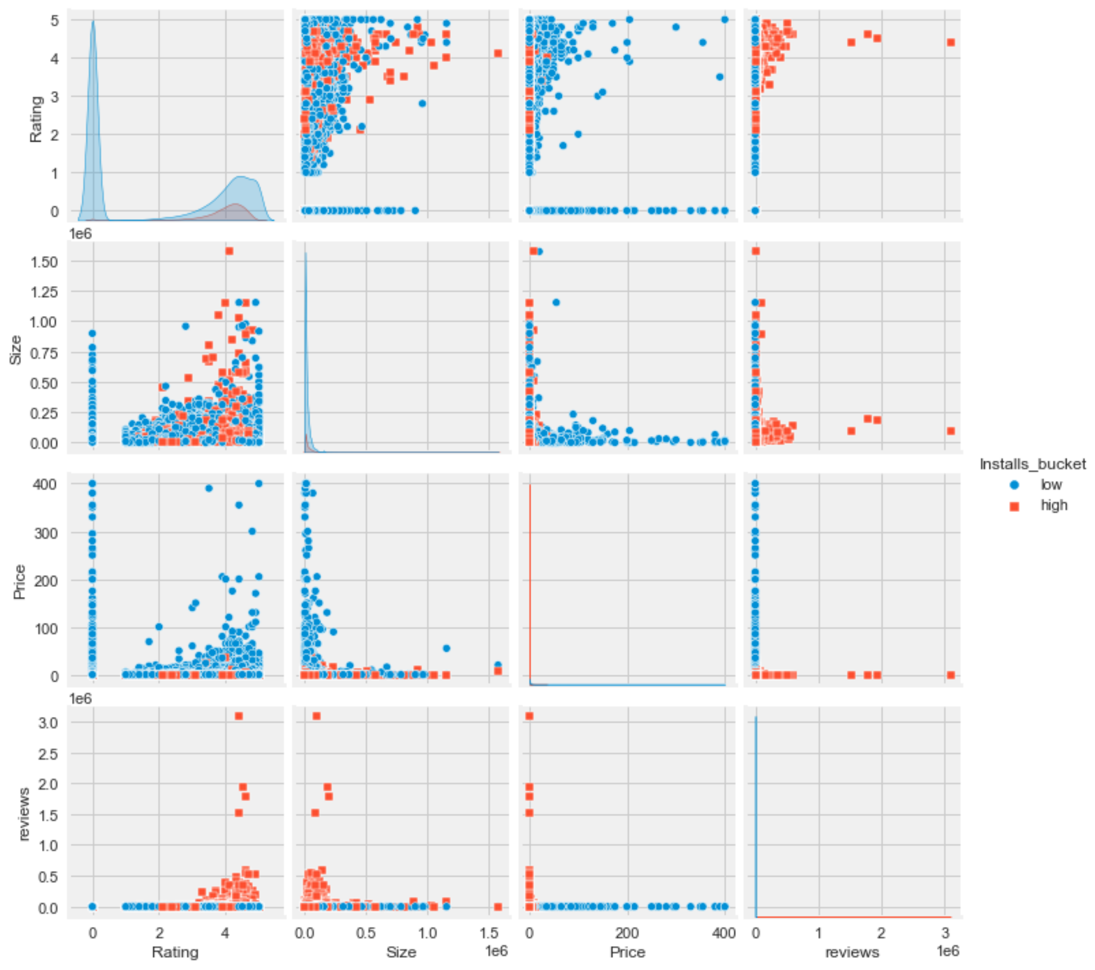


Figure 16. Relationship between variables Rating, Size, Price, reviews, Installs_bucket

3.4 Feature selection

Feature selection is one of the essential steps in building a machine-learning model. This process involves reducing the number of input variables in the dataset and minimizing the loss of information to build an efficient machine-learning model. Principal component analysis (PCA), a type of dimensionality reduction technique, was selected to extract this dataset's features. PCA identifies crucial relationships between variables in the dataset, performs a linear transformation on a dataset, and quantifies the relationships.

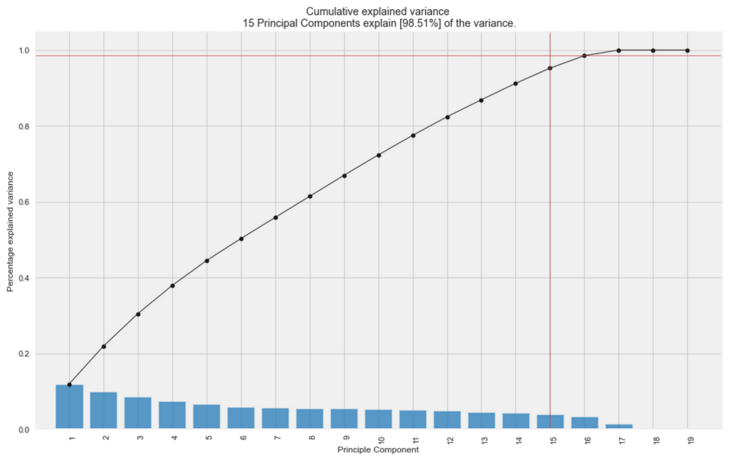


Figure 17. Cumulative explained variance plot

Figure 17 shows a cumulative explained variance plot which contains variance for the initial 19 components in the dataset ranked according to theier explained variance. There is a line that marks the threshold for 95% explained variance. We can see how the curve flattens slightly around 16 or 17 components. The line drawn for 95%, the total explained variance is at approximately 15 components. This means the first 15 components contain most of the information.

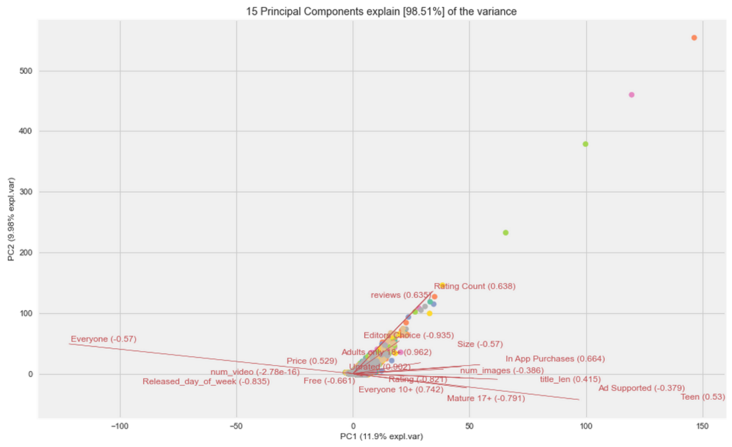
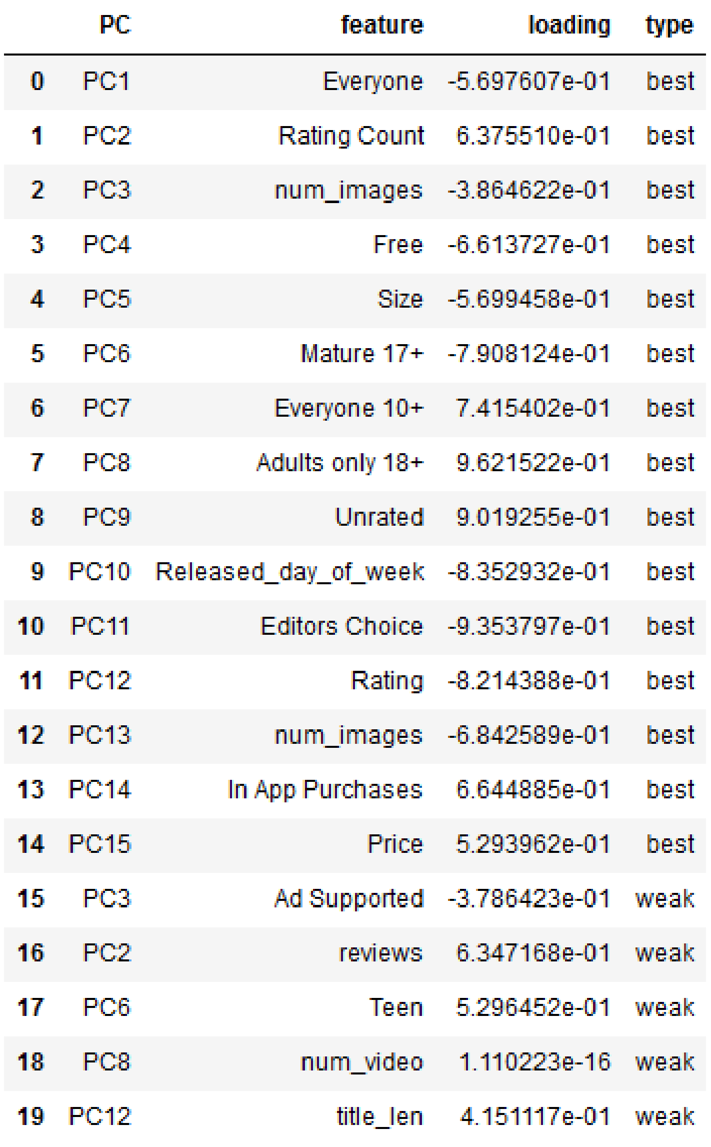


Figure 18. PCA biplot

Figure 17 shows a cumulative explained variance plot that contains variance for the initial 19 components in the dataset ranked according to their explained variance. There is a line that marks the threshold for 95% explained variance. We can see how the curve flattens slightly around 16 or 17 components. The line drawn for 95%, the total explained variance is at approximately 15 components. It means that the first 15 components contain most of the information.



	PC	feature	loading	type
0	PC1	Everyone	-5.697607e-01	best
1	PC2	Rating Count	6.375510e-01	best
2	PC3	num_images	-3.864622e-01	best
3	PC4	Free	-6.613727e-01	best
4	PC5	Size	-5.699458e-01	best
5	PC6	Mature 17+	-7.908124e-01	best
6	PC7	Everyone 10+	7.415402e-01	best
7	PC8	Adults only 18+	9.621522e-01	best
8	PC9	Unrated	9.019255e-01	best
9	PC10	Released_day_of_week	-8.352932e-01	best
10	PC11	Editors Choice	-9.353797e-01	best
11	PC12	Rating	-8.214388e-01	best
12	PC13	num_images	-6.842589e-01	best
13	PC14	In App Purchases	6.644885e-01	best
14	PC15	Price	5.293962e-01	best
15	PC3	Ad Supported	-3.786423e-01	weak
16	PC2	reviews	6.347168e-01	weak
17	PC6	Teen	5.296452e-01	weak
18	PC8	num_video	1.110223e-16	weak
19	PC12	title_len	4.151117e-01	weak

Figure 19. Best performing features

This technique selected Everyone, Rating Count,'Rating', 'Free' 'Price', 'Size', 'In App Purchases', 'Editors Choice', 'Released_day_of_week','Adults only 18+', 'Everyone', 'Everyone 10+', 'Mature 17+', 'Unrated', 'num_images' as important features.

3.5 Building a classifier model

The Random Forest Classifier Algorithm is a popular and sturdy supervised machine learning algorithm that constructs multiple decision trees on training samples. This algorithm selects a random sample of m predictors from the full set of p predictors as split candidates. Each tree is grown independently on random samples of the observations. However, the split on each tree is conducted using a random subset of the features to decorrelate the trees and explore the model space more comprehensively than bagging. The prediction accuracy increases with more trees in the forest, enabling the modeling of multiple decision trees. To achieve this, the bootstrapping technique is utilized. Instead of using the entire training data to build a single tree, numerous samples of equal size are created through sampling with replacement. These samples are then employed to construct individual trees.

Before building a machine learning model, examining the distribution of classes in the dataset is essential. Data imbalance is when a dataset has an unequal distribution of classes. If we train a binary classification model without addressing this issue, the model will be entirely biased toward the majority class. Furthermore, it affects the relationships between features.

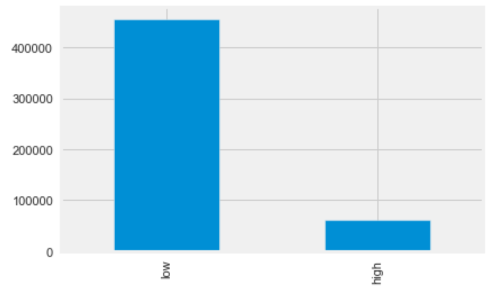
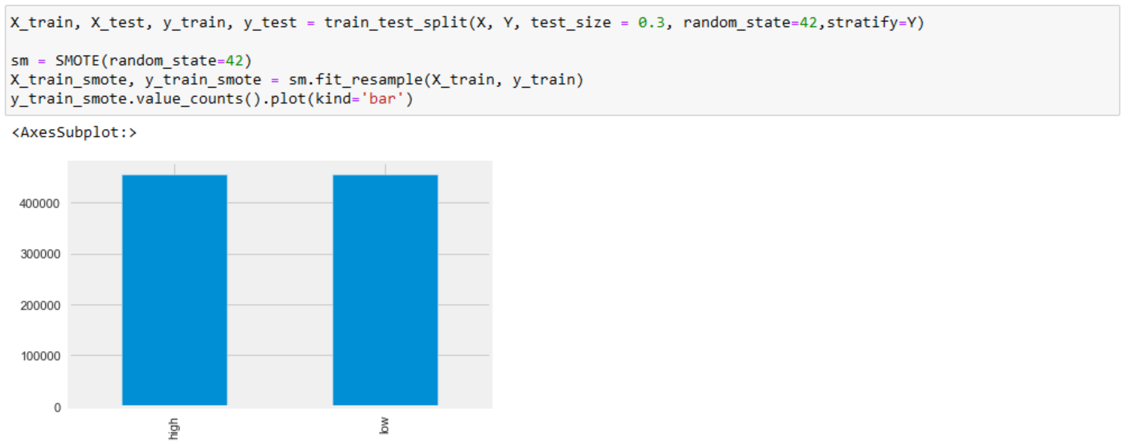


Figure 20. Class distribution before smote

Figure 20 shows the class distribution for the Google play store dataset. There are significantly less than popular apps in the google play store. It is necessary to handle class imbalance to avoid bias and improve the model's accuracy. Several methods for handling class imbalance exist, such as undersampling, oversampling, and ensemble classifier (Chao Chen & Breiman, 2004; James et al., 2014). Undersampling involves randomly deleting records from the majority class. Oversampling involved generating synthetic records using sample attributes from minority classes. The prevalent method used for oversampling is SMOTE, which stands for Synthetic Minority Over-sampling Technique. This method examines the feature space of the data points belonging to the minority class and identifies its k nearest neighbors in basic terms. For this project, we have used the SMOTE method for sampling.



```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state=42,stratify=Y)

sm = SMOTE(random_state=42)
X_train_smote, y_train_smote = sm.fit_resample(X_train, y_train)
y_train_smote.value_counts().plot(kind='bar')
```

```
<AxesSubplot:>
```

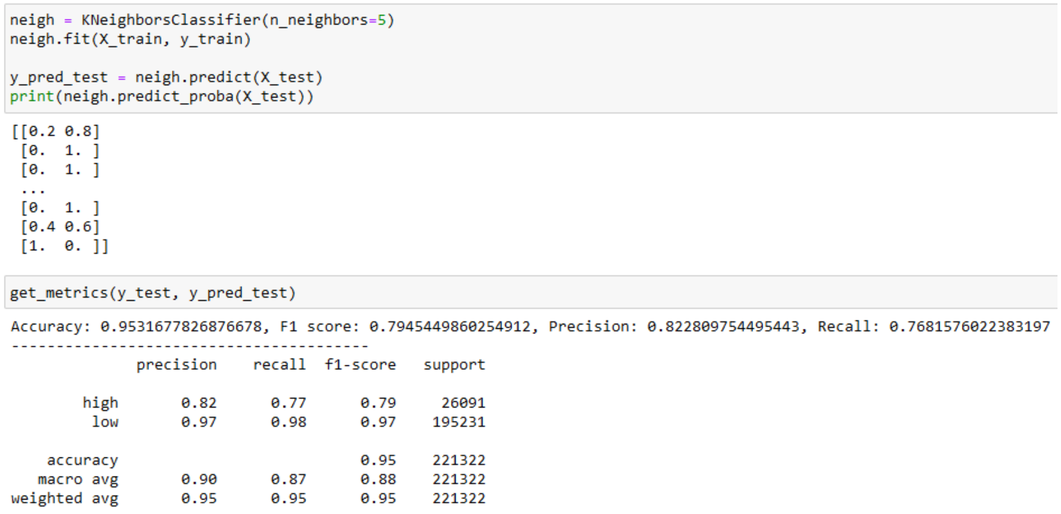
Figure 21. Class distribution after smote

The dataset was divided into training and testing sets using a 70:30 split and stratified k-fold option. Due to this option, each set contains the same percentage of records for the target class as the complete dataset. The Smote algorithm was applied only to the training set. The class distribution greatly improved after applying SMOTE. Approximately 350k synthetic records were added for the high class to balance the dataset.

The following section evaluates the performance of three machine learning models, k-nearest neighbors(kNN), support vector machine(SVM), and Random forest model, on this dataset and after balancing the training dataset using SMOTE. The models used evaluation metrics such as F1-score, confusion matrix, precision and recall.

3.5.1 Building kNN model

The k-Nearest Neighbor algorithm is a highly effective classification method that can handle intricate data and is simple to implement. Its functioning involves identifying the K nearest points in the training set, using their class to predict the classification or value of a fresh data point. It is crucial to select the k-value sensibly to prevent either overfitting or underfitting the model, typically by opting for an odd number as the k-value.



```
neigh = KNeighborsClassifier(n_neighbors=5)
neigh.fit(X_train, y_train)

y_pred_test = neigh.predict(X_test)
print(neigh.predict_proba(X_test))
```

```
[[0.2 0.8]
 [0.  1. ]
 [0.  1. ]
 ...
 [0.  1. ]
 [0.4 0.6]
 [1.  0. ]]
```

```
get_metrics(y_test, y_pred_test)
```

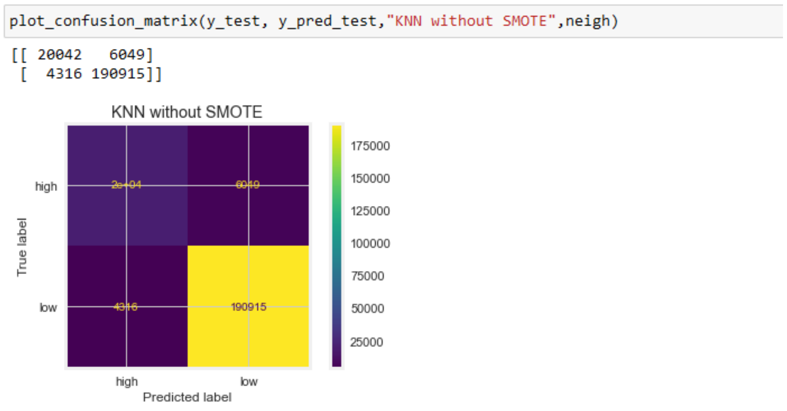
```
Accuracy: 0.9531677826876678, F1 score: 0.7945449860254912, Precision: 0.822809754495443, Recall: 0.7681576022383197
----------------------------------------
              precision    recall  f1-score   support

        high       0.82      0.77      0.79     26091
         low       0.97      0.98      0.97    195231

    accuracy                           0.95    221322
   macro avg       0.90      0.87      0.88    221322
weighted avg       0.95      0.95      0.95    221322
```

Figure 22. kNN before SMOTE

Figure 22 shows the kNN model trained on imbalanced dataset. It achieved accuracy of 95.31%, F1-score of 79.45%. The precision is slightly better than recall.

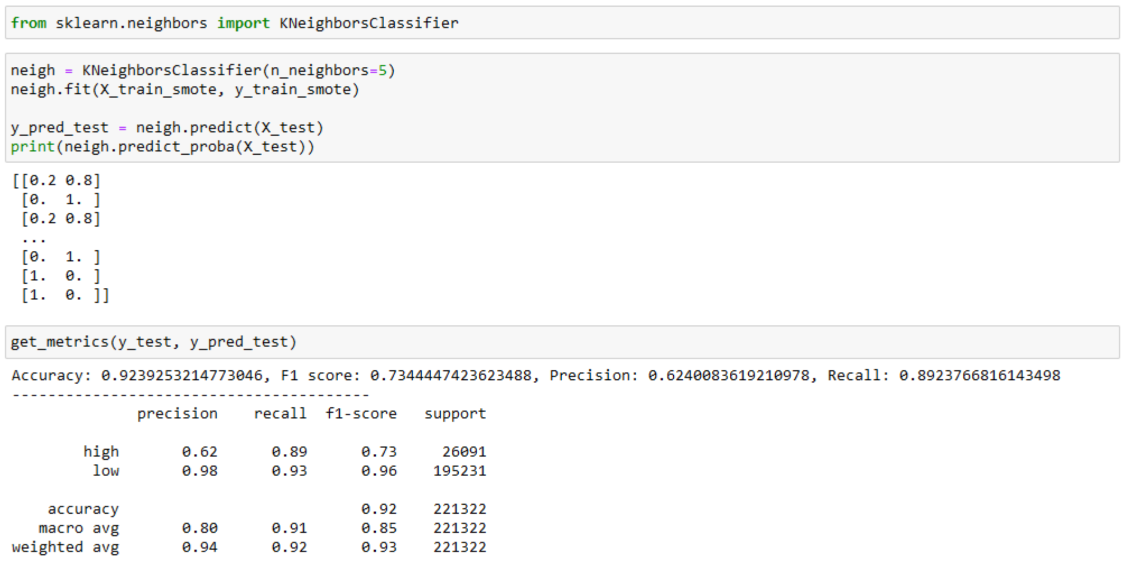


```
plot_confusion_matrix(y_test, y_pred_test,"KNN without SMOTE",neigh)
```

```
[[ 20042   6049]
 [  4316 190915]]
```

Figure 23. Confusion matrix of kNN before SMOTE

Figure 23 shows the confusion matrix of the kNN model. The Type I error is high since the number of false positives for the high class is more.



```
from sklearn.neighbors import KNeighborsClassifier
```

```
neigh = KNeighborsClassifier(n_neighbors=5)
neigh.fit(X_train_smote, y_train_smote)

y_pred_test = neigh.predict(X_test)
print(neigh.predict_proba(X_test))
```

```
[[0.2 0.8]
 [0.  1. ]
 [0.2 0.8]
 ...
 [0.  1. ]
 [1.  0. ]
 [1.  0. ]]
```

```
get_metrics(y_test, y_pred_test)
```

```
Accuracy: 0.9239253214773046, F1 score: 0.7344447423623488, Precision: 0.6240083619210978, Recall: 0.8923766816143498
----------------------------------------
              precision    recall  f1-score   support

        high       0.62      0.89      0.73     26091
         low       0.98      0.93      0.96    195231

    accuracy                           0.92    221322
   macro avg       0.80      0.91      0.85    221322
weighted avg       0.94      0.92      0.93    221322
```

Figure 24. kNN after SMOTE

Figure 24 shows the kNN model trained on the dataset after balancing it using SMOTE. It achieved accuracy of 95.31%, an F1-score of 73.73%. The recall is very good but it has poor precision.

```
plot_confusion_matrix(y_test, y_pred_test,"KNN with SMOTE",neigh)
```

```
[[ 23283   2808]
 [ 14029 181202]]
```

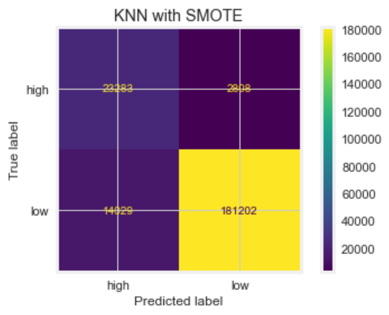
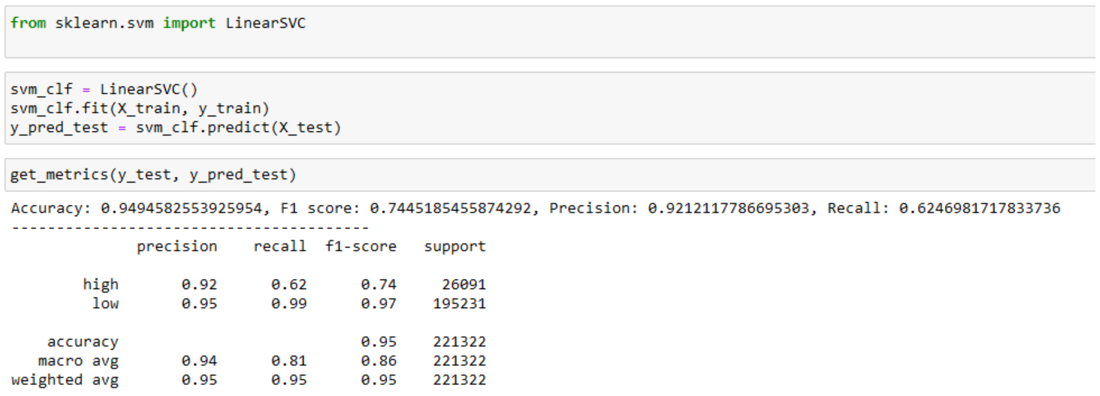


Figure 25. Confusion matrix of kNN after SMOTE

Figure 25 shows the confusion matrix of the kNN model. The Type I error is high since the number of false positives for the high class is more.

3.5.2 Building SVM model

The Support Vector Machine (SVM) is an easy-to-understand algorithm for supervised machine learning that can be used for classification and regression. SVM works by finding a hyperplane, which is essentially a boundary between different types of data. In 2D space, this hyperplane is a line. In SVM, each data item in the dataset is plotted in an N-dimensional space, where N is the number of features or attributes in the data. Then, the optimal hyperplane is found to separate the data. However, SVM is inherently limited to binary classification, meaning it can only choose between two classes. Various techniques can be used for multi-class problems to address this issue. For example, a binary classifier can be created for each class of data, and the two possible results of each classifier are whether the data point belongs to that class or not.SVM can work well with different types of datasets. It is versatile and works well for both high and low-dimensional data. However it is very computationally expensive for large datasets.



```
from sklearn.svm import LinearSVC
```

```
svm_clf = LinearSVC()
svm_clf.fit(X_train, y_train)
y_pred_test = svm_clf.predict(X_test)
```

```
get_metrics(y_test, y_pred_test)
```

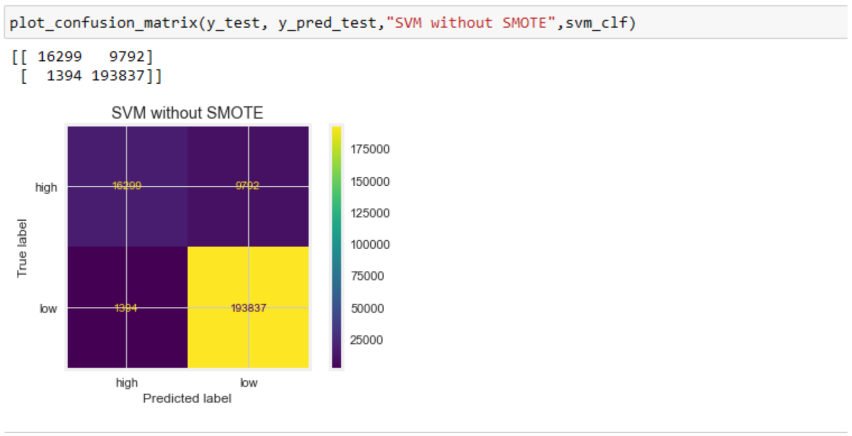
```
Accuracy: 0.9494582553925954, F1 score: 0.7445185455874292, Precision: 0.9212117786695303, Recall: 0.6246981717833736
----------------------------------------
              precision    recall  f1-score   support

        high       0.92      0.62      0.74     26091
         low       0.95      0.99      0.97    195231

    accuracy                           0.95    221322
   macro avg       0.94      0.81      0.86    221322
weighted avg       0.95      0.95      0.95    221322
```

Figure 26. SVM before SMOTE

Figure 26 shows the SVM model trained on the imbalanced dataset. It achieved an accuracy of 94.94%, an F1-score of 74.45%. The precision is much better than recall.

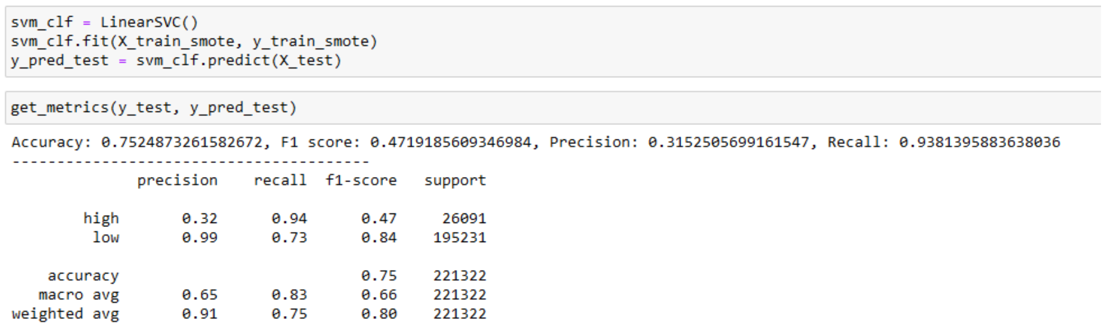


```
plot_confusion_matrix(y_test, y_pred_test,"SVM without SMOTE",svm_clf)
```

```
[[ 16299   9792]
 [  1394 193837]]
```

Figure 27. Confusion matrix of SVM before SMOTE

Figure 27 shows the confusion matrix of the SVM model. The Type I error is high since the number of false positives for the high class is more.



```
svm_clf = LinearSVC()
svm_clf.fit(X_train_smote, y_train_smote)
y_pred_test = svm_clf.predict(X_test)
```

```
get_metrics(y_test, y_pred_test)
```

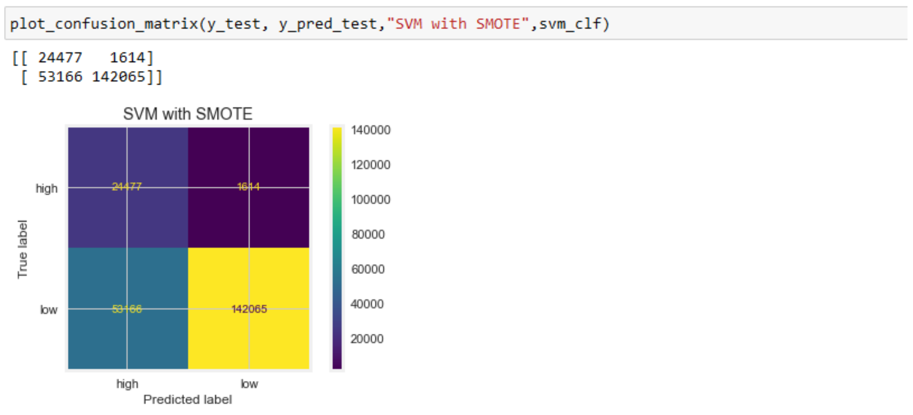
```
Accuracy: 0.7524873261582672, F1 score: 0.4719185609346984, Precision: 0.3152505699161547, Recall: 0.9381395883638036
----------------------------------------
              precision    recall  f1-score   support

        high       0.32      0.94      0.47     26091
         low       0.99      0.73      0.84    195231

    accuracy                           0.75    221322
   macro avg       0.65      0.83      0.66    221322
weighted avg       0.91      0.75      0.80    221322
```

Figure 28. SVM after SMOTE

Figure 28 shows the SVM model trained on the imbalanced dataset. It achieved an accuracy of 75.24%,an F1-score of 47.19%. The recall is much better than precision.



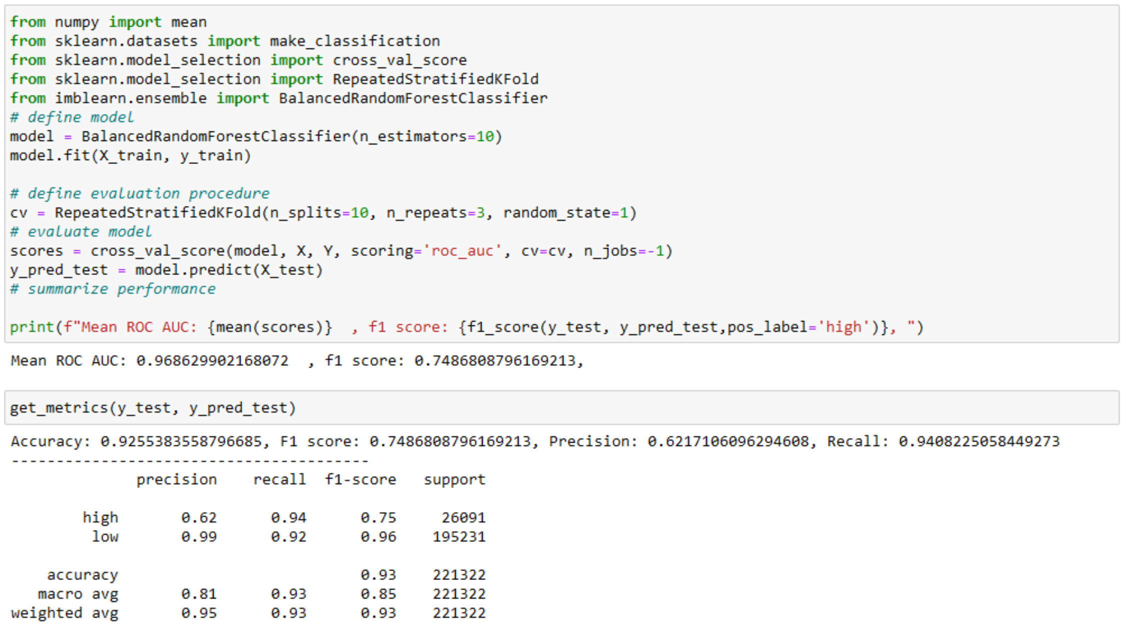
```
plot_confusion_matrix(y_test, y_pred_test,"SVM with SMOTE",svm_clf)
```

```
[[ 24477   1614]
 [ 53166 142065]]
```

Figure 29. Confusion matrix of SVM after SMOTE

Figure 29 shows the confusion matrix of the SVM model. The Type II error is high since the number of false negatives for the low class is more.

3.5.3 Building random forest model



```
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from imblearn.ensemble import BalancedRandomForestClassifier
# define model
model = BalancedRandomForestClassifier(n_estimators=10)
model.fit(X_train, y_train)

# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X, Y, scoring='roc_auc', cv=cv, n_jobs=-1)
y_pred_test = model.predict(X_test)
# summarize performance

print(f"Mean ROC AUC: {mean(scores)}  , f1 score: {f1_score(y_test, y_pred_test,pos_label='high')}, ")
```

```
Mean ROC AUC: 0.968629902168072  , f1 score: 0.7486808796169213,
```

```
get_metrics(y_test, y_pred_test)
```

```
Accuracy: 0.9255383558796685, F1 score: 0.7486808796169213, Precision: 0.6217106096294608, Recall: 0.9408225058449273
----------------------------------------
              precision    recall  f1-score   support

        high       0.62      0.94      0.75     26091
         low       0.99      0.92      0.96    195231

    accuracy                           0.93    221322
   macro avg       0.81      0.93      0.85    221322
weighted avg       0.95      0.93      0.93    221322
```

Figure 30. Balanced Random forest model

Balanced random forest classifier randomly undersamples the records in the majority class and oversamples records from the minority class to balance the data. Figure 30 shows the balanced random forest model built using the google play store dataset. This model performs well in classifying records from the low class but doesn't perform well in classifying records from the high class. Although the accuracy is high for this model, almost 92.5%, the F1- score is 74.9%.

```
plot_important_features(X_train,list(X_train.columns),model.feature_importances_)
```

```
[('Rating', 0.17789574276579528), ('Rating Count', 0.6640726628316714), ('Free', 0.0011966902160452189), ('Price', 0.00245
93000448852676), ('Size', 0.06602519229291716), ('In App Purchases', 0.021688818656881735), ('Editors Choice', 0.000237921
587797433), ('Released_day_of_week', 0.018585454379599516), ('Adults only 18+', 4.809232205585827e-06), ('Everyone', 0.002
332039430160266), ('Everyone 10+', 0.0009228431732704085), ('Mature 17+', 0.0008683224099385596), ('Unrated', 1.7177597640
195533e-05), ('num_images', 0.04369302538119208)]
```

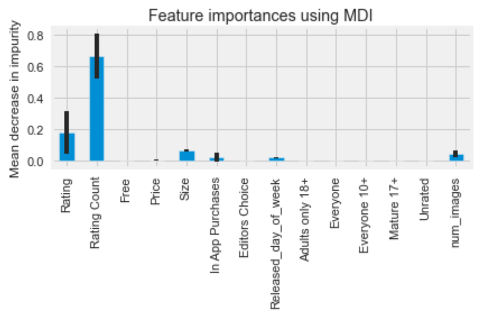


Figure 31. Important features given by Balanced Random forest model

.

```
plot_confusion_matrix(y_test, y_pred_test,model_name="Balanced random forrest without SMOTE")
```

```
[[ 24547   1544]
 [ 14936 180295]]
```

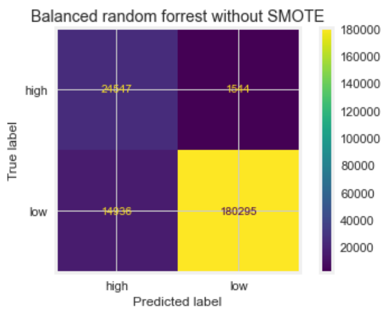
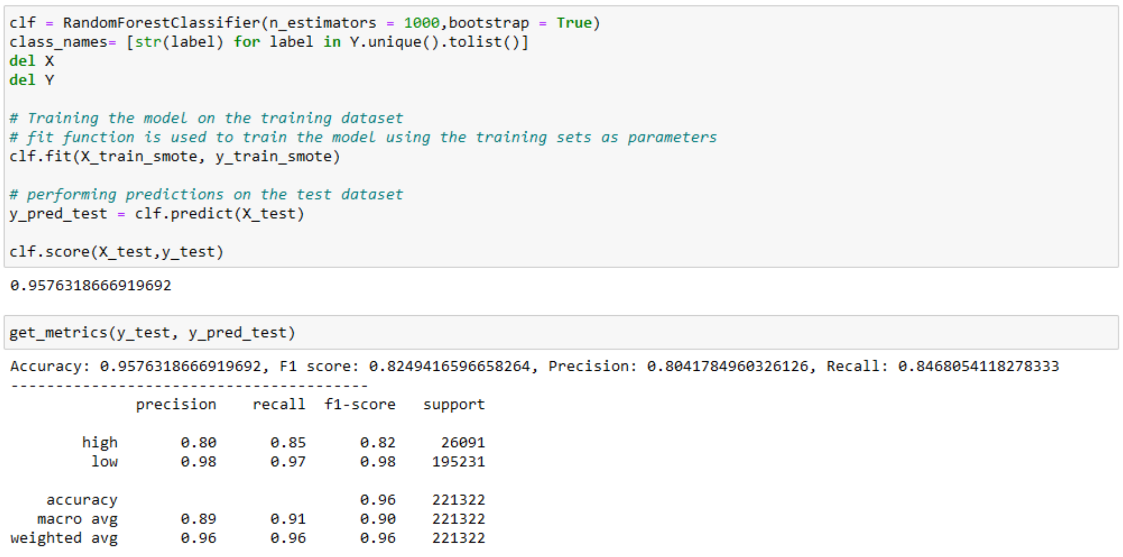


Figure 32. Confusion matrix of Random forest model before applying SMOTE

The balanced random forest model ranked the important features based on Gini(mean decrease impurity) after plotting the feature importance for this dataset. We observed that Rating, Rating Count, Size, No of images, In-App purchases, and released day-of-week features were found to be important. Figure 32 shows the confusion matrix for the predictions made by the model. The Type I error is high since the number of false positives for the high class is more.



```
clf = RandomForestClassifier(n_estimators = 1000,bootstrap = True)
class_names= [str(label) for label in Y.unique().tolist()]
del X
del Y

# Training the model on the training dataset
# fit function is used to train the model using the training sets as parameters
clf.fit(X_train_smote, y_train_smote)

# performing predictions on the test dataset
y_pred_test = clf.predict(X_test)

clf.score(X_test,y_test)
```

```
0.9576318666919692
```

```
get_metrics(y_test, y_pred_test)
```

```
Accuracy: 0.9576318666919692, F1 score: 0.8249416596658264, Precision: 0.8041784960326126, Recall: 0.8468054118278333
----------------------------------------
              precision    recall  f1-score   support

        high       0.80      0.85      0.82     26091
         low       0.98      0.97      0.98    195231

    accuracy                           0.96    221322
   macro avg       0.89      0.91      0.90    221322
weighted avg       0.96      0.96      0.96    221322
```

Figure 33. Random forest model after applying SMOTE

The random forest classifier performed better than the Balanced random forest classifier. The accuracy is 95.8%, and F1-score is 82.6%. The random forest model ranked the critical features based on Gini(mean decrease impurity) after plotting the feature importance for this dataset. Rating, Rating Count, Size, No of images, Everyone, and released day-of-week features were found to be important. Figure 35 shows the confusion matrix for the predictions made by the model. The Type I error is lower than, but the Type II error is higher than in the previous model.

```
plot_important_features(X_train,list(X_train.columns),clf)
```

```
[('Rating', 0.20120491122650508), ('Rating Count', 0.6038243347916546), ('Free', 0.004531712213285925), ('Price', 0.001521
0775649797065), ('Size', 0.034499326259671385), ('In App Purchases', 0.0030895756652571204), ('Editors Choice', 0.00011065
731493397744), ('Released_day_of_week', 0.016053834058373843), ('Adults only 18+', 5.256390254922998e-06), ('Everyone', 0.
038326254934261395), ('Everyone 10+', 0.0033180396466159544), ('Mature 17+', 0.005259503887350514), ('Unrated', 1.12413834
26332252e-05), ('num_images', 0.08824427466342923)]
```

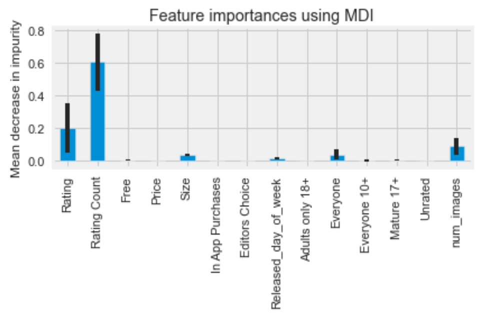


Figure 34. Important features given by Random forest model after applying SMOTE

```
plot_confusion_matrix(y_test, y_pred_test,"Random forrest with SMOTE",clf)
```

```
[[ 21959   4132]
 [  5146 190085]]
```

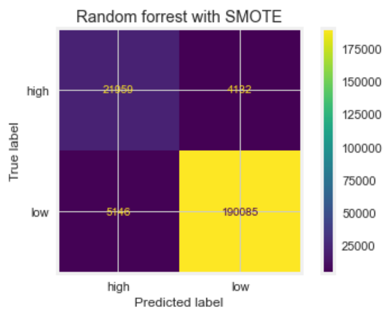


Figure 35. Confusion matrix of Random forest model after applying SMOTE

3.6 Sentimental analysis of App reviews

In this step, the sentimental analysis of app reviews was performed. Around 8769559 app reviews were extracted from Google Play Store, containing several fields such as App id, review content, review date, user id of the reviewer, score(rating) given by the user, and thumbs up count(number of users who found the review helpful). For this project, the review content and score were used.

3.6.1 Data preprocessing

For this stage, 24000 reviews were selected for analysis from the complete dataset. The reviews were for apps from different categories. The first step of data cleaning was to remove the stopwords and punctuation marks from the sentences using apply function. The second step of the process was to extract individual words by applying the nltk word_tokenize function to the sentences. Words are like atoms of natural language. Each word contains some vital information in it to describe objects. Tokenization is essential since it allows us to analyze the dataset's word frequency and frequent phrases. The third step involved lemmatizing the tokens. Lemmatizing reduces words to their core meaning. E.g., words such as player, playing, and plays become play. The last step of the data cleaning process involved parts of speech tagging. Each token word from the content was tagged as a Noun, pronoun, verb, adverb, adjective, preposition, conjunction, or interjection.

3.6.2 Sentimental analysis

The sentimental analysis of reviews was performed using the nltk library's SentimentIntensityAnalyzer and flair library's TextClassifier. Both of these are pre-trained classifiers. SentimentIntensityAnalyzer performed well on the dataset by correctly classifying 21300 reviews out of 24000. It identified the polarity of reviews and assigned a positive, negative, neutral, and compound score depending on the overall sentiment of the words in the review. If the compound score was more significant than 0.5, then the review was classified as positive if the score was less than -0.5, then it was labeled as Negative; and if the score was between -0.5 and 0.5, then the review was classified as Neutral. Flair TextClassifier classified 18345 reviews correctly. It assigned a score and Positive, Negative, and Neutral label to each review. The next step after classifying the reviews was to identify the words/ features associated with positive and negative reviews and find the frequency of these words in the reviews.

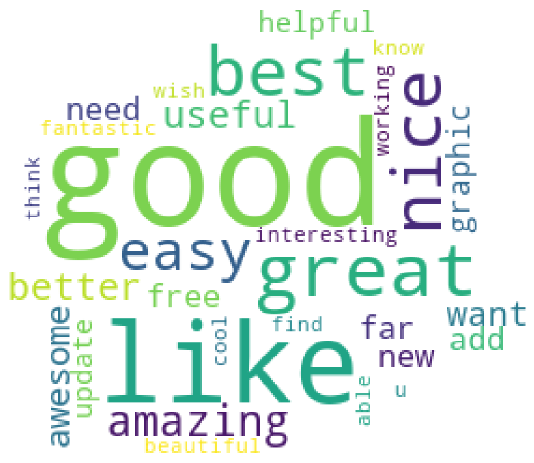
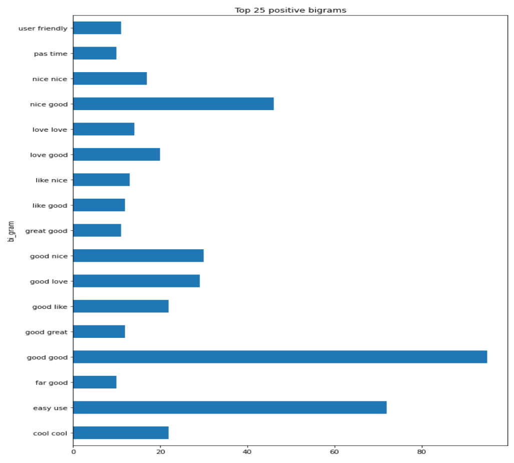


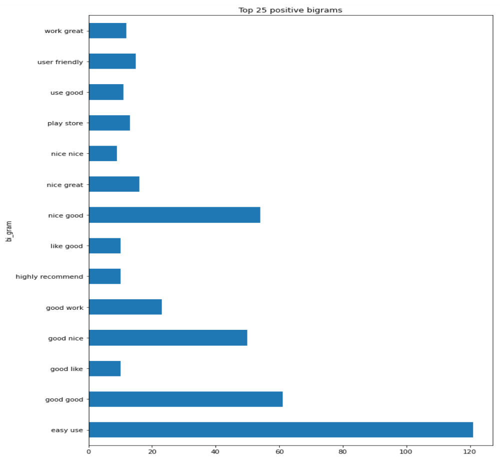
Figure 36. Positive sentiments in reviews

Figure 31 shows frequently occurring words in positive reviews, i.e. reviews with a score greater than three stars. The positive reviews consist of compliments for the app. Some of frequently occurring words are "good" ,"like","best","easy","nice","free","graphic","amazing","helpful", and "useful". These words indicate that users like the app's graphics, and cost and find the app helpful.

TF-IDF or Term Frequency – Inverse document frequency is essential in text analysis. It helps find Bi grams and Trigrams in text. Unigrams help us to find which words frequently occur in the text, but they do not convey enough information as N-grams. Bigrams and Trigrams help us to visualize which groups of words are frequently found together in the text. Analyzing Bigrams and Trigrams would help developers to understand the issues faced and features liked by users.

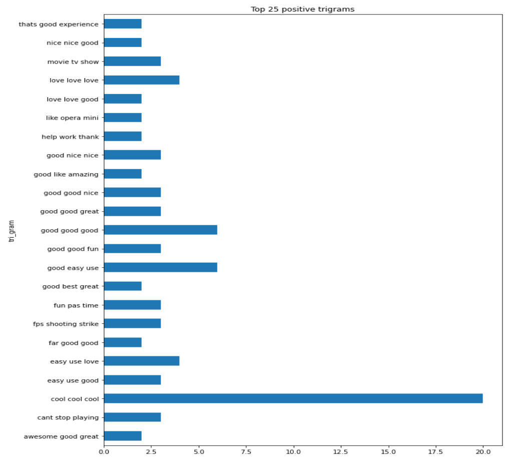


(a) popular apps

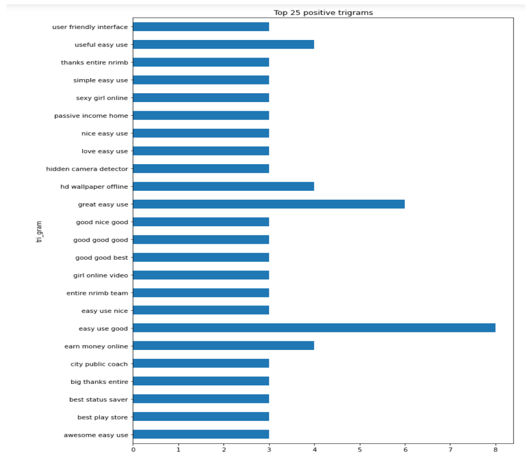


(b) less popular apps

Figure 37. Positive bigrams



(a) popular apps



(b) less popular apps

Figure 38. Positive trigrams

Figure 32 depicts the top 25 positive Bi grams and trigrams in popular and less popular apps. The positive bi grams and trigrams indicate that users find the apps excellent and user-friendly. There is not much difference in the positive reviews of popular and less popular apps.

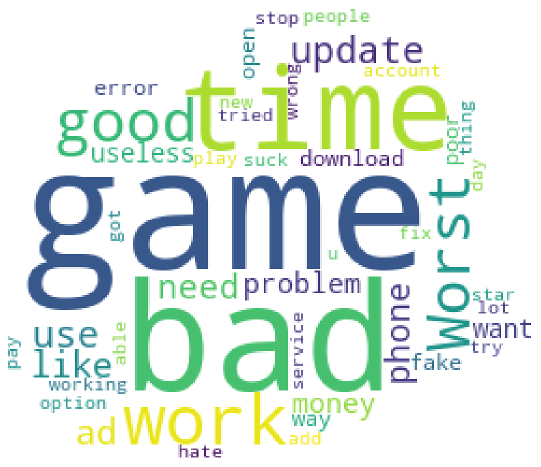
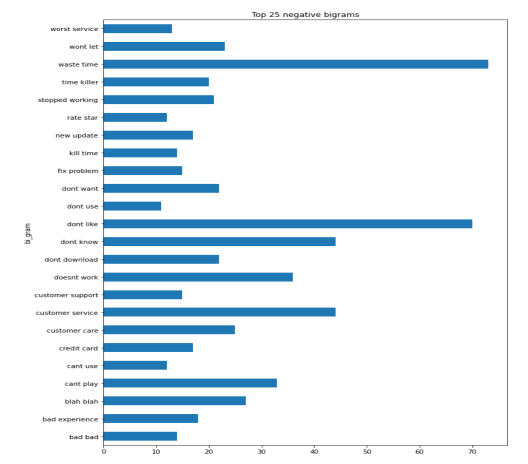
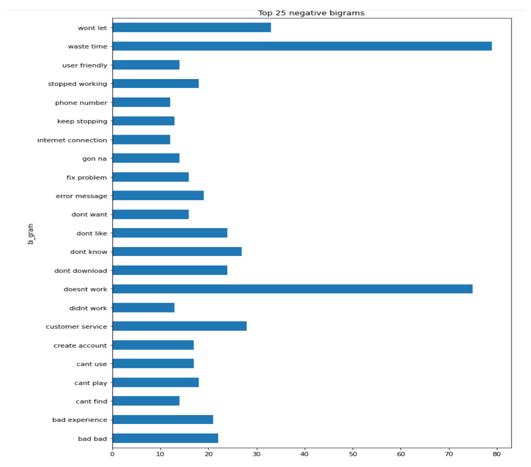


Figure 39. Negative sentiments in reviews

Figure 33 shows frequently occurring words in negative reviews, i.e. reviews with a score of fewer than three stars. The negative reviews consist of user feedback and complaints about the app. Some of the frequently occurring words are ”game”,”bad”,”time” ,”worst”,”update”,”error”,”problem”, and ”fake”. These words indicate the issues users face and their overall experience with the app.

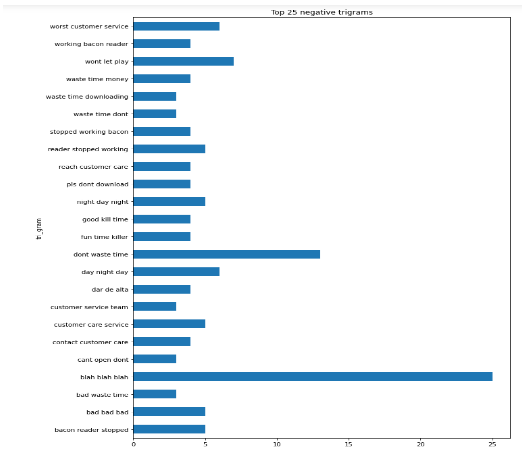


(a) negative bigrams in popular apps

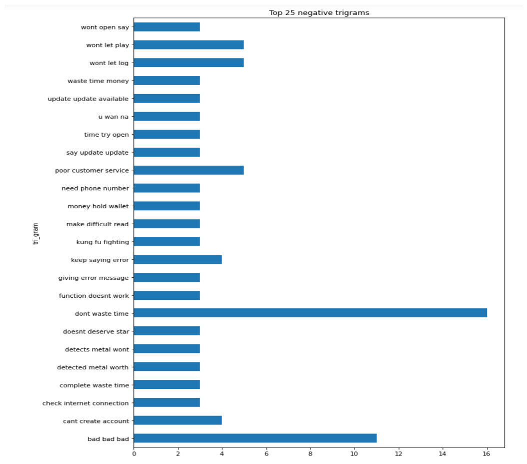


(b) negative bigrams in less popular apps

Figure 40. Negative bigrams



(a) popular apps



(b) less popular apps

Figure 41. Negative trigrams

Figures 34 and 34 depict the top 25 negative bigrams and trigrams in popular and less popular apps. The negative bigrams and trigrams in popular apps indicate that users face poor customer service and some functionality issues. The negative bigrams and trigrams in less popular apps indicate that users face issues with error messages while using apps, app updates, and poor customer service.

3.7 Dashboard for visualization

The Tableau dashboard will be a high-level view of the cleaned and transformed google play store data. The charts shown in the dashboard will help users to view which categories are becoming popular. It will also help to view how popular apps change their size and price over time. The dashboard user can be anyone who wants to deploy apps on Google Play Store. This dashboard can help developers to compare how their app compares to other apps in the Play Store and correlate the different attributes to the number of installs.

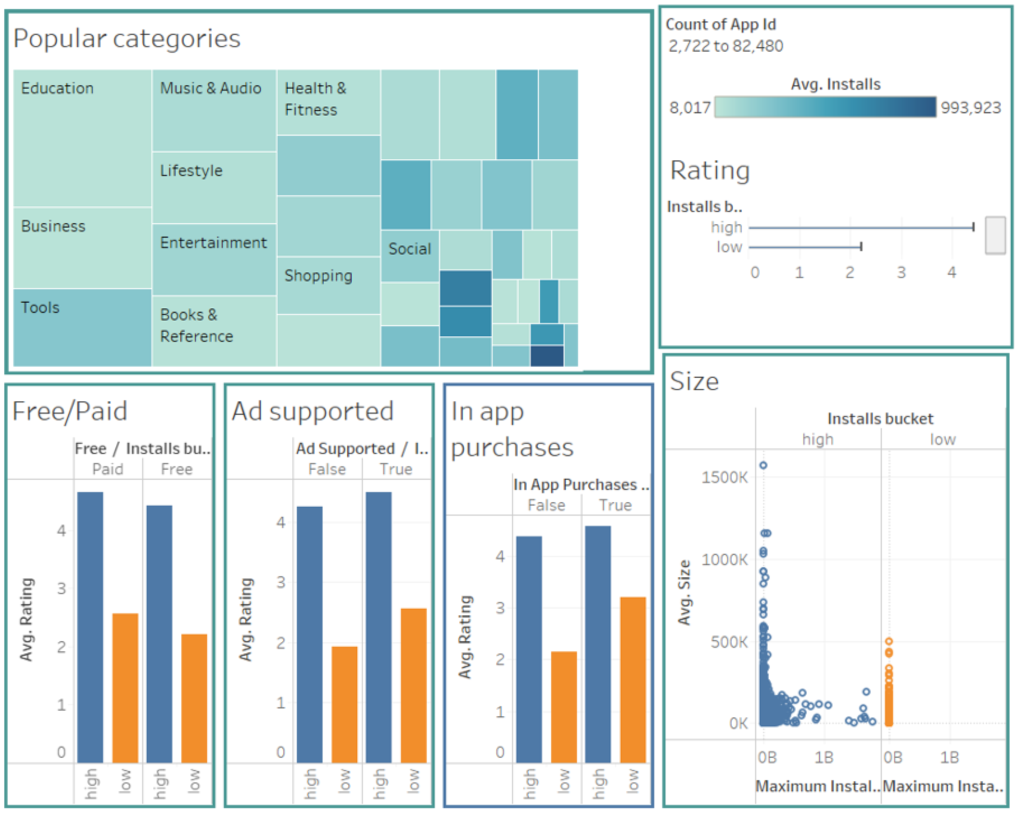


Figure 42. Tableau dashboard

4 Results

The exploratory data analysis done in the previous section helped to understand the relationships between different features in Google. The data analysis was done to find important features to train the random forest model on the google play store dataset, perform sentimental analysis and build a dashboard. The model and sentimental analyzer were built using Python. The exploratory data analysis helped to understand the effect of different attributes of play store app advertisement such, App name, Content rating, In-App Purchases, Ad-supported, No of promotional images and videos, Size, Price, Rating, No of reviews and ratings have on App Installations. Many popular classifier models were trained on the dataset, such as kNN, SVM, random forest model, and balanced forest model, using raw and SMOTE data. The following table summarizes the evaluation of the classifier models trained on the google play store dataset:

Table 2. Summary of Classifier model results

Model Name	Accuracy	F1 score	Precision	Recall
SVM without SMOTE	94.94%	74.45%	92.12%	62.46%
SVM with SMOTE	75.24%	47.19%	35.52%	93.81%
kNN classifier without SMOTE	95.31%	79.45%	82.28%	76.81%
kNN classifier with SMOTE	92.41%	73.73%	62.30%	90.29%
Balanced Random Forest without SMOTE	92.55%	74.86%	62.17%	94.08%
Random Forest with SMOTE	95.76%	82.49%	80.41%	84.68%

The Random forest model with SMOTE-transformed data and bootstrapping performs better than SVM and kNN models after a lot of tuning using different numbers of estimators.

The sentimental analysis of reviews was done to understand users' experience with the apps. Nltk library's SentimentIntensityAnalyzer and flair library's TextClassifier were used for sentimental analysis. The TextClassifier performed better than SentimentIntensityAnalyzer. It was observed that popular apps received higher ratings and compliments from users compared to other apps. The text analysis done during sentimental analysis helped to understand the types of issues users face. Users who installed popular apps frequently faced issues with customer service and fewer issues with functionality. Users who installed less popular apps faced more issues with error messages while using different app features. After sentimental analysis, a Tableau dashboard was built using the cleaned and transformed Google Play Store data. This dashboard can be used by any user who wants to deploy an app on the google play store to understand what sizing and pricing strategy needs to be used for the apps.

5 Conclusion

The objective of this project was to find important features of a google play store app that influence users to install the app. Based on the exploratory data analysis, it was observed that Ratings, Rating count, Size, and In-app purchases were important attributes that affected app installations. The category of apps also influenced app installations. Gaming apps, educational, music, and tools apps are popular among users. The sentimental analysis revealed what users liked about the apps and what type of issues they faced.

5.1 Limitations

As with most studies, this project is subject to the following limitations because of the limited data available for analysis.

1. The app description couldn't be evaluated without crowdsourcing to understand the quality and structure of the description of popular apps
2. The app's quality of promotional images and videos couldn't be evaluated.
3. The dataset only contained data about the app at a certain time. The study could not evaluate how the apps performed over time after being published on Google Play Store.
4. The sentimental analysis could only be performed on reviews in English. Some reviews were in regional languages, such as Spanish, so they could not be evaluated correctly.

5.2 Future Work

1. Evaluate the effect of permissions of apps on installations and ratings.
2. Collect data about the privacy policy of popular apps and compare them with other apps.
3. study the effect of app updates on installations.
4. Evaluate how other classifier algorithms perform on this dataset.

References

Agrawal, S. C., Singh, S., & Gupta, S. (2021). Evaluation of machine learning techniques in sentimental analysis. *2021 5th International Conference on Information Systems and Computer Networks (ISCON)*, 1–5. https://doi.org/10.1109/ISCON52037.2021.9702430

Carbunar, B., & Potharaju, R. (2015). A longitudinal study of the google app market. *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM '15.*

Carreño, L. V. G., & Winbladh, K. (2013). Analysis of user comments: An approach for software requirements evolution. *2013 35th International Conference on Software Engineering (ICSE)*, 582–591. https://doi.org/10.1109/ICSE.2013.6606604

Chao Chen, A. L., & Breiman, L. (2004). Using random forest to learn imbalanced data. (666). https://statistics.berkeley.edu/tech-reports/666

Google. (2021). Support.google.com. get discovered on google play search - play console help. [Retrieved May 11, 2022, from www.support.google.com website:]. https://support.google.com/googleplay/android-developer/answer/4448378?hl=en

Henze, N., & Boll, S. (2011). Release your app on sunday eve: Finding the best time to deploy apps. *13th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '11)*, 581–586.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2014). *An introduction to statistical learning: With applications in r*. Springer Publishing Company, Incorporated.

Jiang, H., Ma, H., Ren, Z., Zhang, J., & Li, X. (2015). What makes a good app description? *6th Asia-Pacific Symposium on Internetware (INTERNETWARE 2014)*, 45–53.

Jin, J., & Liu, Y. (2010). How to interpret the helpfulness of online product reviews: Bridging the needs between customers and designers. *the 2nd international workshop on Search and mining user generated contents (SMUC '10)*, 87–94.

Makaba, T., & Dogo, E. (2019). A comparison of strategies for missing values in data on machine learning classification algorithms. *2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*, 1–7. https://doi.org/10.1109/IMITEC45504.2019.9015889

MarketWatch. (2022). Mobile enterprise application market growth 2022-2029: In-depth industry analysis on size, share, cost structure, prominent key players strategies, global demand, emerging trends, recent developments, future investments and regional forecast [Retrieved May 11, 2022, from www.marketwatch.com website:]. https://www.marketwatch.com/press-release/mobile-enterprise-application-market-growth-2022-2029-in-depth-industry-analysis-on-size-share-cost-structure-prominent-key-players-strategies-global-demand-emerging-trends-recent-developments-future-investments-and-regional-forecast-2022-05-04

Play, G. (2022). Meta ads manager. google play [Retrieved May 11, 2022, from www.support.google.com website:]. https://play.google.com/store/apps/details?id=com.facebook.adsmanager&%20hl=en_IN&gl=USt.google.com/google-ads/answer/6247380?hl=en

Prakash, G. (2021). Google play store apps [Retrieved May 11, 2021, from www.kaggle.com website:]. https://www.kaggle.com/datasets/gauthamp10/google-playstore-apps

Reddy, P. B. P., & Nallabolu, R. (2020). Machine learning based descriptive statistical analysis on google play store mobile applications. *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA),647-655*.

Saad, S. M., & Nanath, K. (2020). Investing in applications based on predictive modeling. *2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA)*, *1*(1), 105–110. https://doi.org/http://doi.acm.org/10.1109/DATABIA50434.2020.9190301

Taylor, V. F., & Martinovic, I. (2017). To update or not to update: Insights from a two-year study of android app evolution. *2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '17)*, 45–57.

van Buuren, S., & Groothuis-Oudshoorn, K. (2011). Mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, *45*(3), 1–67. https://doi.org/10.18637/jss.v045.i03

Ware, C. (2012). *Information visualization: Perception for design*. Elsevier.

Wei, X., Gomez, L., Neamtiu, I., & Faloutsos, M. (2012). Permission evolution in the android ecosystem. *28th Annual Computer Security Applications Conference*.

Zhai, Kristensson, S., Gong, P., Greiner, P., Peng, M., Liu, S., & Dunnigan, L. (2009). Shapewriter on the iphone: From the laboratory to the real world. *Proc. CHI, 2009*.