## ABSTRACT

The growth of the mobile application market has led to the increasing popularity of app stores such as Google Play Store. It has become a central hub for mobile application distribution and has become a crucial part of the mobile ecosystem. In this study, we analyzed the Google Play Store applications to understand the trends and patterns in the mobile application market. Our analysis focused on several key factors, including the categories of apps, app ratings, app reviews, and app size. Our results showed that the majority of the applications on the Google Play Store belong to the "Games" and "Tools" categories, and the average rating for apps was 4.3 stars. Additionally, we found that the logarithm distribution of installs is concentrated around 10^10 – 10^15, and the majority of the app reviews were positive. Our findings provide valuable insights into the current state of the mobile application market and can help developers and stakeholders make informed decisions. Along with exploratory data analysis, machine-learning algorithms such as Random Forest Regressor and K-Means Clustering will prove to be a worthwhile addition to research regarding Google Play Store applications.


## 1. Introduction

The growth of the mobile application industry has led to an enormous amount of data generation. To gain insights into the patterns and trends in this data, it is crucial to analyze the available information through various analytical techniques. This research paper aims to analyze the Google Play Store applications using Random Forest Regressor and K-Means clustering for forecasting and segmentation, respectively. The paper starts with an exploratory data analysis to understand the features and characteristics of the data. The analysis is followed by the application of two machine learning algorithms - Random Forest Regressor and K-Means clustering - to forecast and segment the Google Play Store applications. The results obtained from these algorithms are then evaluated and compared to draw meaningful conclusions. The study provides insights into the application of machine learning techniques for analyzing mobile application data and the usefulness of the same for businesses and developers.

### 1.1 Problem Statement

The mobile app market is highly competitive, and there is a constant need for app developers to understand user behavior, market trends, and the performance of their app in order to remain relevant and successful. However, there is a lack of tools and methods for effectively analyzing the vast amounts of data generated by Google Play Store Applications, making it difficult for app developers to make informed decisions and improve their app's performance. While there are numerous tools available for analysis of App Store applications and App Store has an in-built mechanism to analyze and forecast predictions, Google Play Store suffers from the problem of analysis deficit.

## 1.2 Problems Addressed

Market Research: Analysis of Google Play Store data can provide valuable insights into the mobile app market, including the most popular app categories, top-performing apps, and the latest trends.

App Optimization: Data analysis can help app developers optimize their app by identifying performance bottlenecks, improving the user experience, and increasing user engagement and retention.

App Monetization: Analysis of Google Play Store data can help app developers understand the most effective monetization strategies, including in-app purchases, advertising, and subscriptions, and determine the optimal pricing for their app.

Identifying Local App Trends: The analysis of Google Play Store data can provide insights into local app trends and help app developers understand what types of apps are popular in a specific region.

Supporting Local Businesses: The analysis of Google Play Store data can help local businesses identify opportunities for growth and expand their reach by developing apps that cater to local needs and preferences.

Supporting Local Entrepreneurship: The analysis of Google Play Store data can help support local entrepreneurship by providing valuable insights into local app trends and identifying opportunities for app development that can help entrepreneurs succeed in a specific region.

# 2. Literature Survey

## 2.1 Survey of the Existing Models/Works

**[1] P. B. Prakash Reddy and R. Nallabolu, "Machine learning based Descriptive Statistical Analysis on Google Play Store Mobile Applications," 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2020, pp. 647-655, Doi: 10.1109/ICIRCA48905.2020.9183271.**

The paper is about using machine learning algorithms to analyze user ratings and reviews of mobile applications in the Google Play Store. The objective of the paper is to identify the relationship between user ratings and reviews, and how they can be used to predict user satisfaction.

The study uses a dataset of 1500 mobile applications from the Google Play Store, which were selected based on their popularity and the number of downloads. The dataset consists of information such as the application name, category, rating, number of reviews, and user reviews. The authors use various machine learning algorithms, including Naive Bayes, Support Vector Machines (SVM), and Decision Trees, to classify user reviews as positive or negative based on the text content of the review.

The authors also use descriptive statistical analysis to explore the relationship between user ratings and reviews. They analyze the correlation between the two variables and find that there is a positive correlation between user ratings and reviews. The study also identifies important features of mobile applications that are associated with user satisfaction, such as application stability, ease of use, and functionality.

The results of the study show that machine learning algorithms can effectively analyze and classify user reviews, with SVM and Decision Trees outperforming Naive Bayes in terms of classification accuracy. The study also shows that user ratings and reviews can be used to predict user satisfaction, with the Decision Tree algorithm performing the best in predicting user satisfaction.

Overall, the study demonstrates the potential of machine learning algorithms in analyzing user ratings and reviews of mobile applications and identifying important features associated with user satisfaction. The authors suggest that this type of analysis can be used by developers to improve their applications and enhance user experience.

**[2] R. M. Amir Latif, M. Talha Abdullah, S. U. Aslam Shah, M. Farhan, F. Ijaz and A. Karim, "Data Scraping from Google Play Store and Visualization of its Content for Analytics," 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), Sukkur, Pakistan, 2019, pp. 1-8, Doi: 10.1109/ICOMET.2019.8673523.**

The paper is about the use of data scraping and visualization techniques for analyzing mobile application data from the Google Play Store. The authors explain the process of data scraping, which involves the collection of data from the Google Play Store through web scraping tools. They also describe various visualization techniques that can be used to analyze the collected data, such as bar charts, word clouds, and network graphs.

The study demonstrates the potential of data scraping and visualization techniques for identifying trends and patterns in mobile application data. For instance, the authors analyze the data to determine the popularity of different categories of mobile applications, such as entertainment and social media, and the ratings and reviews of individual applications. The paper also explores how data scraping and visualization techniques can provide insights for developers and users of mobile applications.

Overall, the paper emphasizes the importance of data scraping and visualization techniques for analyzing mobile application data and providing valuable insights for stakeholders in the mobile application industry. The authors conclude that these techniques have the potential to improve decision-making processes and enhance the overall user experience of mobile applications.

**[3] Venkatakrishnan, S., Kaushik, A., Verma, J.K. (2020). Sentiment Analysis on Google Play Store Data Using Deep Learning. In: Johri, P., Verma, J., Paul, S. (eds) Applications of Machine Learning. Algorithms for Intelligent Systems. Springer, Singapore. https://doi.org/10.1007/978-981-15-3357-0_2**

The paper discusses the application of deep learning techniques for sentiment analysis on mobile application reviews from the Google Play Store. The authors begin by providing an overview of sentiment analysis, which involves the use of machine learning algorithms to classify text as positive, negative, or neutral.

The study focuses on the application of deep learning models, specifically convolutional neural networks (CNN) and long short-term memory (LSTM) networks, for sentiment analysis on mobile application reviews. The authors compare the performance of these models and evaluate their accuracy in predicting sentiment.

The results demonstrate that the LSTM model outperforms the CNN model in terms of accuracy for sentiment analysis on mobile application reviews. The authors also discuss the limitations of the study, such as the need for a more diverse dataset and the potential biases in user reviews.

Overall, the paper highlights the potential of deep learning techniques for sentiment analysis on mobile application reviews and provides insights into the strengths and limitations of these models. The authors suggest that the application of these techniques can be useful for developers and marketers to monitor user feedback and improve the user experience of mobile applications.

**[4] S. Shashank and B. Naidu, "Google Play Store Apps- Data Analysis and Ratings Prediction," Dec. 2020. https://mail.irjet.net/archives/V7/i12/IRJET-V7I1248.pdf**

The paper focuses on data analysis and ratings prediction of mobile applications on the Google Play Store. The authors start by discussing the growing importance of mobile applications in the digital world and the need for analyzing user feedback to improve their performance.

The study uses a dataset of 10,000 mobile applications from the Google Play Store and applies data mining techniques to extract relevant information such as app category, size, number of installs, and user ratings. The authors then use various machine learning algorithms such as linear regression, decision trees, and support vector regression to predict the user ratings of mobile applications.

The results demonstrate that the support vector regression algorithm outperforms other models with an accuracy of 84.21% in predicting user ratings. The authors also analyze the impact of different features such as app size and category on user ratings and find that certain features such as app size have a significant impact on user ratings.

The paper highlights the importance of data analysis and prediction in improving the user experience of mobile applications. The authors suggest that the application of these techniques can be useful for developers to identify areas for improvement and optimize their applications to meet user expectations. Additionally, the findings can be useful for marketers to understand user preferences and tailor their marketing strategies accordingly.

**[5] I. Journal, "IRJET- Play Store App Analysis," IRJET, Jan. 2021. Available: https://www.academia.edu/52123932/IRJET_Play_Store_App_Analysis**

The paper provides a technical analysis of mobile applications available on the Google Play Store. The study uses a dataset of 108 mobile applications and applies various statistical techniques such as descriptive statistics, correlation analysis, and regression analysis to extract relevant information such as app category, size, number of installs, and user ratings.

The results show that the most popular app categories on the Google Play Store are communication, social, and entertainment apps. Moreover, the correlation analysis reveals that the size of the application has a negative impact on the user ratings, while the number of installs has a positive impact.

The paper also uses regression analysis to identify the factors that have a significant impact on the user ratings. The results demonstrate that the app category, content rating, and number of installs are the most significant factors affecting the user ratings. The authors suggest that developers can optimize their applications by focusing on these factors to meet user expectations.

The study also discusses the limitations of the analysis, such as the small sample size and the limited scope of the study. The authors suggest that further research can be conducted using a larger sample size and a more diverse set of mobile applications to provide more insights into user preferences and behaviors.

Overall, the paper provides valuable insights into the analysis of mobile applications on the Google Play Store and highlights the importance of data-driven decision making in the development and marketing of mobile applications. The findings can be useful for developers to optimize their applications and for marketers to tailor their marketing strategies based on user preferences.

**[6] Maredia, Rimsha. (2020). Analysis of Google Play Store Data set and predict the popularity of an app on Google Play Store.**

The paper presents an analysis of the Google Play Store dataset using various techniques such as data cleaning, data preprocessing, and data visualization. The main objective of this research is to predict the popularity of an app on the Google Play Store using various attributes such as category, rating, reviews, size, and price.

The data cleaning process involves removing duplicates, missing values, and correcting the data format. The data preprocessing involves converting the categorical variables into numerical variables using one-hot encoding and feature scaling to normalize the data.

The data visualization techniques used in the paper include scatter plots, heatmaps, and bar charts to understand the correlation between different attributes of the dataset. The analysis shows that the number of installs, ratings, and reviews are positively correlated with the popularity of the app.

The paper also presents a machine learning model to predict the popularity of the app using various attributes. The models used include linear regression, decision trees, and random forests. The results show that the random forest model performs the best with an accuracy of 94.5%.

Finally, the paper concludes that the analysis of the Google Play Store dataset and the predictive model can be useful for app developers and marketers to understand the factors that influence the popularity of an app on the Google Play Store.

**[7] Bashir, G. M. M., Hossen, M. S., Karmoker, D., & Kamal, M. J. (2019, December). Android apps success prediction before uploading on google play store. In 2019 International Conference on Sustainable Technologies for Industry 4.0 (STI) (pp. 1-6). IEEE.**

The paper presents a method for predicting the success of Android apps before uploading them on the Google Play Store. The proposed approach is based on a set of features that are extracted from the app's metadata, including the app's category, size, price, and number of downloads, ratings, and reviews. These features are used to train a machine learning model, specifically a decision tree algorithm, which can predict the success of an app based on the input features.

The dataset used for training and testing the model consists of 10,000 Android apps collected from the Google Play Store. The dataset is preprocessed to remove missing data and handle categorical data through one-hot encoding. The dataset is split into 70% training and 30% testing data. The decision tree algorithm is trained on the training dataset and tested on the testing dataset, achieving an accuracy of 79.15% in predicting the success of an app.

The paper also investigates the importance of each feature in predicting the app's success. The results show that the number of downloads, followed by the number of ratings, is the

most significant feature in predicting the app's success. The app's category and price are also important features in predicting the success of an app, while the size of the app and the number of reviews have a relatively lower impact on the success of an app.

Overall, the proposed approach shows promising results in predicting the success of Android apps before uploading them to the Google Play Store, which can help developers in optimizing their app's features for better marketability and user acceptance.

**[8] MarediaGao, S., Liu, L., Liu, Y., Liu, H., & Wang, Y. (2021). API recommendation for the development of Android App features based on the knowledge mined from App stores. Science of Computer Programming, 202, 102556.**

The paper proposes a method for recommending application programming interfaces (APIs) based on the features and requirements of existing Android applications. The authors argue that using existing applications as a basis for API recommendation can save development time and reduce the likelihood of errors in the development process.

The proposed method involves three steps: feature extraction, feature mapping, and API recommendation. In the first step, the features of existing Android applications are extracted and represented as a feature vector. In the second step, a mapping function is developed to map the feature vector to a set of candidate APIs. Finally, in the third step, the candidate APIs are ranked based on their relevance to the extracted features, and the top-ranked APIs are recommended to the developer.

The authors evaluate the proposed method using a dataset of over 2,000 Android applications from the Google Play Store. The evaluation results show that the proposed method outperforms baseline methods in terms of API recommendation accuracy and coverage. The authors also conduct a case study to demonstrate the effectiveness of the proposed method in recommending APIs for the development of a new Android application.

Overall, the paper presents a promising approach for API recommendation in Android application development based on knowledge mined from existing applications in app stores. The proposed method could potentially save time and reduce errors in the development process, and could be useful for developers who are unfamiliar with the Android API or who want to quickly develop applications with specific features.

**[9] Song, G., Hu, L., & Liu, Y. (2020). Common but innovative: learning features from apps in different domains. IEEE Access, 8, 186904-186918.**

The paper proposes a novel approach for automatic feature extraction from mobile apps in different domains using a deep learning model. The authors address the issue of domain-specific feature learning and the lack of labeled data in specific domains by proposing a framework that exploits both labeled and unlabeled data. The proposed method consists of two main components: a domain-specific feature learning network and a common feature learning network. The domain-specific network is trained using only labeled data from a specific domain, whereas the common feature learning network is trained on a large corpus of apps from multiple domains using a semi-supervised learning approach. The learned common features are then used to improve the performance of the domain-specific network by fine-tuning. The authors evaluate the proposed approach on a dataset of apps from four domains (games, education, health & fitness, and shopping) and demonstrate that it outperforms several baseline methods. They also conduct experiments to show the effectiveness of the common feature learning network in improving the performance of the domain-specific network. Finally, they conduct a case study to demonstrate the usefulness of the proposed approach in recommending features for app developers in a specific domain. Overall, the paper proposes a promising approach for automatic feature extraction from mobile apps that could help app developers in feature engineering and reduce the time and effort required in manual feature selection.

**[10] Lengkong, O., & Maringka, R. (2020, October). Apps Rating Classification on Play Store Using Gradient Boost Algorithm. In 2020 2nd International Conference on Cybernetics and Intelligent System (ICORIS) (pp. 1-5). IEEE.**

The paper presents an approach to classify the app ratings on the Google Play Store using the Gradient Boost algorithm. The authors collected data from the Play Store and performed preprocessing techniques such as data cleaning, data transformation, and data reduction. They also used feature selection techniques to select the relevant features for the classification task. The selected features were then used to train the Gradient Boost algorithm. The performance of the proposed approach was evaluated using metrics such as accuracy, precision, recall, and F1-score. The experimental results show that the proposed approach achieved an accuracy of 84.85%, which is higher than the accuracy of other classification algorithms such as Decision Tree, Naive Bayes, and Random Forest. The authors also discussed the limitations and future directions of the proposed approach, such as improving the data preprocessing techniques and exploring other classification

algorithms. The paper contributes to the field of machine learning and data mining by presenting a novel approach to classify app ratings on the Google Play Store.

**[11] Liu, Y., Liu, L., Liu, H., Wang, X., & Yang, H. (2019). Mining domain knowledge from app descriptions. Journal of Systems and Software, 133, 126-144.**

The paper presents a novel approach to mine domain knowledge from app descriptions in the Google Play Store. The authors propose a framework that can automatically identify key phrases in app descriptions and classify them into predefined categories using natural language processing techniques. The proposed framework consists of two main components: a phrase extraction module and a classification module. The phrase extraction module uses a combination of linguistic rules and statistical methods to identify key phrases from the app descriptions. The classification module then uses a machine learning algorithm to classify the extracted phrases into predefined categories such as functionality, usability, and design. The authors evaluated the proposed framework on a dataset of 5000 app descriptions and achieved an average accuracy of 85% in classifying the extracted phrases into the predefined categories. The proposed approach can be used to improve app search and recommendation systems by providing more accurate and relevant results based on the user's search query or profile. It can also assist developers in designing and developing better apps by analyzing the strengths and weaknesses of existing apps in the same category.

## 2.2 Summary/Gaps/Limitations/Future Work identified in the Survey

| Details | Brief Summary | Observations | Limitations |
|---|---|---|---|
| P. B. Prakash Reddy and R. Nallabolu, "Machine learning based Descriptive Statistical Analysis on Google Play Store Mobile Applications," 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2020, pp. 647-655, Doi: 10.1109/ICIRCA48905.2020.9183271.<br><br>IEEE 2020 | ● The dataset has got 11 attributes for each application - app name, category, rating, reviews, installs, size, price, content rating, last updated, minimum android version, latest app version.<br><br>● To compare all categories with all the attributes under one roof, the attributes are standardized based on the min-max standardization. Analyzed the Count, Mean, Standard deviation, min, max, 25%, 50%, and 75% of all the number attributes and modeled the correlation matrix | ● The authors observed that some of the key factors that contribute to the success of an app on the Google Play Store are the number of downloads, the user ratings, and the price of the app.<br><br>● They also found that machine learning algorithms can be effectively used to analyze large amounts of data from the Play Store and can help app developers make data-driven decisions.<br>● Based on the given dataset and the performed analysis it is evident that the Ratings, Reviews, Price, Size, and Installs have no strong correlation. | ● Since the dataset is not enough to create a machine learning model, the future work should be focused on gathering the data that suits to create a machine learning model.<br><br>● The study is limited to the specific set of features analyzed, and other features may also have an impact on the success of the app. |
| R. M. Amir Latif, | ● Used a | ● There is a significant correlation | ● Scraping techniques used aren't |

| | | | |
|---|---|---|---|
| M. Talha Abdullah, S. U. Aslam Shah, M. Farhan, F. Ijaz and A. Karim, "Data Scraping from Google Play Store and Visualization of its Content for Analytics," 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), Sukkur, Pakistan, 2019, pp. 1-8, Doi: 10.1109/ICOMET.2019.8673523.<br>**IEEE 2019** | Google-Play-Scraper to build a Google Play Store dataset with all categories of games.<br>● Scraping at least 550 applications of each category of games in free and respectively in paid applications by using Google play scraper, cumulatively scraping the 3600 paid applications and 10k free applications of all categories in games.<br>● The categories of games are then separated into Free and Paid versions and 4 important attributes are selected for analysis namely, Ratings, InApplication Purchases, Advertisements Support, and Installs. Visualization is performed using RStudio and CIRCOS. | between the number of downloads and the ratings of the app as visualized using CIRCOS.<br><br>● Spectrums of visualizations have been plotted such as pie charts, histograms and bar plots for mapping trends and attributes analysis. | effective to build a robust dataset which might have resulted in anomalies in the results.<br><br>● Needs more work with respect to obtaining a greater number of attributes. |

| | | | |
|---|---|---|---|
| Venkatakrishnan, S., Kaushik, A., Verma, J.K. (2020). Sentiment Analysis on Google Play Store Data Using Deep Learning. In: Johri, P., Verma, J., Paul, S. (eds) Applications of Machine Learning. Algorithms for Intelligent Systems. Springer, Singapore. https://doi.org/10.1007/978-981-15-3357-0_2 **Springer 2020** | • A comprehensive and extensive research that performed sentiment analysis on google play store data using deep learning algorithms.<br>• Includes Convolutional Neural Networks and LSTM.<br>• Analyzed the different variants of data together by combining text, numerical and categorical data.<br>• Performed text vectorization using Count and TF-IDF methods.<br>• Categorical Encoding. | • TF-IDF vectorization seems more appropriate and superior in some cases for text classification scenarios as it balances out the frequency of the entire text tensor under question using its inverse document frequency.<br>• Model successfully determines the app user ratings, given various play store app metrics and user reviews. | • A sub-optimal 70% accuracy is obtained using the trained model which can be further enhanced by permuting different neural nets.<br>• Only few variants of MLP models have been implemented which limits the research. |
| S. Shashank and B. Naidu, "Google Play Store Apps-Data Analysis and Ratings Prediction," Dec. 2020. https://mail.irjet.net/archives/V7/i12/IRJET-V7I1248.pdf **IRJET 2020** | • Proposes a methodology for analyzing mobile applications on the Google Play Store and predicting their ratings using machine learning.<br>• Performed data pre-processing to remove any noise and extract relevant features.<br>• Then applied machine learning algorithms, including decision trees, random forest, KNN, | • Significant pre-processing must be done before the model is trained.<br>• Can predict about 92% accuracy if an app will have more than 100,000 installs and be a hit on the Google Play Store.<br>• KNN algorithm has topped the accuracy figures, gaining a significant edge over even Ensemble methods. | • Accuracy attained is 69.56% with the maximum accuracy of an individual model obtained reaching 92%.<br>• Machine learning models can be further boosted by neural networks to obtain greater precision and accuracy.<br>• Accepts the need to include reviews-based data for classification purposes. |

| | | | |
|---|---|---|---|
| | KMeans and linear regression, to analyze the data and predict the ratings of the apps. | | |
| I. Journal, "IRJET-Play Store App Analysis," IRJET, Jan. 2021. Available: https://www.academia.edu/52123932/IRJET_Play_Store_App_Analysis **IRJET 2021** | ● The authors used web scraping techniques to collect data on various app features, including app category, ratings, reviews, price, and other metrics.<br>● They then applied machine learning algorithms, including linear regression and decision trees, to analyze the data and identify trends and patterns.<br>● The parameters used include App, Category, Ratings, Reviews, Size, Installs, Price and Genres. | ● Past trends of all applications can be effectively computed and visualized.<br>● Contains immense possibilities to improve business values and have a positive impact. | ● Correlations between various attributes haven't been explored which could have been useful in optimizing the model aspects.<br>● No details on the inferences deduced on training the model.<br>● Excessive focus on User Interface. |
| Maredia, Rimsha. (2020). Analysis of Google Play Store Data set and predict the popularity of an app on Google Play Store. **Texas A&M University 2020** | ● Performed Classification Analysis using Decision Tree, KNN, Gaussian Naïve Bayes and Logistic Regression models.<br>● Decision tree model has been used in data visualization for comprehensive prediction of regression values.<br>● Analysis approach has been divided into three phases: data extraction, data cleaning and visualization and data modeling. | ● Found that Decision trees fits best for the given problem statement based on statistics.<br>● Decision tree is easy to visualize and explain the model implementation and it also saves computational power. It gave the maximum accuracy of 95.32% among all the models analyzed. | ● Limitations concerning accuracy statistics of machine learning models used.<br>● Problem of overfitting noticed. |

| | | | |
|---|---|---|---|
| Bashir, G. M. M., Hossen, M. S., Karmoker, D., & Kamal, M. J. (2019, December). Android apps success prediction before uploading on google play store. In 2019 International Conference on Sustainable Technologies for Industry 4.0 (STI) (pp. 1-6). IEEE. **IEEE 2019** | ● Their study predicts a new app's performance by predicting its user rating and installation quantity before uploading to the Google Play Store.<br>● They have used a scraped dataset which contains 267000 unique apps data.<br>● They compared predicted and original ratings and installations to predict app success.<br>● They have built various Bar Charts to demonstrate the link between its parameters like category, rating, size, installs, types, etc.<br>● They have used Random Forest, K-Nearest Neighbor, and Support Vector Machine to predict app success. | ● The KNN and SVM algorithms predicted success rates better than the Random Forest after comparing original and predicted ratings and installations.<br><br>● Their suggested idea will make it easier for the developers to decide whether they are moving in the right direction or not. | ● They have not predicted the type of review and hence there is mismatch in ratings and reviews.<br><br>● The correlation between the sizes of the apps, the version of android on the number of installs was not considered. |
| MarediaGao, S., Liu, L., Liu, Y., Liu, H., & Wang, Y. (2021). API recommendation for the development of Android App features based on the knowledge | ● This paper uses App store data to recommend Android APIs. First, they created API-functionality mappings using UI elements. Second, they created a framework to describe Apps in the same category and use API relationships to build API knowledge for each node. | ● According to test results, their method can accurately extract feature and API information from pertinent data (the average accuracy of supplemental data used for mining features of UI elements without textual information is 88.50%). | ● To suggest the combinations of APIs for features to be realized, mining of other relevant data, such as API official documents and Java code of existing products, was not taken into consideration. |

| | | | |
|---|---|---|---|
| mined from App stores. Science of Computer Programming, 202, 102556.<br>**Elsevier 2021** | ● Thirdly, they identified nodes by queried features and gave developers recommendation lists to demonstrate API knowledge. Based on the Google Play store, they tested this approach. | | |
| Song, G., Hu, L., & Liu, Y. (2020). Common but innovative: learning features from apps in different domains. IEEE Access, 8, 186904-186918.<br>**IEEE 2020** | ● This paper proposes a method to help developers learn app features from products in different domains.<br>● They first extracted features and relationships from App descriptions to describe one domain. Then, similar domain features are used as bridges to search for developer-reusable information.<br>● Finally, they created an interactive recommendation system to help developers learn. They used Google Play data to test their approach. | ● The experiments' findings, which were supported by the dataset from Google Play, indicate that their method can identify features that are similar to those shared by various domains with an average precision of 82.38%, and that these features can indicate how relevant a domain is to others.<br><br>● Additionally, the survey of developers reveals that the knowledge suggested by their strategy is helpful for updating Apps and motivating developers to come up with fresh, original ideas. | ● More features from the dataset could have been used to improve the efficiency.<br>● The dataset used in this experiment is limited and their study uses only 8 domains and hence when applied to unpopular domains, the results obtained would not be reliable. |
| Lengkong, O., & Maringka, R. (2020, October). Apps Rating Classification on Play Store Using | ● In order to identify the qualities of highly rated apps, this paper examined various features of apps on Google Play Store.<br>● 10-fold cross validation was also employed on the dataset. | ● The Gradient Boost algorithm, which outperforms Random Forest, K-NN, and Decision Tree algorithm with:<br><br>● a 99.93% accuracy<br>● 99.91% recall | ● Further performance improvement and lower error levels during the modeling process could have been achieved by using alternative techniques and algorithms. |

| | | | |
|---|---|---|---|
| Gradient Boost Algorithm. In 2020 2nd International Conference on Cybernetics and Intelligent System (ICORIS) (pp. 1-5). IEEE.<br>**IEEE 2020** | ● A random-forest classifier determines which features of highly rated Google Play Store apps are most important.<br>● The Gradient Boost Algorithm was also used to determine which characteristics contributed most to high-rated apps on the Google Play Store. | ● 99.91% precision<br>● 0.062 Root Mean Squared Error<br>● was used by the authors to categorize the highly rated apps. | |
| Liu, Y., Liu, L., Liu, H., Wang, X., & Yang, H. (2019). Mining domain knowledge from app descriptions. Journal of Systems and Software, 133, 126-144.<br>**Scopus 2020** | ● They have proposed an approach to mine domain knowledge from App descriptions automatically.<br>● In their approach, the information of features in a single app description is first extracted and formally described by a Concern-based Description Model (CDM), which is based on predefined rules of feature extraction and a modified topic modeling method;<br>● Then the overall domain knowledge is identified by classifying, clustering, and merging the knowledge in the set of CDMs and topics, and the results are formalized by a Data-based Raw Domain Model (DRDM).<br>● They also proposed a quantified | ● Experiments show that automatic CDM construction performs well: F-measure is above 0.8 and variation tendency is stable. This approach can be used for feature extraction in other domain analysis.<br>● DRDM supports domain analysis, especially for overall domain analysis and creative idea generation, according to surveys. | ● It is uncertain whether their work can produce comparable results when applied to other app marketplaces because two datasets were created based on Google Play.<br>● Their quantified method uses a small number of market attributes. They used two attributes (Rating and Downloads) to rank the importance of domain knowledge, but real app markets use more attributes (like "Price" and "Time"). |

| | | | |
|---|---|---|---|
| | DRDM knowledge prioritization method. | | |

# 3. Overview of the Proposed System

## 3.1 Introduction

### 3.1.1 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is an approach to analyzing and visualizing data in order to summarize its main characteristics and identify patterns, trends, relationships, and anomalies. The goal of EDA is to gain a deeper understanding of the data and its underlying structure, which can then be used to inform further analysis and modeling.

EDA typically involves several steps, which may include data cleaning and preprocessing, visualization, summary statistics, and hypothesis testing. The first step in EDA is to examine the data for any errors, missing values, or outliers, and to clean and preprocess the data if necessary. This may involve removing duplicates, filling missing values, or transforming the data into a more suitable format.

The next step in EDA is to visualize the data using various techniques such as histograms, scatter plots, box plots, and heat maps. These visualizations can help to reveal patterns, trends, and relationships between variables. For example, a scatter plot can be used to visualize the relationship between two continuous variables, while a box plot can be used to visualize the distribution of a continuous variable across different categories.

Summary statistics such as mean, median, mode, standard deviation, and correlation coefficients can also be used to summarize the main characteristics of the data. These statistics can provide insights into the central tendency, variability, and linear relationships between variables in the data.

EDA is an important first step in any data analysis or modeling project, as it helps to identify potential issues with the data and to gain a deeper understanding of its underlying structure. EDA can also help to inform the selection of appropriate statistical methods and models for further analysis. Overall, EDA is a valuable tool for data scientists and analysts in their quest to gain insights from complex data.

### 3.1.2 Random Forest Regression

Random Forest Regression is a machine learning algorithm used for regression analysis. It is an ensemble learning method that combines multiple decision trees and generates a forest of such trees. The Random Forest algorithm is a supervised learning method, which means that it requires a labeled dataset for training. It is useful for both classification and regression tasks.

In the case of Random Forest Regression, the algorithm constructs a set of decision trees, where each tree is trained on a random subset of the data and a random subset of the features. The idea behind this approach is to reduce overfitting and variance by reducing the correlation between the individual trees.

The algorithm works by creating a forest of decision trees where each tree is constructed using a random subset of the data points and a random subset of the features. The number of trees in the forest is determined by the user, and each tree is built using a different subset of data points and features. During training, the algorithm first selects a random subset of features to split on, then finds the feature that maximizes the split using a specified criterion (such as Gini impurity or entropy). This process continues recursively, with each new node being split based on a subset of the features until the tree is fully grown. The algorithm then repeats this process to construct a forest of decision trees. During prediction, the algorithm takes the input data and passes it through each tree in the forest, and the final prediction is the average of the predictions made by each tree. This approach helps to improve the accuracy and generalization of the model, as the average prediction of multiple trees is likely to be more accurate than a single tree.

One of the advantages of the Random Forest Regression algorithm is that it is highly scalable and can handle large datasets with high-dimensional features. It is also robust to noise and missing data, making it useful for real-world problems. Additionally, the Random Forest Regression algorithm provides a measure of feature importance, which can be used to identify the most important features in the dataset.

### 3.1.3 K-Means Clustering

K Means clustering is a machine learning algorithm used for unsupervised clustering. It is a simple yet powerful method that groups data points into k clusters, where k is a user-defined parameter. The algorithm aims to minimize the distance between each data point and its assigned cluster center, while maximizing the distance between different cluster centers.

The K Means algorithm works by iteratively partitioning the data into k clusters, where each data point is assigned to the cluster with the closest cluster center. The cluster center is then updated as the mean of all the data points assigned to that cluster. This process continues until convergence, where the cluster assignments no longer change.

The K Means algorithm has a few key parameters that need to be defined before the algorithm is run. The first parameter is k, which is the number of clusters to be generated. Choosing the right value of k is important as it directly impacts the quality of the clustering results. There are various methods for selecting the optimal value of k, including the elbow method, silhouette score, and gap statistic. The second parameter is the distance metric used to calculate the distance between data points and cluster centers. The most commonly used distance metrics are Euclidean distance and Manhattan distance, but other distance metrics can also be used depending on the nature of the data.

K Means clustering has several advantages that make it a popular clustering algorithm. Firstly, it is easy to implement and computationally efficient, making it suitable for large datasets. Secondly, it is a flexible algorithm that can be applied to a wide range of data types and can handle both continuous and categorical data. Finally, K Means clustering is a simple and interpretable method that provides meaningful insights into the structure of the data.

## 3.2 Architecture diagram and Modules of the Proposed System

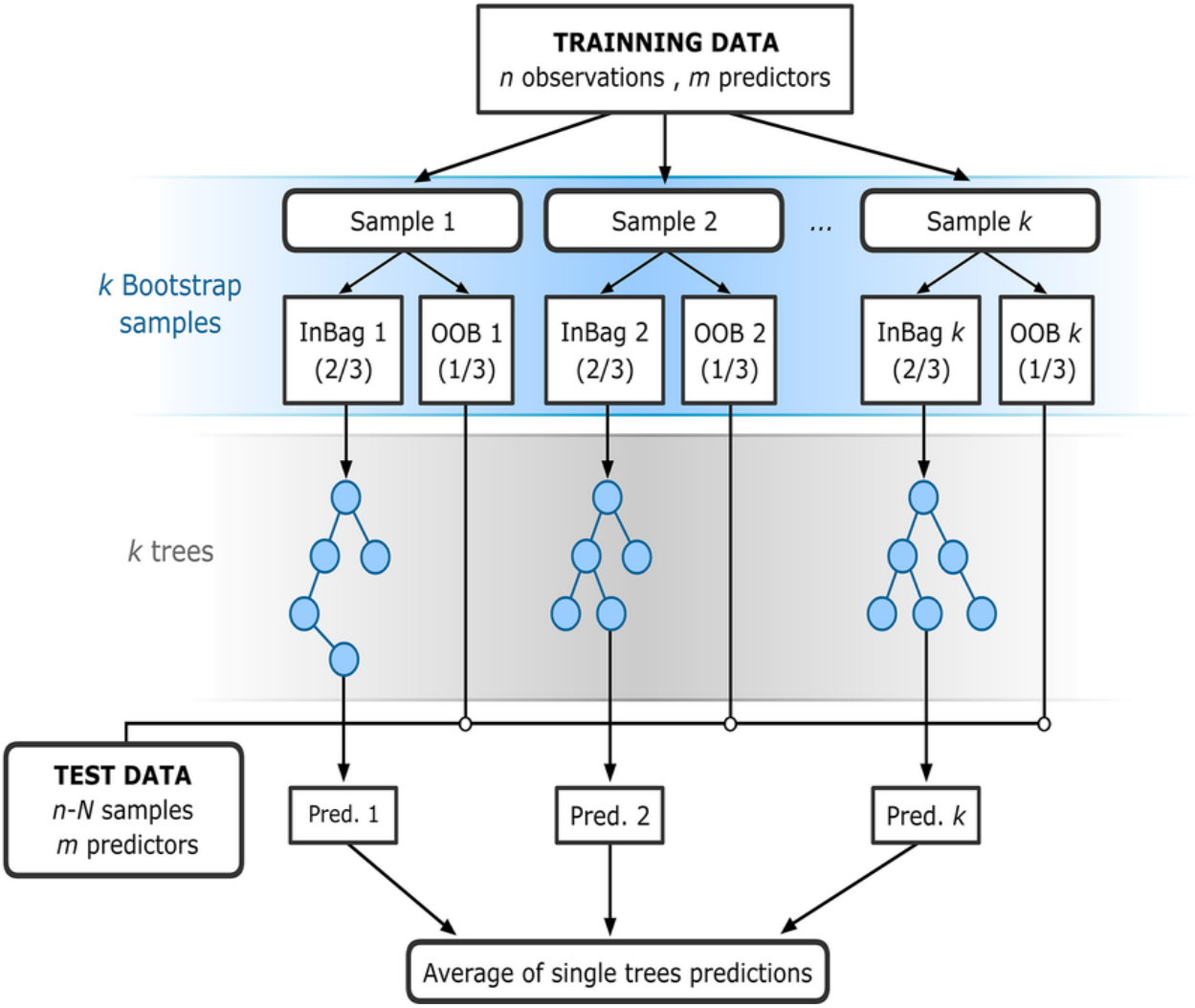



***Figure 1 : Random Forest Regression***

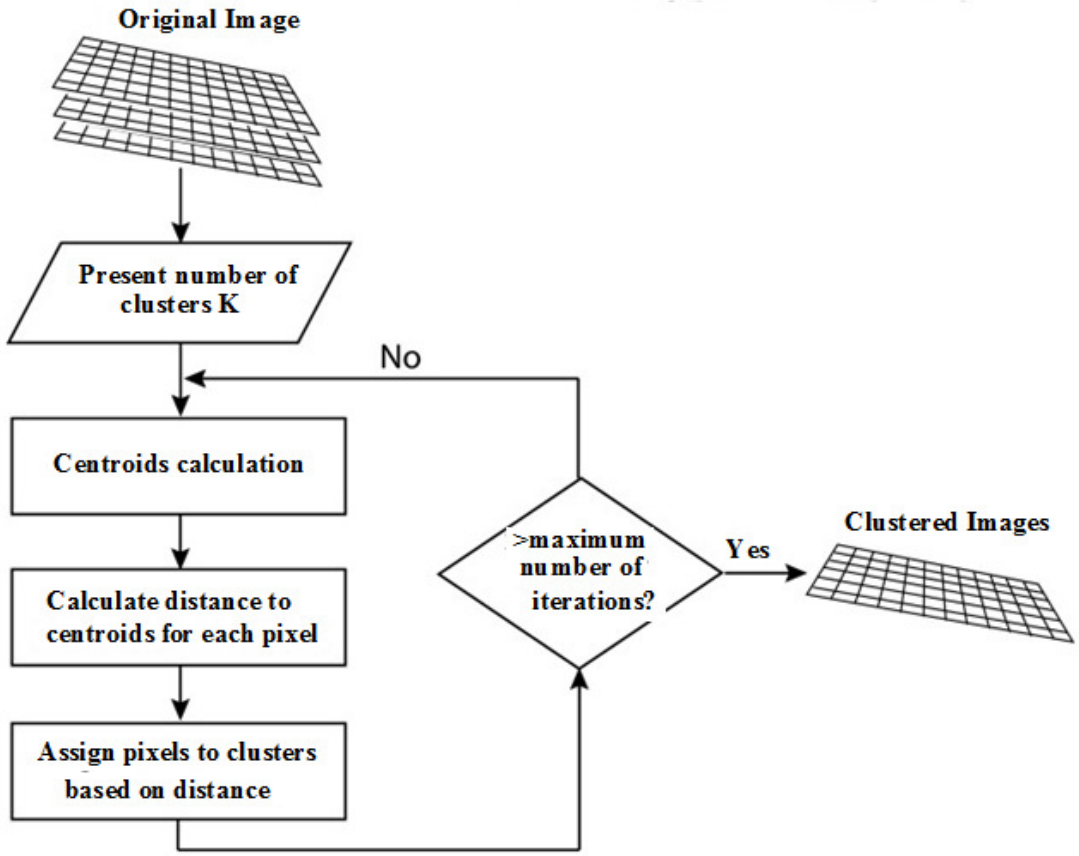



*Figure 2 : K-Means Clustering*

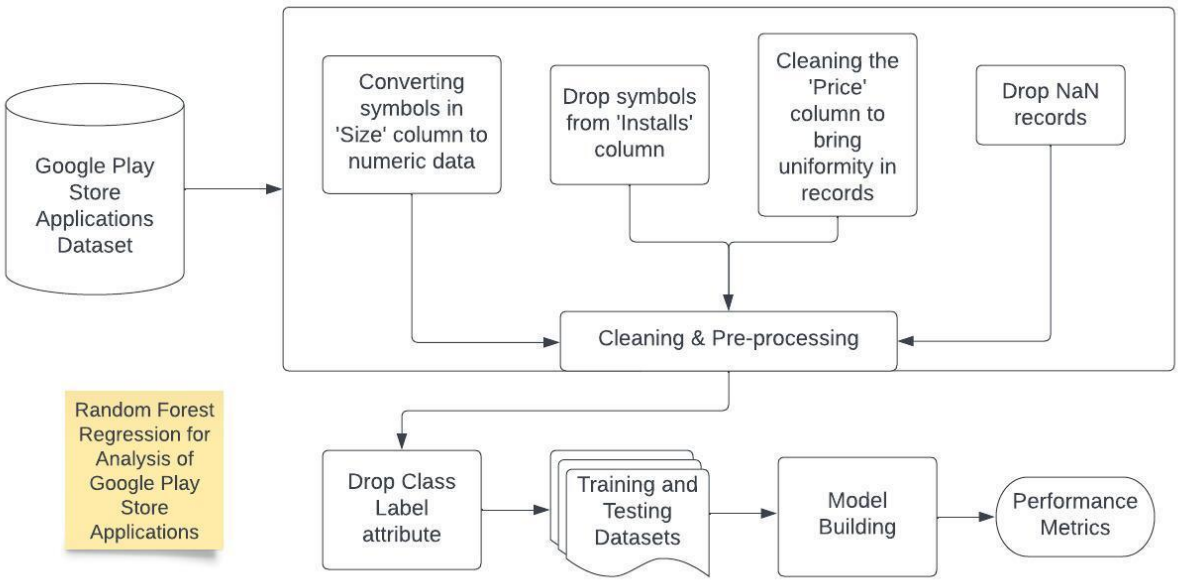



*Figure 3 : Random Forest Regressor Architecture*

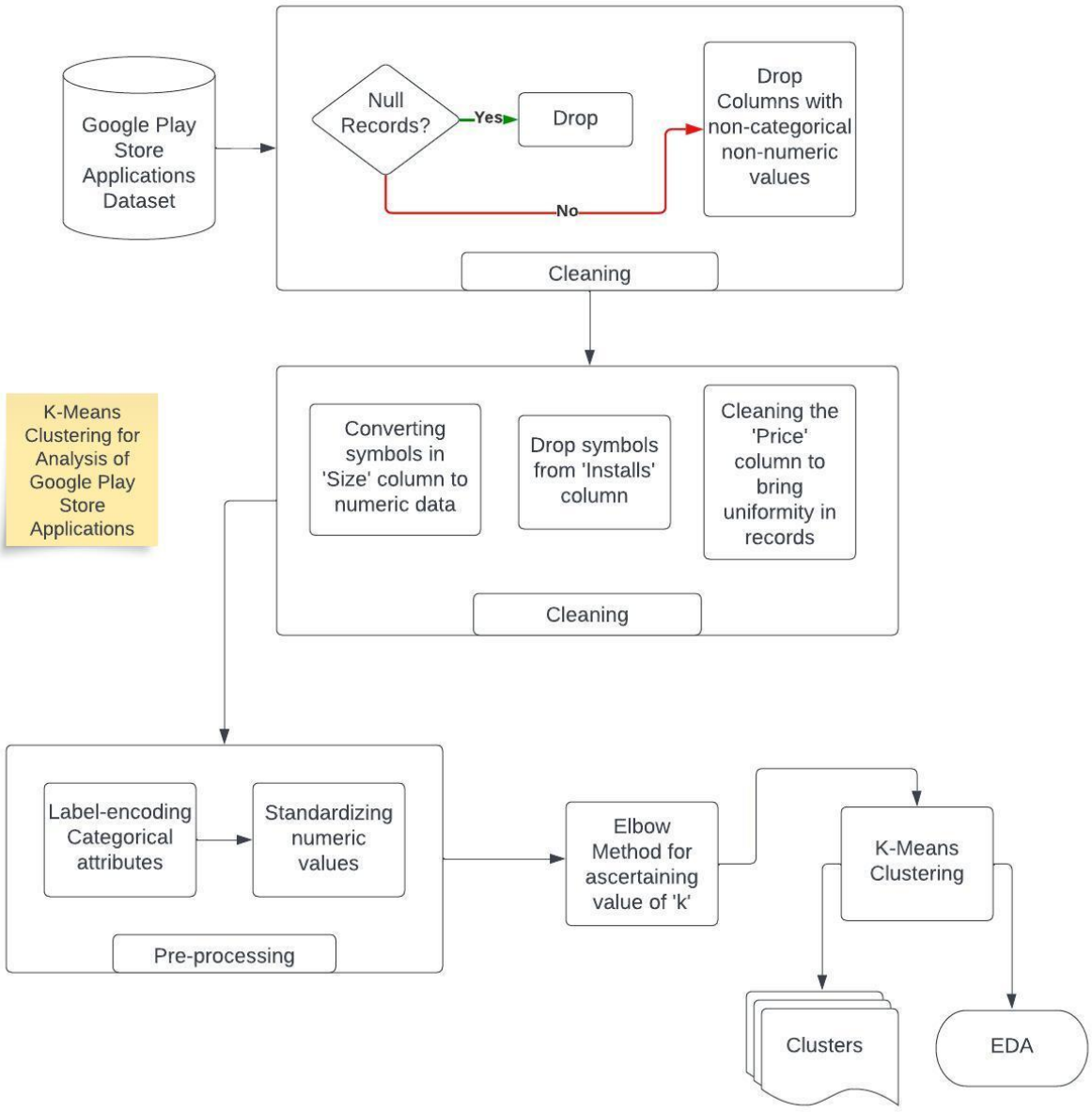



***Figure 4 : K-Means Model Architecture***

### 3.2.1 Novelty in our proposed work

The project focused on the analysis of Google Play Store applications, which is a relatively underexplored area in the domain of data science. The novelty lies in the fact that there has been very little high-quality work done in this field, which opens up opportunities for fresh research and insights.

Furthermore, the project involves the application of clustering techniques for the first time. This lays the foundation to perform market segmentation analysis, which is a critical component of any marketing strategy. The use of clustering helps to identify distinct groups of users with similar preferences and characteristics, allowing businesses to tailor their marketing efforts accordingly.

In addition to clustering, the project also includes the use of random forest regression to predict ratings of the applications. This approach has not been seen in previous works and can provide valuable insights for businesses to improve their products and services. Overall, the project's unique combination of clustering and ratings prediction using random forest regression provides a novel and valuable contribution to the field of data science and market analysis.

## 3.3 Proposed System Model

The process of building a random forest regression model involves several mathematical equations. One important step is to determine the importance of each feature, which is used to decide which features to include in each decision tree. There are several metrics used to calculate the importance of each feature, such as the Gini index, the mean decrease impurity, or the permutation importance.

The Gini index is a measure of the degree of impurity in the data. It is calculated as follows:
**Gini index = 1 - sum(p_i^2)**
where p_i is the proportion of data points in the i-th class. A feature with a high Gini index is considered to be more important, as it is a better predictor of the target variable. The Gini index is used to calculate the mean decrease impurity, which is another metric used to measure the importance of each feature.

The mean decrease impurity is calculated as follows:
**Mean decrease impurity = sum(Gini index(parent) - Gini index(children))**
where Gini index(parent) is the Gini index of the parent node and Gini index(children) is the weighted sum of the Gini indices of the child nodes. This metric measures the reduction in impurity achieved by splitting on a particular feature. A feature with a high mean decrease impurity is considered to be more important, as it leads to a greater reduction in impurity.

The permutation importance is another metric used to calculate the importance of each feature. It is calculated by randomly permuting the values of each feature and measuring the increase in the mean squared error of the model. The permutation importance measures how much the model performance decreases when a particular feature is removed from the model. It is calculated as follows:
**Permutation importance = sum((error - permuted_error)/error)**
where error is the mean squared error of the original model and permuted_error is the mean squared error of the model with the permuted feature. The summation is over all the data points. A feature with a high permutation importance is considered to be more important, as it leads to a greater decrease in model performance when removed.

Once the importance of each feature is determined, the random forest regression model can be trained using the selected features. The final prediction of the model is calculated as the average of all the predictions made by the individual decision trees. It is calculated as follows:
**y_hat = (1/N) * sum(y_i)**
where y_hat is the predicted value, N is the number of decision trees in the random forest, and y_i is the predicted value of the i-th decision tree. The summation is over all the decision trees in the random forest.

K-means clustering is a popular unsupervised machine learning algorithm used for clustering data points into groups or clusters based on their similarity. The algorithm works by iteratively assigning each data point to the nearest cluster center and updating the cluster centers based on the newly assigned data points. This process continues until the cluster assignments and cluster centers converge to stable values.

The mathematical equations involved in k-means clustering are as follows:
Initialization:
The algorithm first randomly selects k initial cluster centers, denoted by **μ_1, μ_2, ..., μ_k.**
Assignment:
Each data point x_i is assigned to the nearest cluster center μ_j, based on the Euclidean distance between the data point and the cluster center. This is done using the following equation:
**argmin_j ||x_i - μ_j||^2**

where **||x_i - μ_j||^2** is the squared Euclidean distance between x_i and μ_j, and argmin_j returns the index j of the cluster center that minimizes the distance.

Update:

After all the data points are assigned to clusters, the cluster centers are updated based on the newly assigned data points. The new cluster centers μ_j' are calculated as the mean of all the data points assigned to that cluster. This is done using the following equation:
**μ_j' = (1/|C_j|) * sum_{x_i in C_j} x_i**

where C_j is the set of data points assigned to cluster j, |C_j| is the number of data points in cluster j, and sum_{x_i in C_j} x_i is the sum of all the data points in cluster j.

Repeat:
Steps 2 and 3 are repeated until the cluster assignments and cluster centers converge to stable values.
The objective of k-means clustering is to minimize the within-cluster sum of squares (WCSS), which is the sum of the squared distances between each data point and its assigned cluster center. The WCSS can be calculated using the following equation:

**WCSS = sum_{j=1}^k sum_{x_i in C_j} ||x_i - μ_j||^2**

where j is the index of each cluster, k is the total number of clusters, x_i is the i-th data point, C_j is the set of data points assigned to cluster j, and **||x_i - μ_j||^2** is the squared Euclidean distance between x_i and μ_j.

# 4. Proposed System Analysis and Design

First of all, we are importing all the necessary libraries such as numpy, pandas, seaborn and matplotlib. We are then taking two datasets as input. The first dataset is related to the different applications of Google Play Store and the second dataset is related to the User Reviews of the applications. We are then doing the Data Preprocessing of the first dataset. We are first checking for missing values and we found some missing values in the dataset. We considered dropping those rows, as they were not very important and were not having a big influence in the analysis. The Installs column of the dataset was having string characters such as '+' and ','. We removed these characters so that the column values get converted to pure numbers. Similarly, the Price column was containing values with symbols like '$', so we removed those symbols too. The variance for all the numerical columns was then checked and it was found that the Installs column had a very large variance, so we applied logarithm to the values of that column to lower the variance. This would also help in better Data Visualization of the data. We also checked for duplicate rows in the dataset and then removed all the duplicate rows to avoid redundancy in the

dataset. After the end of the Data Preprocessing phase, we started the Plotting of Data to get insights about the data. We did plotting for various app developers and common audience related questions such as: “What are the top 20 apps in the Google Play Store organized by genre?”, “What are the most popular Genres among the top 20 Genres?”, “What are the Highest and Lowest Performing Genres?”, “What is the count of applications in each category differentiated by their type?”, “How many apps were installed according to its type?”, “What is the Sentiments percentages for different types of reviews?”, etc. After the Data Visualization phase, we went to the Machine Learning phase. We have implemented two algorithms for analysis. The first is the Random Forest Regression Algorithm and the second is the K-Means Clustering Algorithm. We again preprocessed the data to make it suitable for Machine Learning. We converted all categories to numerals so that the model can take it. We also converted the COntent Rating to numerals for making the column suitable for the model. We dropped the columns such as Last Updated, Current Ver, Android Ver, and App as they were irrelevant for our models. We converted the values of the Genres column also to numerals. The values of the Price column were also converted to float data types. After doing all this, we converted all the categorical variables to numbers using the python get_dummies method. We created functions to print the various types of errors such as: Mean Squared Error, Mean Absolute Error, and Mean Squared Log Error for evaluating the performance of our model. We then created our Machine Learning model: Random Forest Regressor. We did a train-test split of the dataset keeping test data as 30% of the complete data. We are trying to apply regression by keeping the Ratings of the applications as the dependent variable. The model would be able to predict the Ratings of an application based on the given inputs like Reviews, Size, Installs, Price, etc. After the regression is getting completed, we are also showing what variables are affecting the Ratings of the application a lot. After the Regression, we have performed K-Means Clustering on the dataset. We scaled the dataset so that all the numerical values stay between the same scale. We used the Elbow Method to depict the optimal number of clusters or the value of K and found it to be 6. We performed K-Means Clustering and 6 clusters were formed in the dataset. After finding the clusters, we have done Data Visualizations to get insights and trends about the data.

## 5. Implementation

### *Data Input*

```
[ ]  data1 = pd.read_csv("/content/drive/MyDrive/Data Mining Project Component/Datasets/googleplaystore.csv")
     data2 = pd.read_csv("/content/drive/MyDrive/Data Mining Project Component/Datasets/googleplaystore_user_reviews.csv")
```

### *Data Preprocessing*

```
[ ]  data1.isna().sum()

App                0
Category           0
Rating          1474
Reviews            0
Size               0
Installs           0
Type               1
Price              0
Content Rating     1
Genres             0
Last Updated       0
Current Ver        8
Android Ver        3
dtype: int64
```

```
[ ]  data1_1 = data1.dropna()
```

```
[ ]  data1_1['Installs'] = data1_1['Installs'].map(lambda x: x.rstrip('+'))

<ipython-input-12-b81741e21d9b>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1_1['Installs'] = data1_1['Installs'].map(lambda x: x.rstrip('+'))
```

```
[ ]  data1_1['Installs'] = pd.to_numeric(data1_1['Installs'].str.replace(',',''))

<ipython-input-13-72ad1ae10485>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1_1['Installs'] = pd.to_numeric(data1_1['Installs'].str.replace(',',''))
```

```
[ ]  data1_1['Price'] = pd.to_numeric(data1_1['Price'].str.replace('$',''))

<ipython-input-14-303564ef37c2>:1: FutureWarning: The default value of regex will change from True to False in a future version. In addition, single
  data1_1['Price'] = pd.to_numeric(data1_1['Price'].str.replace('$',''))
<ipython-input-14-303564ef37c2>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1_1['Price'] = pd.to_numeric(data1_1['Price'].str.replace('$',''))
```

```
[ ]  data1_1.var()
```

```
<ipython-input-16-b8e6293d39a7>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a
  data1_1.var()
Rating      2.654959e-01
Installs    8.329549e+15
Price       2.503243e+02
dtype: float64
```

```
[ ]  data1_1['Installs with Log'] = np.log(data1_1["Installs"])
```

```
<ipython-input-17-250f173b120b>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1_1['Installs with Log'] = np.log(data1_1["Installs"])
```

```
[ ]  data1_1["App"].duplicated().sum()
```

```
1170
```

```
[ ]  data1_1.drop_duplicates(inplace = True)
```

```
<ipython-input-21-738895c7e2af>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1_1.drop_duplicates(inplace = True)
```

```
[ ]  data1_1['Reviews'] = pd.to_numeric(data1_1['Reviews'].str.replace('$',''))
```

```
<ipython-input-24-f07ee014e66a>:1: FutureWarning: The default value of regex will change from True to False in a future version. In addition, single charact
  data1_1['Reviews'] = pd.to_numeric(data1_1['Reviews'].str.replace('$',''))
<ipython-input-24-f07ee014e66a>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1_1['Reviews'] = pd.to_numeric(data1_1['Reviews'].str.replace('$',''))
```

## *Data Visualization*

```
[ ]  plt.figure(figsize=(14,7))
     plt.xticks(rotation=65)
     plt.xlabel("Genres")
     plt.ylabel("Number of Application")
     plt.title("Top 20 Genres")
     sns.barplot(x = install_20.Genres, y = install_20.Count)
     plt.show()
```

```
[ ]  plt.figure(figsize=(14,7))
     plt.xticks(rotation=65)
     plt.xlabel("Genres")
     plt.ylabel("Installs")
     plt.title("Installs Vs. Genres")
     sns.barplot(x = install_20.Genres, y = install_20.Installs)
     plt.show()
```

```
[ ]  plt.figure(figsize=(14,7))
     plt.xticks(rotation=65)
     plt.xlabel("Genres")
     plt.ylabel("Installs")
     plt.title("Installs Vs. Genres")
     sns.barplot(x = install_20.Genres, y = install_20.Installs)
     plt.show()
```

```
[ ]  best_cat = data1_1.Category.value_counts().reset_index().rename(columns={'Category':'Count','index':'Category'})
     cat_install = data1_1.groupby(['Category'])[['Installs']].sum()
     best_cat_install = pd.merge(best_cat, cat_install, on='Category')
     best_20_cat_install = best_cat_install
     plt.figure(figsize=(14,7))
     plt.xticks(rotation=90)
     plt.xlabel("Category")
     plt.ylabel("Number of application")
     plt.title("Applications that exist category wise")
     sns.barplot(x = best_20_cat_install.Category, y = best_20_cat_install.Count)
     plt.show()
```

```
[ ]  plt.figure(figsize=(14,7))
     plt.xticks(rotation=90)
     plt.xlabel("Category")
     plt.ylabel("Installs")
     plt.title("Category wise Installed Apps")
     sns.barplot(x = best_20_cat_install.Category, y = best_20_cat_install.Installs)
     plt.show()
```

```
[ ]  average_rating_of_apps = data1_1.groupby(['Genres'])[['Rating']].mean()
```

```
[ ]  ratings_of_genres = pd.merge(best_gen, average_rating_of_apps, on='Genres')
```

```
[ ]  plt.figure(figsize=(14,7))
     g = sns.kdeplot(ratings_of_genres.Rating, color="Red", shade = True)
     g.set_xlabel("Rating")
     g.set_ylabel("Frequency")
     plt.title('Rating Distribution',size = 20)
     plt.show()
```

```
[ ]  ratings_of_genres.sort_values('Rating', ascending =False, inplace=True)

[ ]  high_rating_gen = ratings_of_genres.iloc[0:20]
     low_rating_gen = ratings_of_genres.iloc[-20:]
     low_rating_gen = low_rating_gen[low_rating_gen['Rating'].notnull()]

[ ]  plt.figure(figsize=(14,7))
     plt.xticks(rotation=65)
     plt.xlabel("Genres")
     plt.ylabel("Rating")
     plt.title("Genre wise Rating")
     sns.barplot(x = high_rating_gen.Genres, y = high_rating_gen.Rating)
     plt.show()
```

```
[ ]  count_of_apps = data1_1.groupby(['Category','Type'])[['App']].count().reset_index().rename(columns={'App':'Count','index':'App'})
     count_of_apps_data1_1 = count_of_apps.pivot('Category', 'Type', 'Count').fillna(0).reset_index()
     count_of_apps_data1_1.set_index('Category').plot(kind='bar', stacked=True, figsize=(18,9))
     plt.xlabel("Category", fontsize=15)
     plt.ylabel("Count", fontsize=15)
     plt.title("The number of applications in each category, broken down by type.")
     plt.show()
```

```
[ ]  data1_1['Gaming Apps'] = data1_1['Category']=='GAME'
     install_by_cat = data1_1.groupby(['Category','Type'])[['Installs']].sum().reset_index()
     install_by_cat['log_Installs'] = np.log10(install_by_cat['Installs'])
     plt.figure(figsize=(18,9))
     plt.xticks(rotation=65,fontsize=9)
     plt.xlabel("Category")
     plt.ylabel("Installs(base10)")
     plt.title("According to Category, the number of instals in  log(base10)")
     sns.barplot(x = install_by_cat.Category, y = install_by_cat.log_Installs, hue = install_by_cat.Type);
     plt.show()
```

```
[ ]  plt.figure(figsize=(14,7))
     plt.title("Size has an effect on the number of installations in log(base10)")
     sns.scatterplot(x = data1_1['Size'], y = data1_1['Installs with Log'], hue=data1_1['Type'])
     plt.show()
```

```
[ ]  data1_1.loc[data1_1['Installs with Log']==data1_1['Installs with Log'].min(),'Installs with Log']=0
     plt.xlabel("Installs in Log")
     plt.title("Logrithm of Installs Distribution in  log(base10)")
     plt.hist(data1_1['Installs with Log']);
```

```
[ ]  plt.xlabel("Size")
     plt.title("Size Distribution")
     plt.hist(data1_1['Size']);
     plt.show()
```

```
[ ]  data_merge = data1_1.merge(data2, on="App")

[ ]  sentiments = data_merge.groupby(['Category','Sentiment']).size().reset_index(name='Sentiment Count')
     sentiments['Sentiments in Log'] = np.log2(sentiments['Sentiment Count'])
     plt.figure(figsize=(18,9))
     plt.xticks(rotation=90,fontsize=11)
     plt.xlabel("Category", fontsize=15)
     plt.ylabel("Installs", fontsize=15)
     plt.title("According to genres, the number of instals varies.",fontsize=15)
     sns.barplot(x = sentiments.Category, y = sentiments['Sentiments in Log'], hue = sentiments.Sentiment);
```

```
[ ]  plt.figure(figsize=(18,9))
     plt.xlabel("Subjectivity")
     plt.title("Subjectivity Distribution")
     plt.hist(data_merge[data_merge['Sentiment_Subjectivity'].notnull()]['Sentiment_Subjectivity'])
     plt.show()
```

```
[ ]  import matplotlib
     counting = list(data_merge['Sentiment'].value_counts())
     label_names = 'Positive', 'Negetive','Neutral'
     matplotlib.rcParams['font.size'] = 12
     matplotlib.rcParams['figure.figsize'] = (8, 8)
     plt.pie(counting, labels=label_names, explode=[0, 0.05, 0.005], shadow=True, autopct="%.2f%%")
     plt.title('The Percentage of Review Sentiments in a Pie Chart', fontsize=20)
     plt.axis('off')
     plt.legend()
     plt.show()
```

## *Machine Learning (Random Forest Regression)*

```
[ ]  def Evaluationmatrix(y_true, y_predict):
         print ('Mean Squared Error: '+ str(metrics.mean_squared_error(y_true,y_predict)))
         print ('Mean absolute Error: '+ str(metrics.mean_absolute_error(y_true,y_predict)))
         print ('Mean squared Log Error: '+ str(metrics.mean_squared_log_error(y_true,y_predict)))

[ ]  def Evaluationmatrix_dict(y_true, y_predict, name = 'Linear - Integer'):
         dict_matrix = {}
         dict_matrix['Series Name'] = name
         dict_matrix['Mean Squared Error'] = metrics.mean_squared_error(y_true,y_predict)
         dict_matrix['Mean Absolute Error'] = metrics.mean_absolute_error(y_true,y_predict)
         dict_matrix['Mean Squared Log Error'] = metrics.mean_squared_log_error(y_true,y_predict)
         return dict_matrix
```

```
[ ]  from sklearn.ensemble import RandomForestRegressor
     from sklearn import metrics
     from sklearn.model_selection import train_test_split
     import random

     X = df.drop(labels = ['Category','Rating','Genres','Genres_c'],axis = 1)
     y = df.Rating
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
     model3 = RandomForestRegressor()
     model3.fit(X_train,y_train)
     Results3 = model3.predict(X_test)

     resultsdf = pd.DataFrame()
     resultsdf = resultsdf.from_dict(Evaluationmatrix_dict(y_test,Results3),orient = 'index')
     resultsdf = resultsdf.transpose()

     resultsdf = resultsdf.append(Evaluationmatrix_dict(y_test,Results3, name = 'RFR - Integer'),ignore_index = True)

     X_d = df2.drop(labels = ['Rating','Genres','Category_c','Genres_c'],axis = 1)
     y_d = df2.Rating
     X_train_d, X_test_d, y_train_d, y_test_d = train_test_split(X_d, y_d, test_size=0.30)
     model3_d = RandomForestRegressor()
     model3_d.fit(X_train_d,y_train_d)
     Results3_d = model3_d.predict(X_test_d)

     resultsdf = resultsdf.append(Evaluationmatrix_dict(y_test,Results3_d, name = 'RFR - Dummy'),ignore_index = True)
```

```
[ ]  plt.figure(figsize=(12,7))
     sns.regplot(x = Results3, y = y_test,color='teal', label = 'Integer', marker = 'x')
     sns.regplot(x = Results3_d, y = y_test_d,color='orange',label = 'Dummy')
     plt.legend()
     plt.title('RFR model - excluding Genres')
     plt.xlabel('Predicted Ratings')
     plt.ylabel('Actual Ratings')
     plt.show()
```

```
[ ]  Feat_impt = {}
     for col,feat in zip(X.columns,model3.feature_importances_):
         Feat_impt[col] = feat

     Feat_impt_df = pd.DataFrame.from_dict(Feat_impt,orient = 'index')
     Feat_impt_df.sort_values(by = 0, inplace = True)
     Feat_impt_df.rename(index = str, columns = {0:'Pct'},inplace = True)

     plt.figure(figsize= (14,10))
     Feat_impt_df.plot(kind = 'barh',figsize= (14,10),legend = False)
     plt.show()
```

## *Machine Learning (K-Means Clustering)*

```
[ ]  # Importing Important Libraries
     from sklearn.preprocessing import StandardScaler
     from sklearn.cluster import KMeans
```

```
[ ]  df_scaled = StandardScaler().fit_transform(df)
     df_scaled = pd.DataFrame(df_scaled,columns=['Category', 'Rating', 'Reviews', 'Size', 'Installs',
                                                 'Type', 'Price', 'Content Rating', 'Genres'])
     DF = df_scaled.to_numpy()
     df_scaled
```

```
[ ]  wcss = []
     for i in range(2,9):
         kmeans = KMeans(n_clusters = i, init = 'k-means++',
                         max_iter = 300, n_init = 10,random_state = 0)
         kmeans.fit(df_scaled)
         wcss.append(kmeans.inertia_)
```

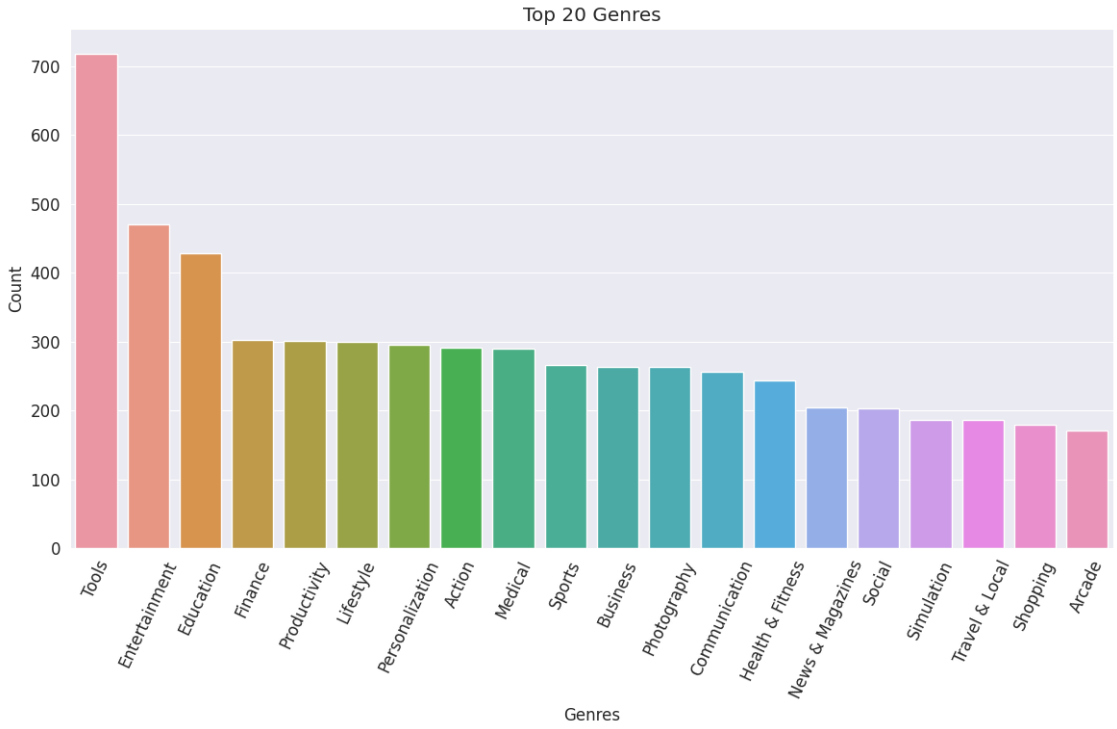
```
[ ]  plt.plot(range(2,9),wcss)
     plt.title('Elbow Method Plot')
     plt.xlabel('Number of Clusters')
     plt.ylabel('Within-Cluster Sum of Square') # Within cluster sum of squares
     plt.tight_layout()
     plt.show()
```

```
[ ]  kmeans = KMeans(n_clusters = 6, init = 'k-means++',
                     max_iter = 300, n_init = 10,random_state = 42)
     kmeans.fit(df_scaled)
```

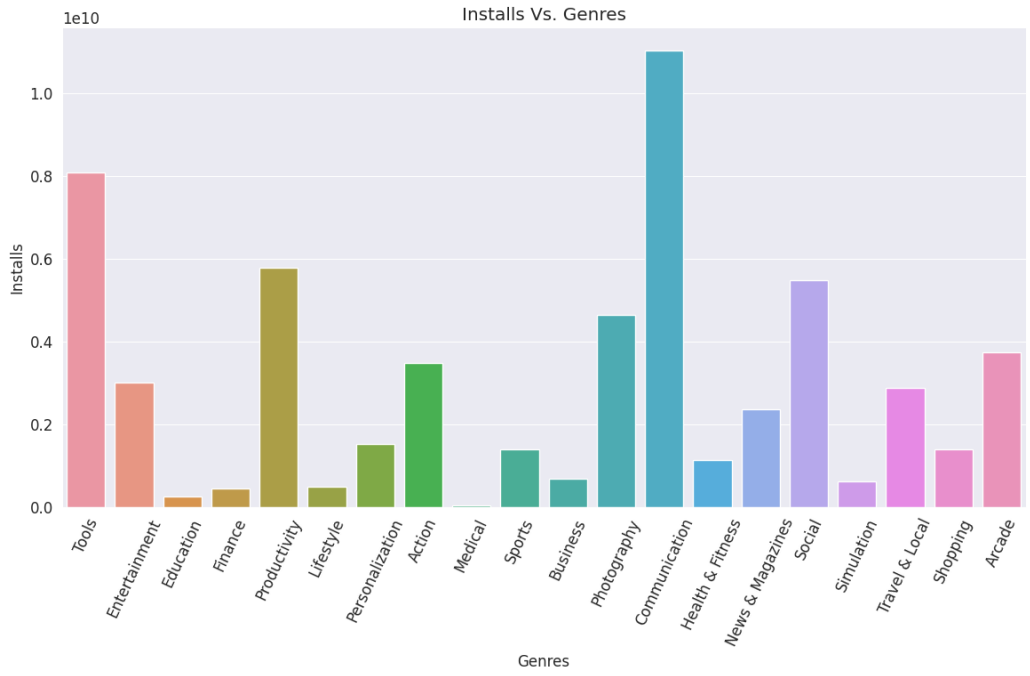
```
KMeans
KMeans(n_clusters=6, n_init=10, random_state=42)
```

## 6. Results and Discussions

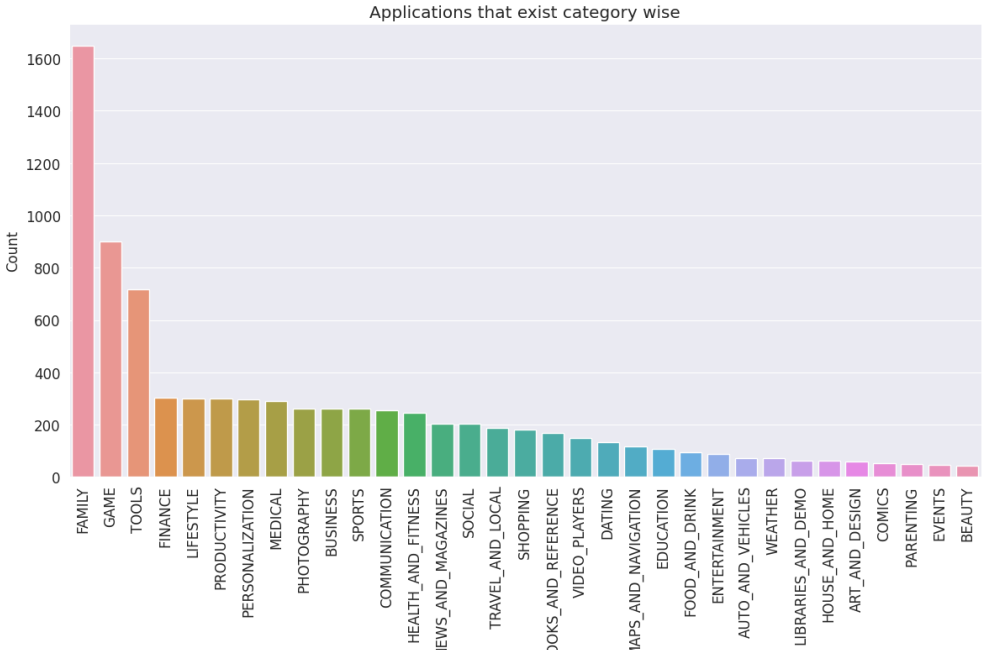
### What are the top 20 apps in the Google Play Store organized by genre?

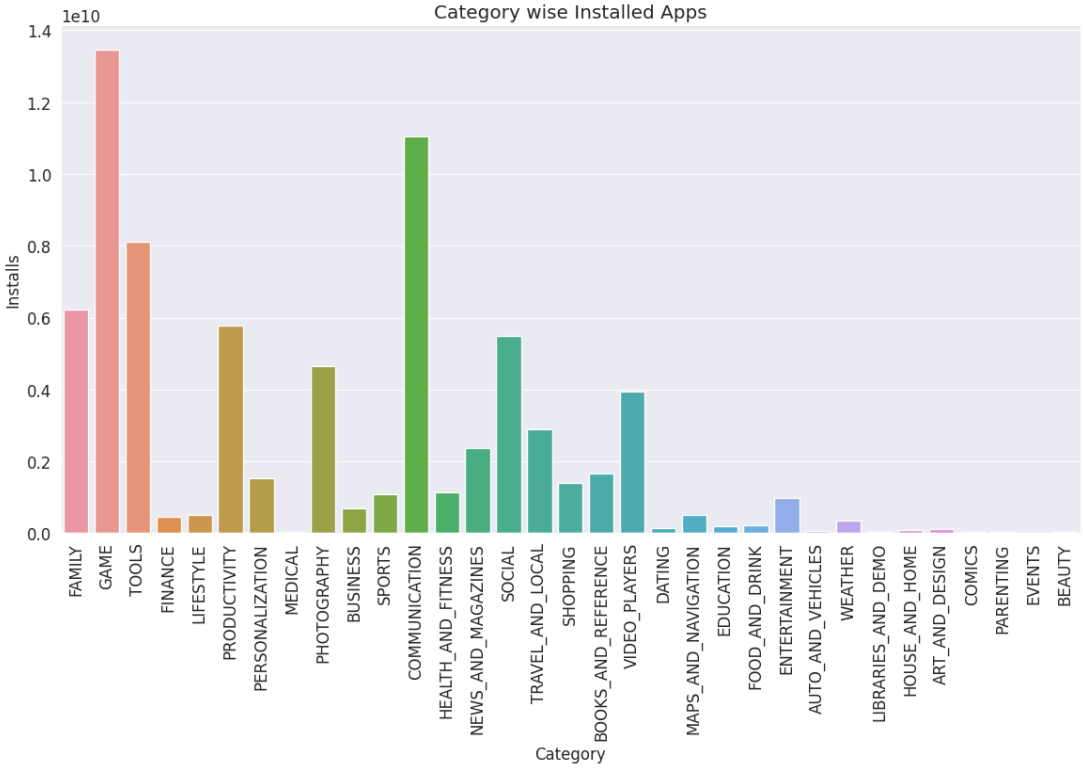



### What are the most popular Genres among the top 20 Genres?



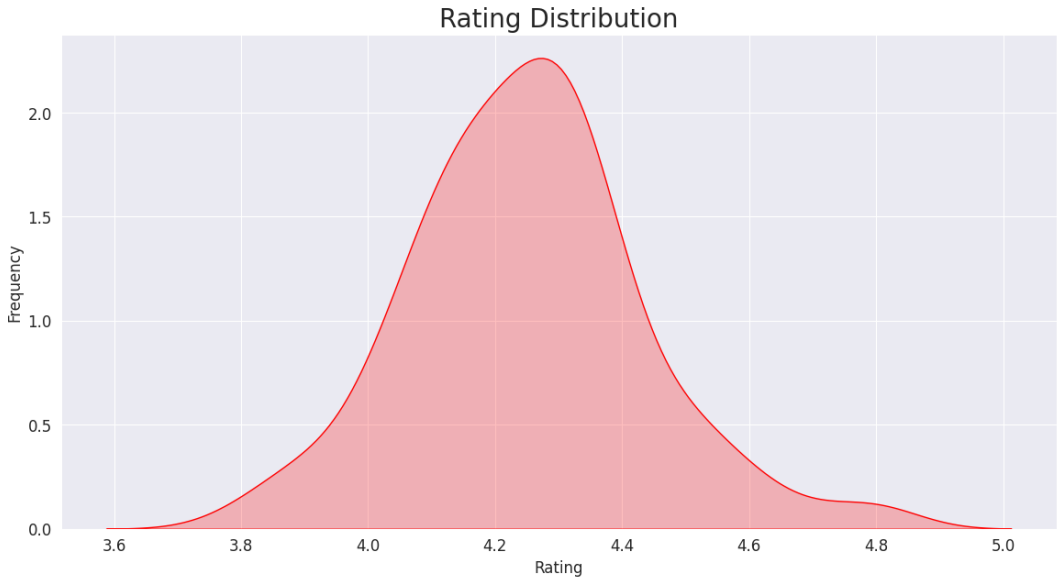
**As the following two graphs demonstrate: Although the majority of apps in the Google Play Store fall under the Tools, Entertainment, and Education categories, the situation is not the same in terms of installation and demand in the market. Communication, Tools, and Productivity are the genres with the most apps installed.**



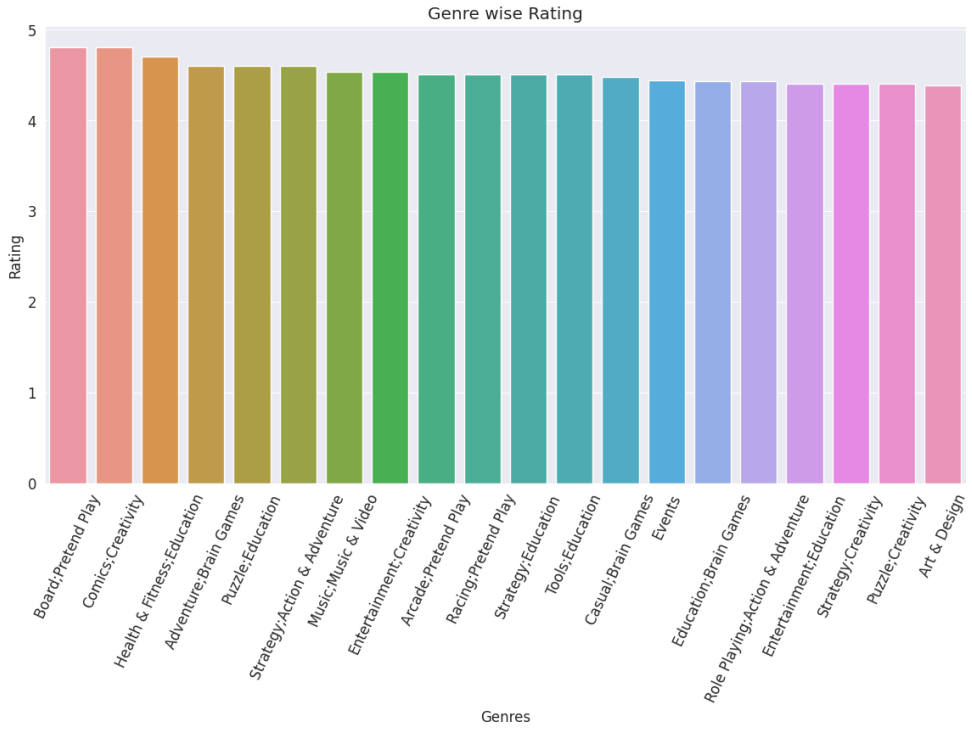



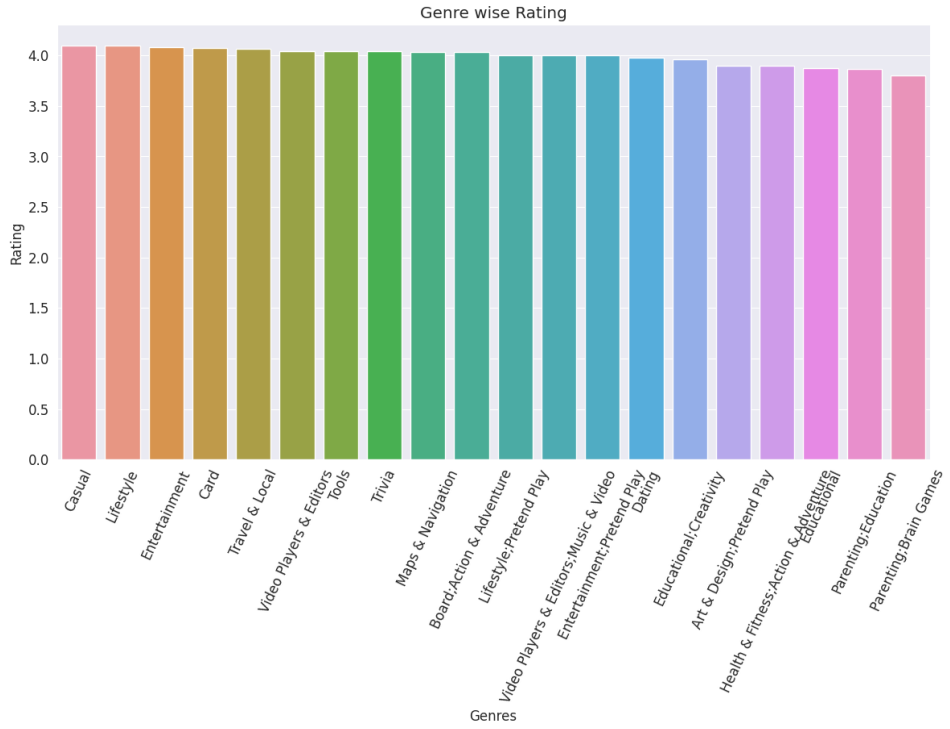
**As may be seen from the two graphs above: The majority of apps in the Google Play Store are in the Family, Games, and Tools categories, however the scenario is not the same in terms of installation and demand in the market. Games, Communication, and Tools have the most installed apps.**

### Rating Distribution

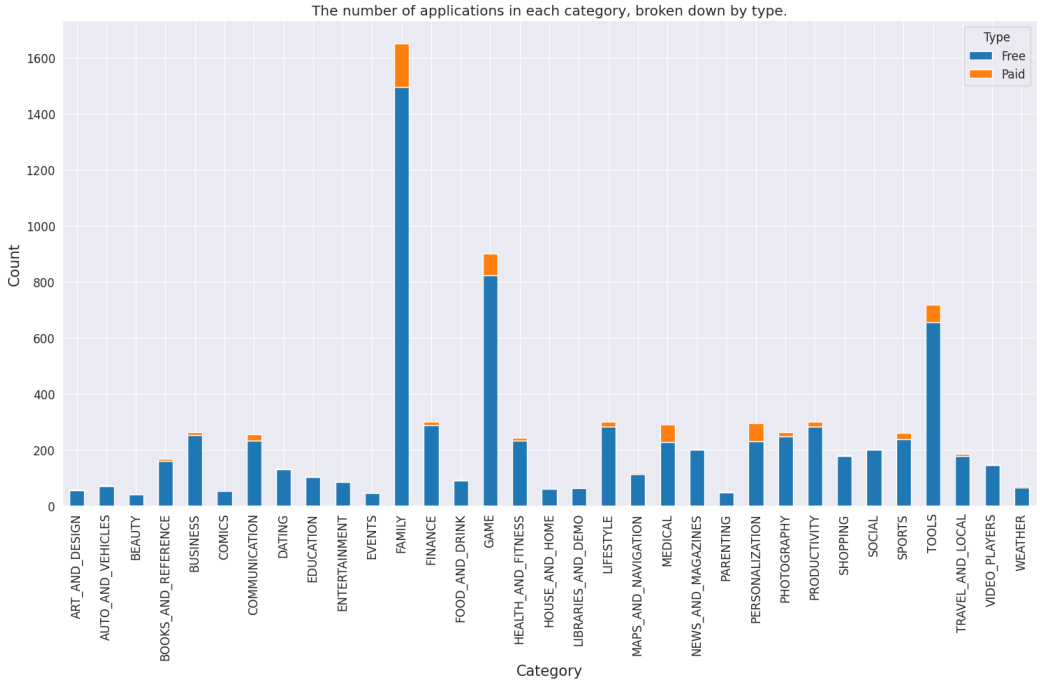



### What are the Highest and Lowest Performing Genres?



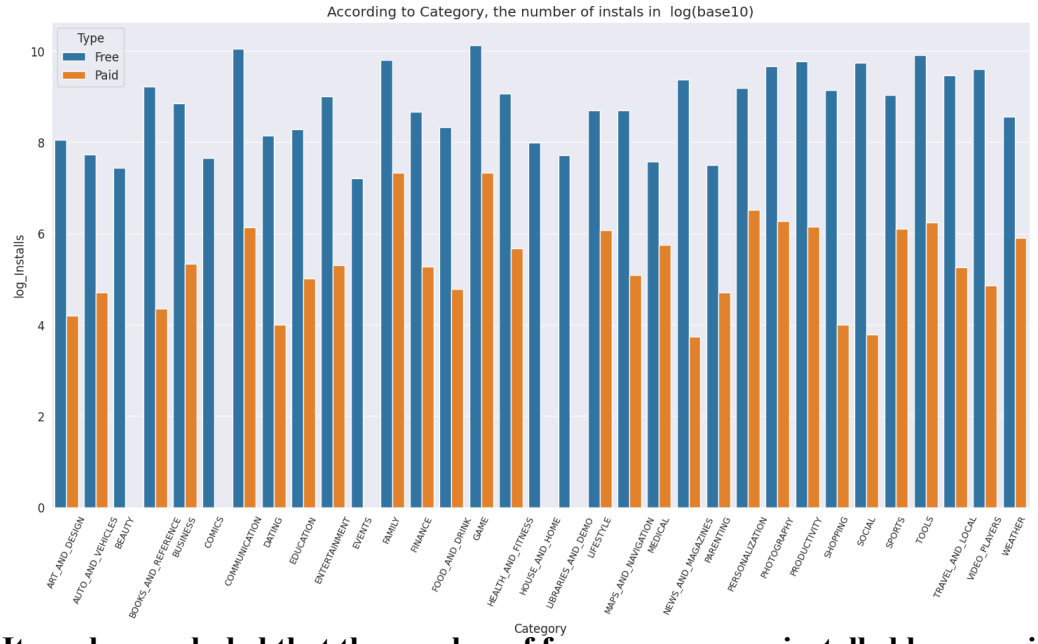



## What are the count of applications in each category differentiated by their type?



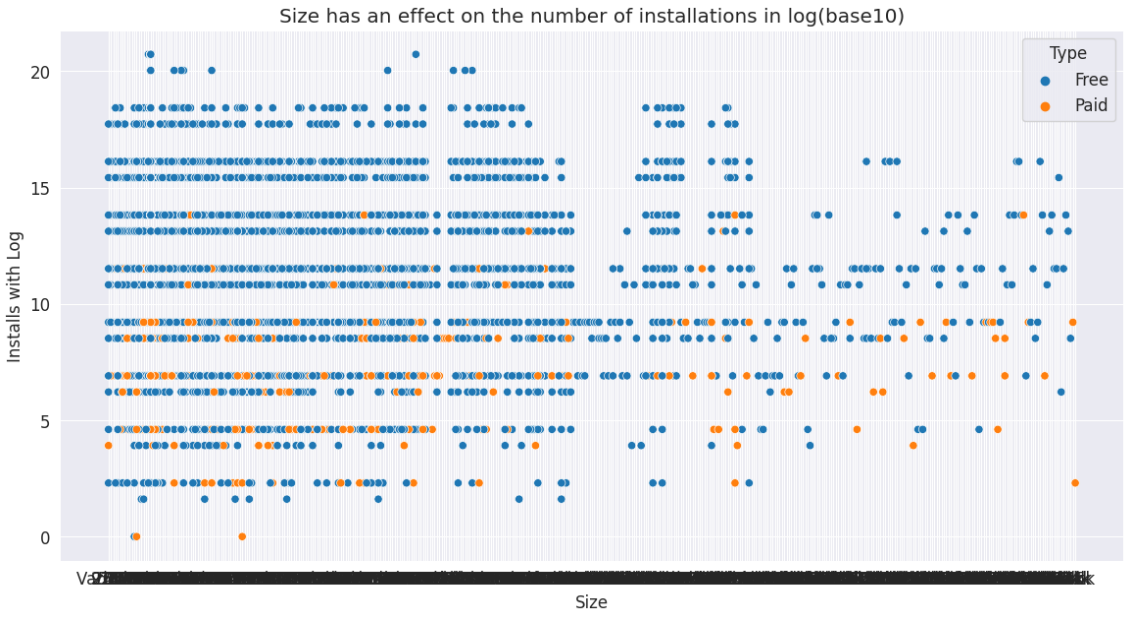
**It appears that some app categories have more free apps to download than others. The majority of apps in the Family, Food & Drink, Tools, and Social categories in our sample were free to download. At the same time, the categories of Family, Sports, Tools, and Medical had the most paid apps accessible for download.**

**How many apps were installed according to its type?**

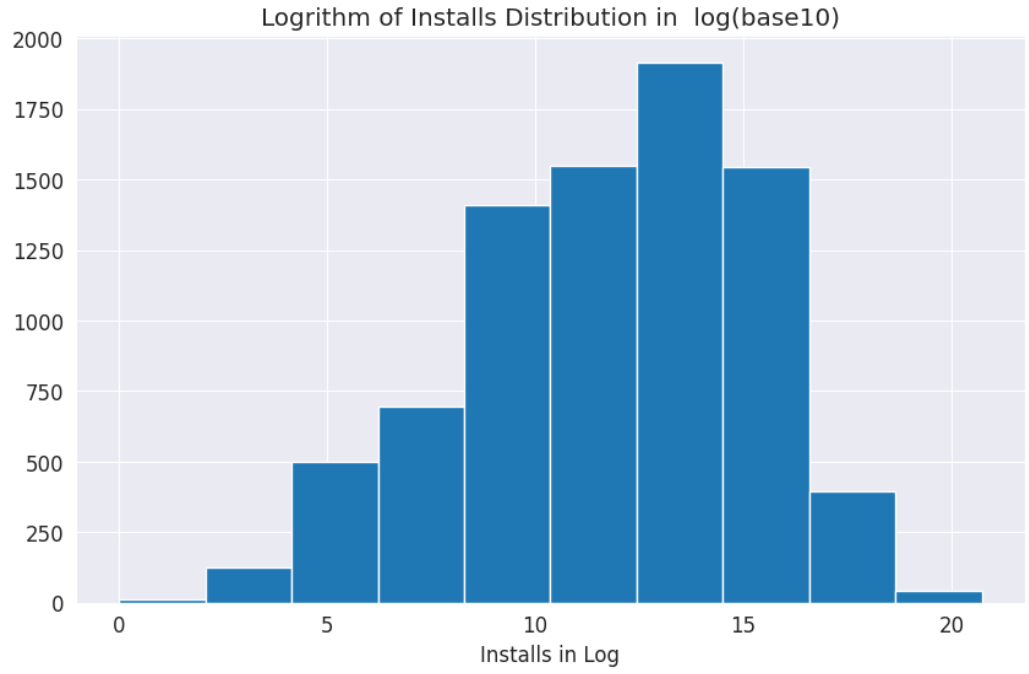


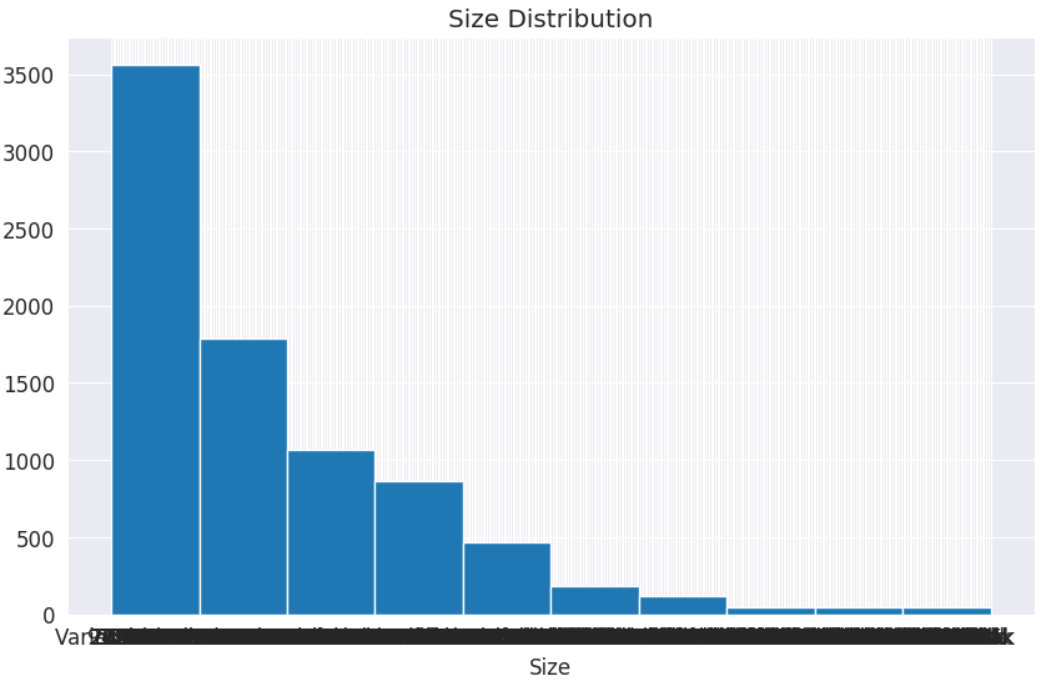

**It can be concluded that the number of free programmes installed by users is more than the number of paid applications.**

**The below-mentioned graphic shows how size might affect the number of installations. Users are less likely to install bulky programmes.**

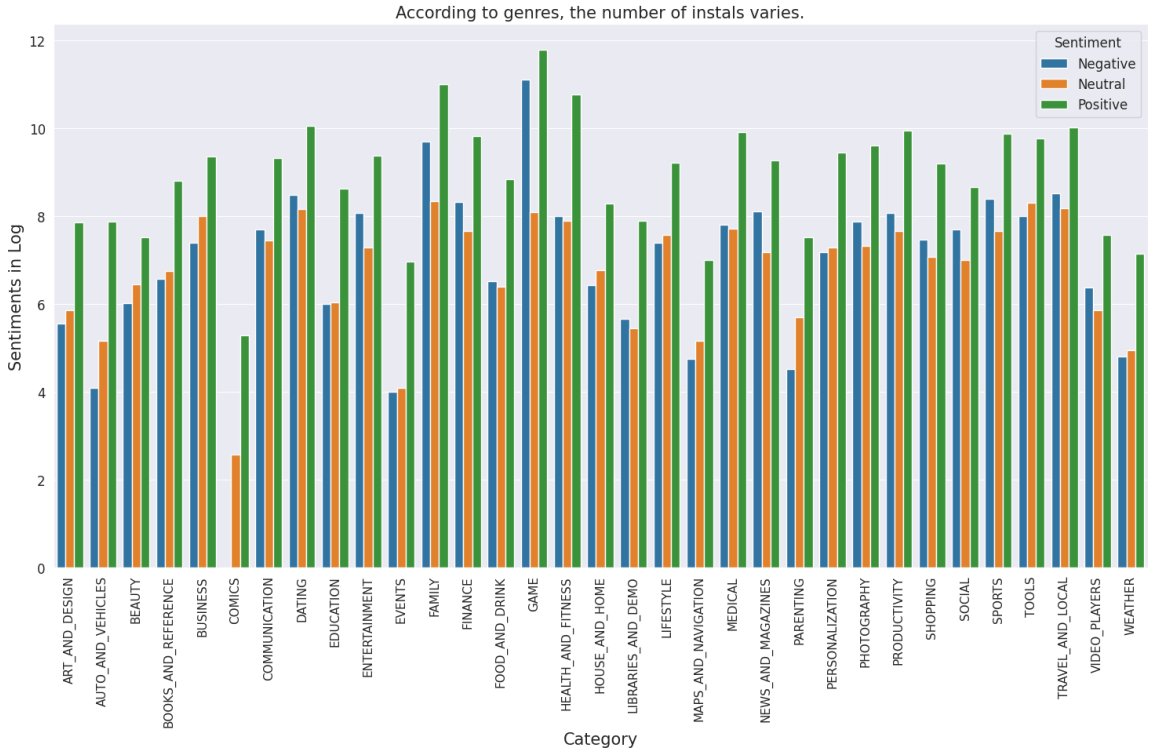


**Histogram for various attributes**

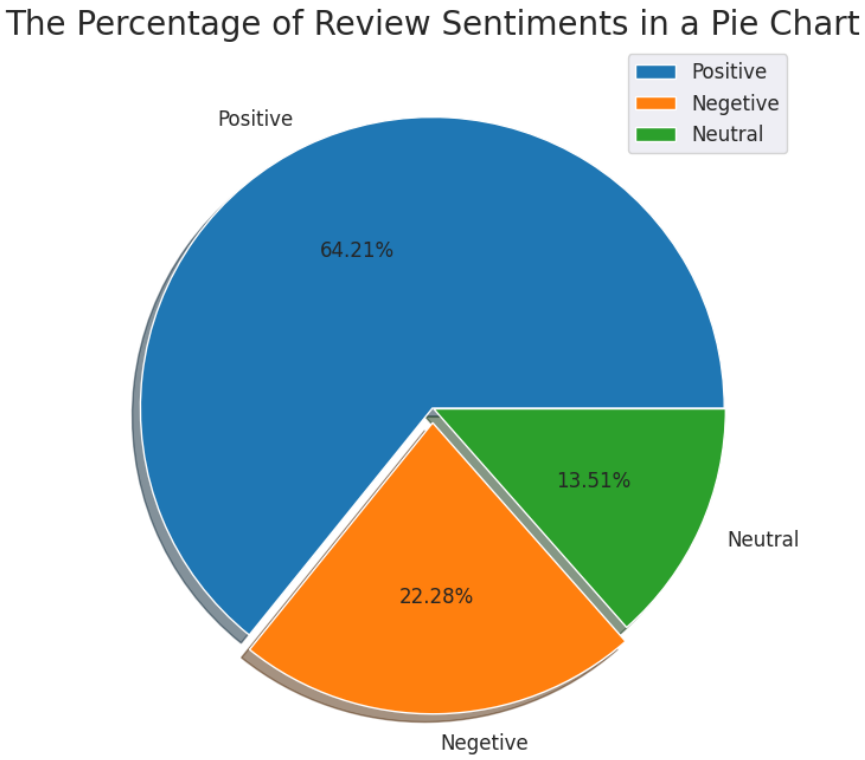


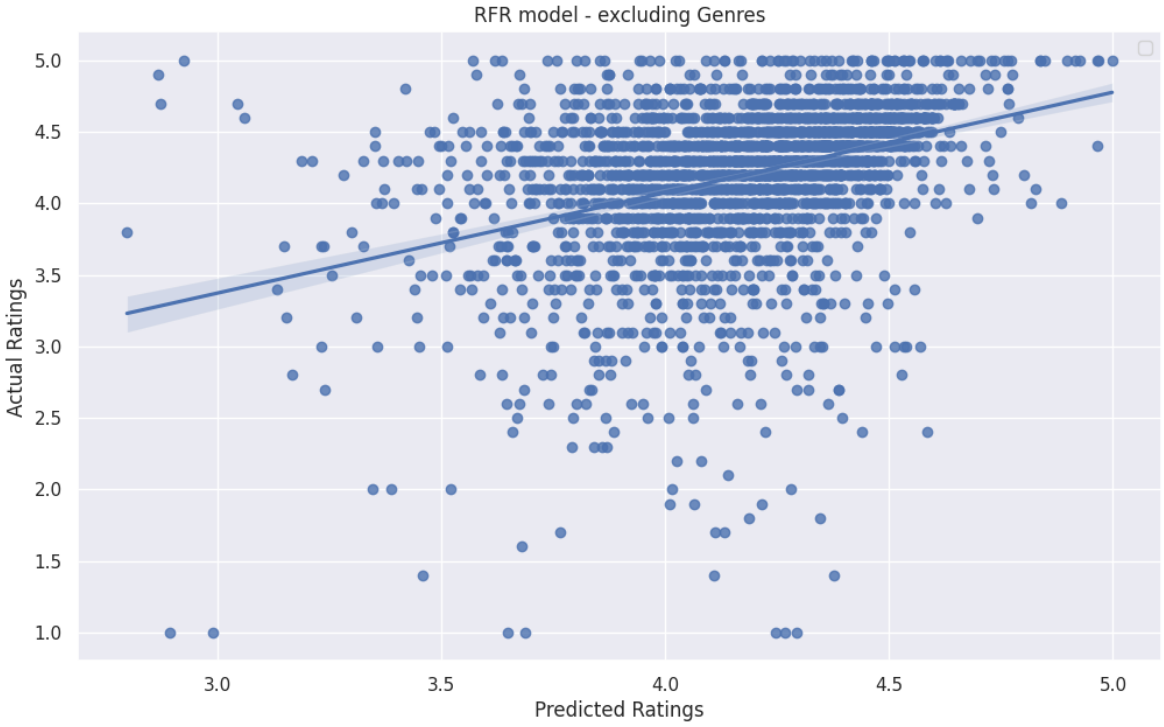
## What are the Number of Installs according to the Genres?




## Sentiments of Reviews made by the People

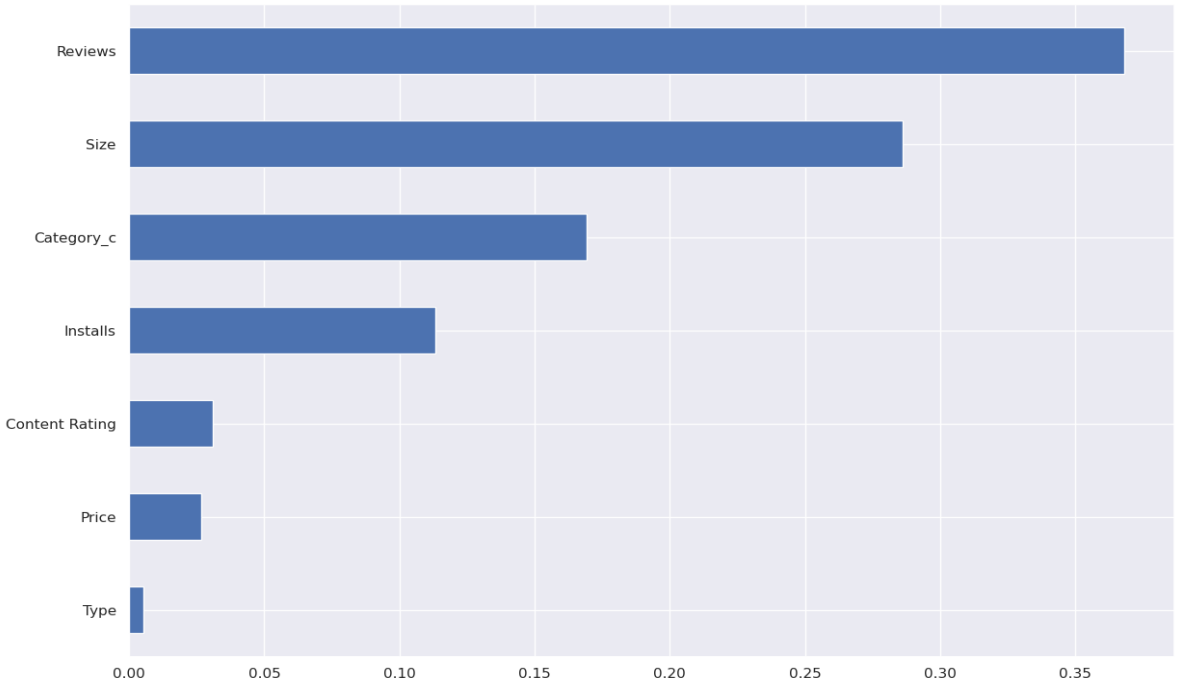


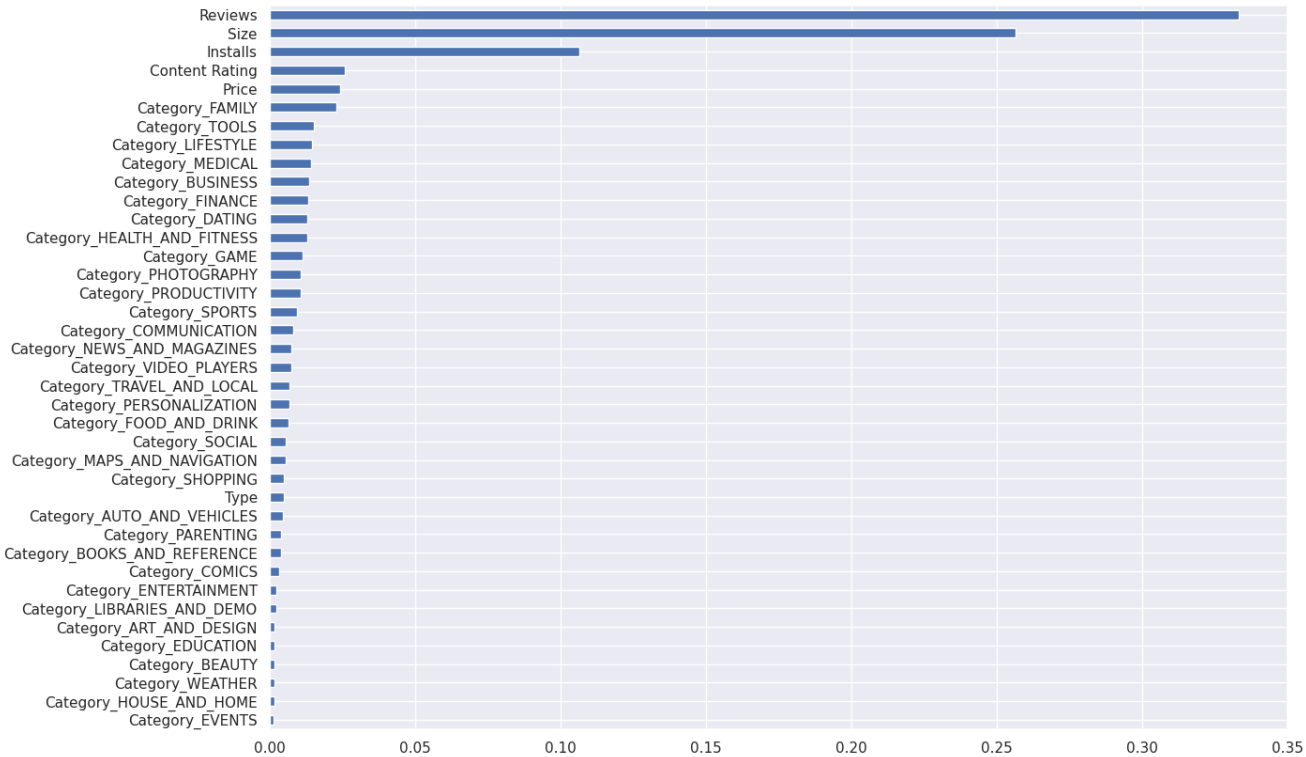
**Random Forest Regression Results**



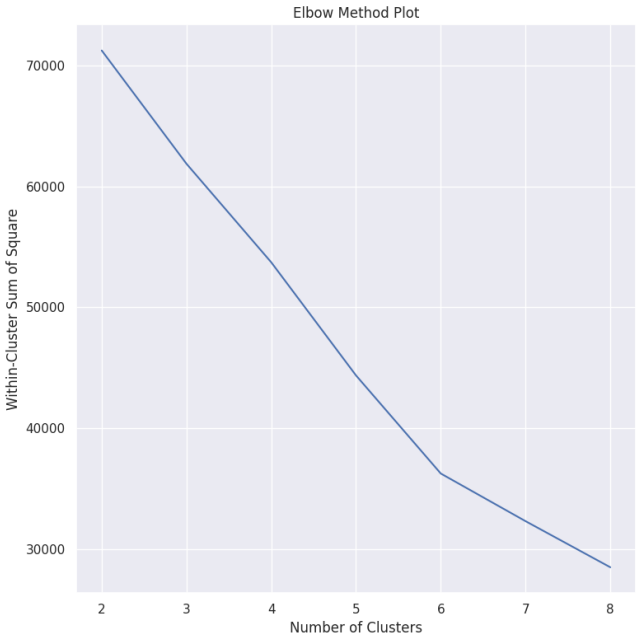

```
Evaluationmatrix(y_test, Results3)

Mean Squared Error: 0.2406593844651195
Mean absolute Error: 0.3233065586419752
Mean squared Log Error: 0.012135782454623937
```

## The variables that are affecting the Rating of the Applications most

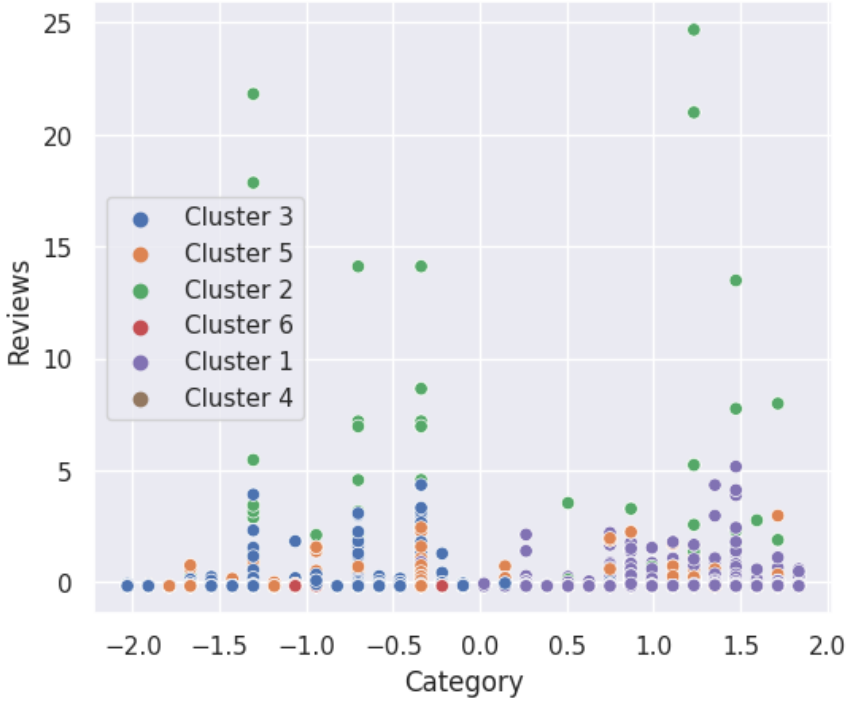


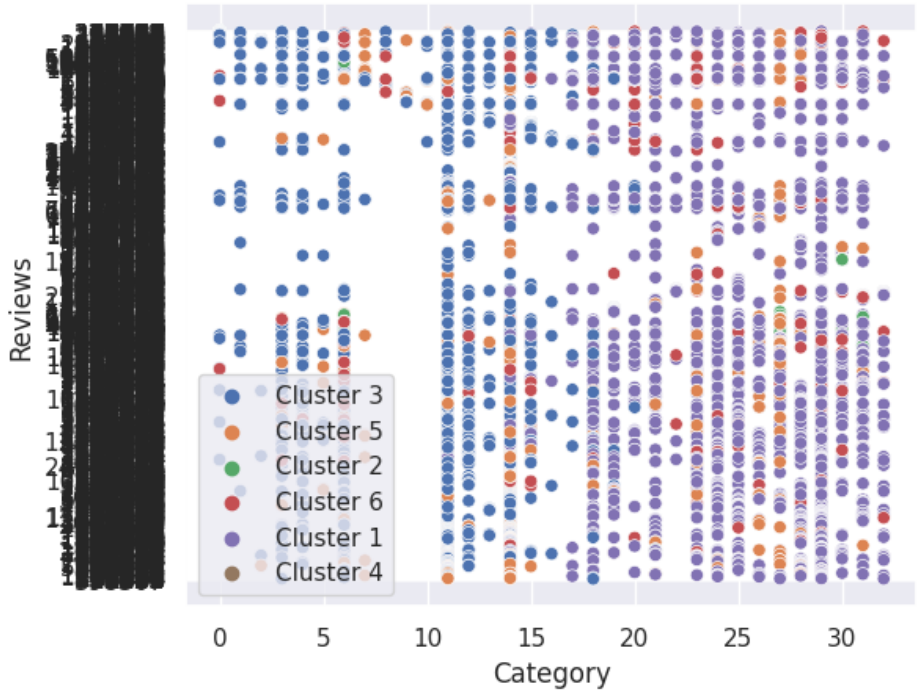
**K-Means Clustering Results**



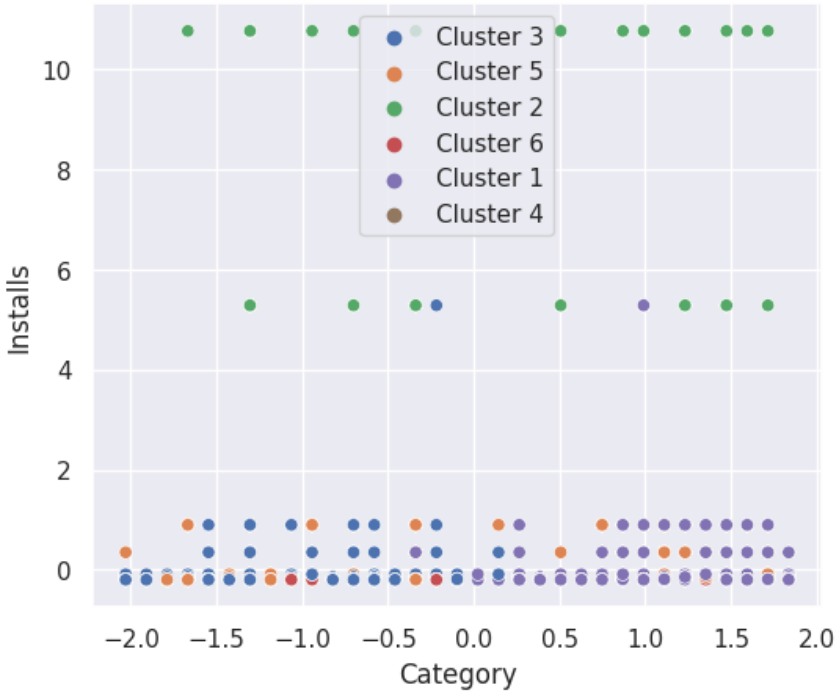

The Elbow Method has resulted in the optimal ‘k’ value of 6.



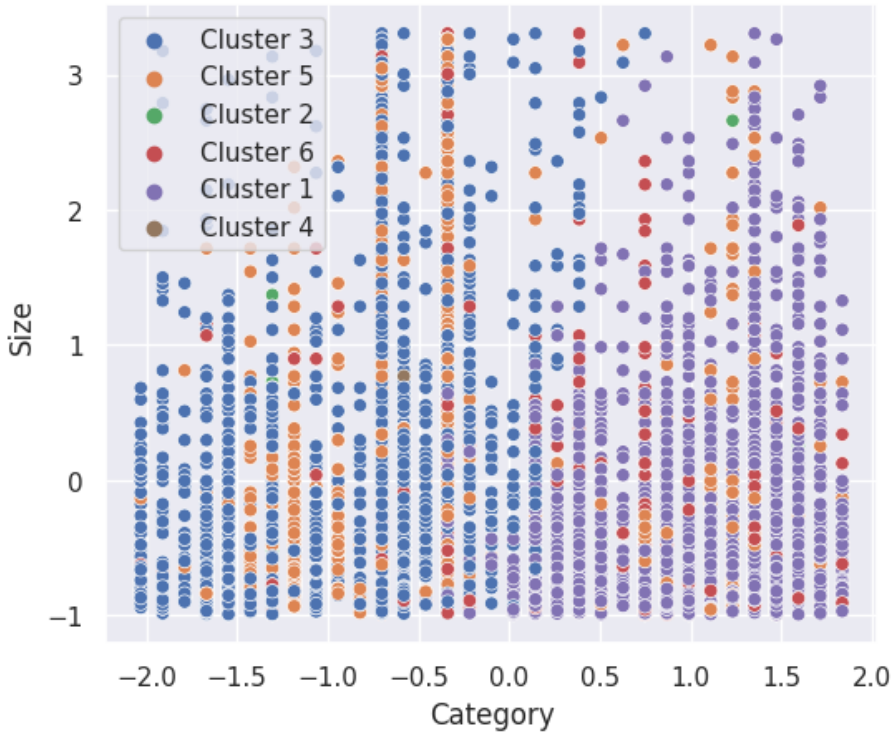

The standardized values gave a better version of clustering with Reviews being the highest for Cluster 2 Applications.

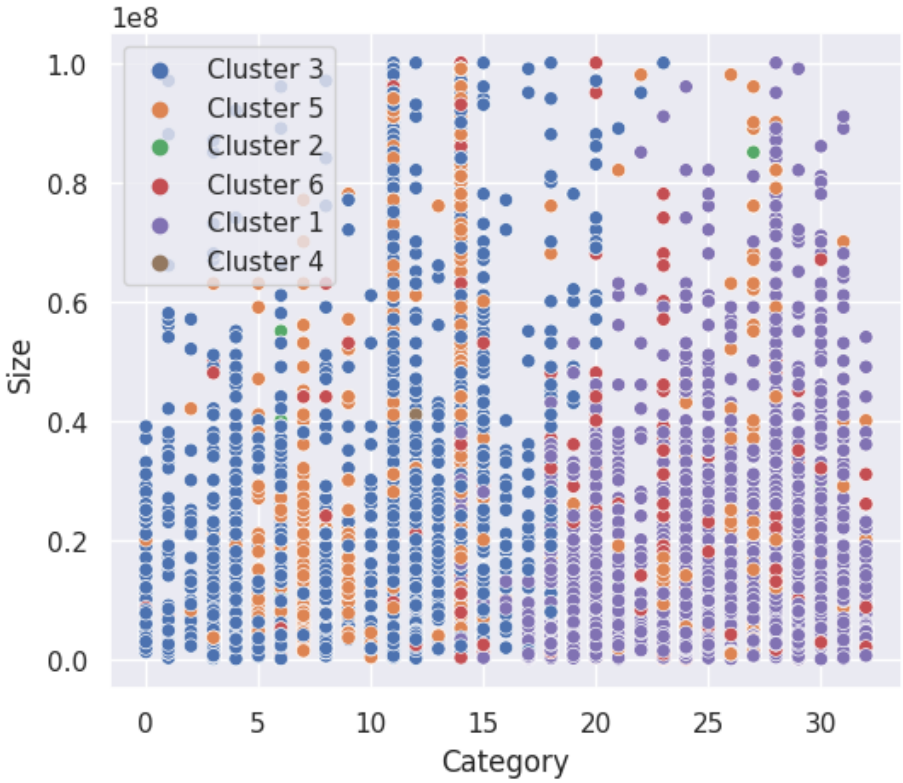


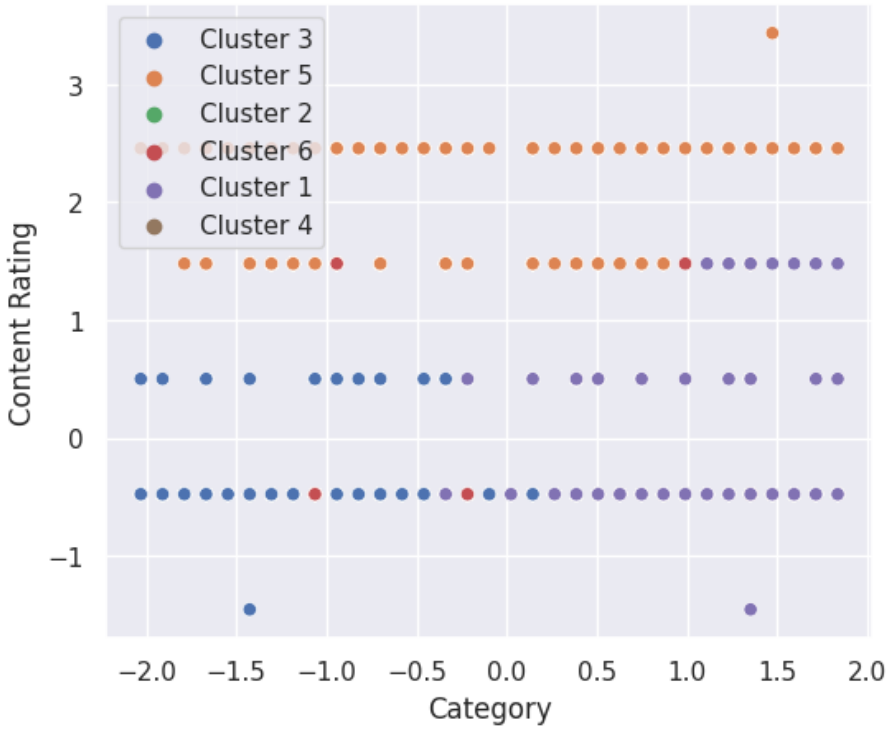

The non-standardized dataset clusters gave a very congested view of Clustering.

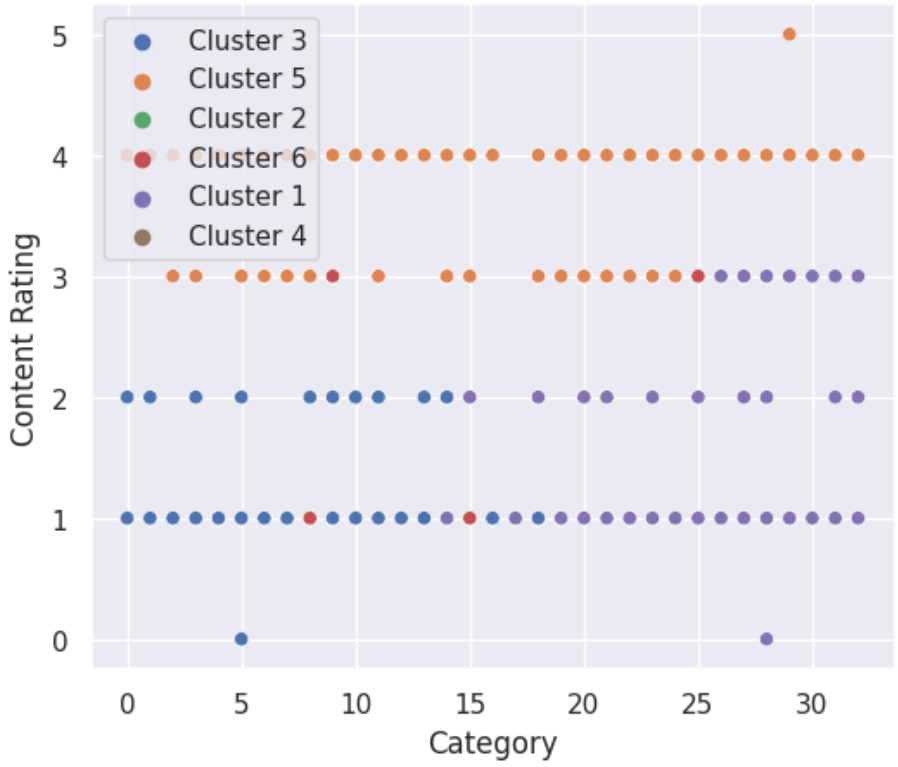


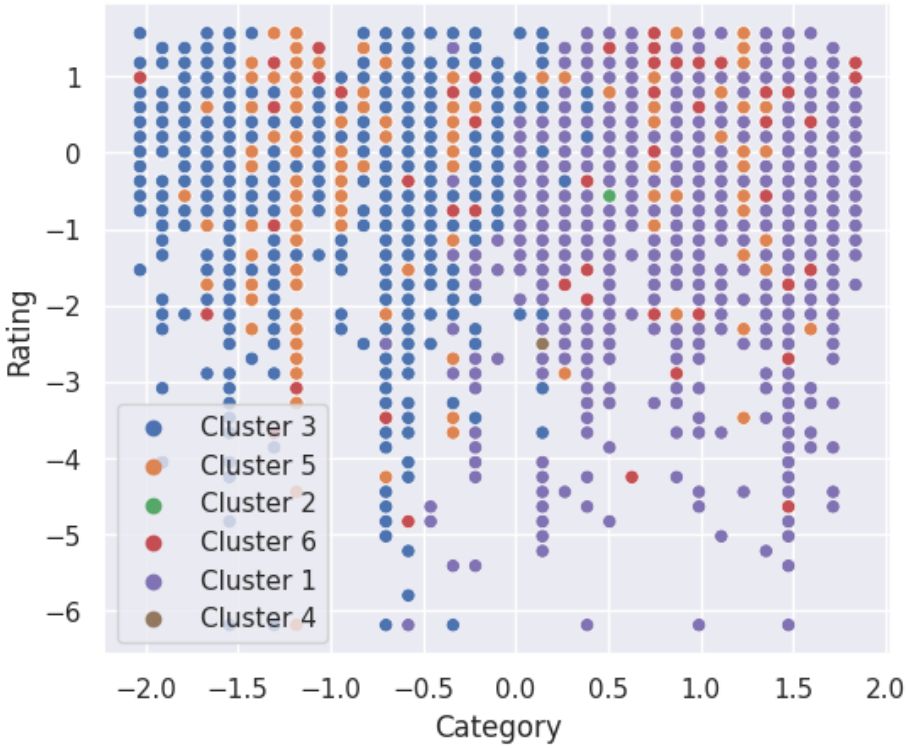

1e9
1.0
0.8
0.6
0.4
0.2
0.0
Installs
Cluster 3
Cluster 5
Cluster 2
Cluster 6
Cluster 1
Cluster 4
0
5
10
15
20
25
30
Category

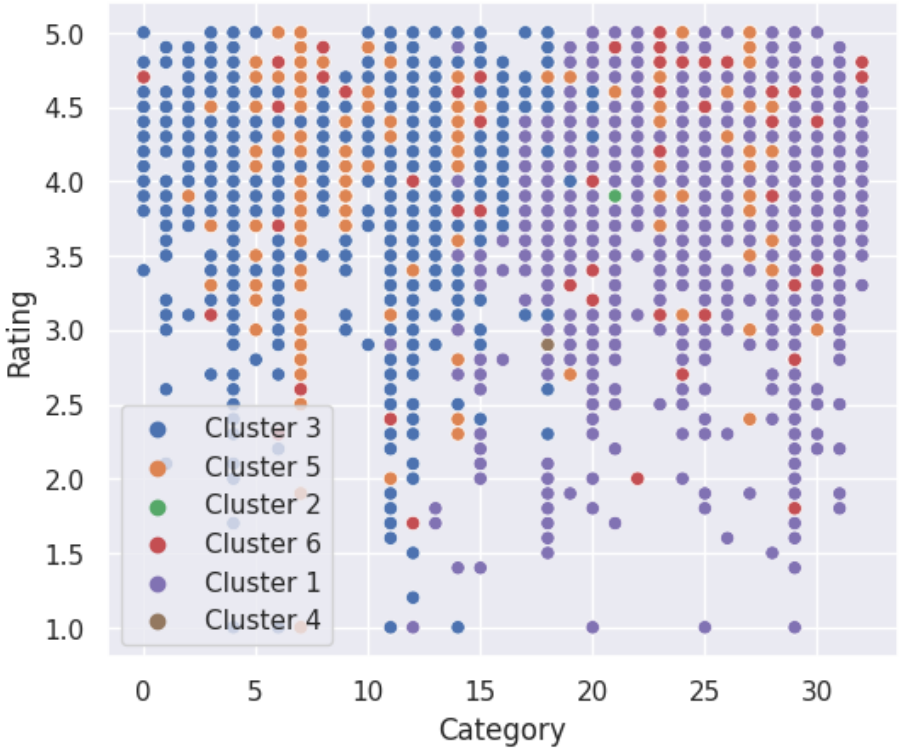

Cluster 3
Cluster 5
Cluster 2
Cluster 6
Cluster 1
Cluster 4
3
2
1
0
−1
Size
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Category


1e8
1.0
0.8
0.6
0.4
0.2
0.0
Size
Cluster 3
Cluster 5
Cluster 2
Cluster 6
Cluster 1
Cluster 4
0
5
10
15
20
25
30
Category


Content Rating
3
2
1
0
−1
Cluster 3
Cluster 5
Cluster 2
Cluster 6
Cluster 1
Cluster 4
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Category


Cluster 3
Cluster 5
Cluster 2
Cluster 6
Cluster 1
Cluster 4
Content Rating
Category
0
1
2
3
4
5
0
5
10
15
20
25
30


Cluster 3
Cluster 5
Cluster 2
Cluster 6
Cluster 1
Cluster 4
Rating
Category
1
0
−1
−2
−3
−4
−5
−6
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Rating
Category
Cluster 3
Cluster 5
Cluster 2
Cluster 6
Cluster 1
Cluster 4
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
5
10
15
20
25
30

## 7. Conclusion, Limitations and Scope for future Work

The analysis of Google Play Store applications revealed interesting insights regarding the user ratings and app characteristics. Exploratory data analysis allowed us to understand the distribution of app ratings and the most popular app categories. Random Forest Regression was employed to predict app ratings with a very low MSE of 0.24 implying very good accuracy and precision, which could be useful for developers in enhancing app quality. Furthermore, K Means clustering allowed us to segment the apps based on their features, which could aid marketers in designing targeted advertising campaigns. Overall, the study provides a comprehensive understanding of the dynamics of the Google Play Store applications.

The study has some limitations that must be considered while interpreting the results. Firstly, the analysis was based on the data available on the Google Play Store, which may not be entirely representative of the entire mobile app market. Secondly, the study focused solely on the relationship between app ratings and app characteristics, ignoring other important factors such as user demographics and usage patterns. Lastly, the study used a limited number of algorithms for analysis, and future research could explore other machine learning algorithms to uncover more patterns and relationships.

More sophisticated machine learning algorithms such as deep learning and neural networks could be employed to model app characteristics and ratings more accurately. Lastly, future research could expand the scope of the study to include other app stores such as the Apple App Store, to provide a comparative analysis of the mobile app market.

## 8. References


[1] P. B. Prakash Reddy and R. Nallabolu, "Machine learning based Descriptive Statistical Analysis on Google Play Store Mobile Applications," 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2020, pp. 647-655, Doi: 10.1109/ICIRCA48905.2020.9183271.

[2] R. M. Amir Latif, M. Talha Abdullah, S. U. Aslam Shah, M. Farhan, F. Ijaz and A. Karim, "Data Scraping from Google Play Store and Visualization of its Content for Analytics," 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), Sukkur, Pakistan, 2019, pp. 1-8, Doi: 10.1109/ICOMET.2019.8673523.

[3] Venkatakrishnan, S., Kaushik, A., Verma, J.K. (2020). Sentiment Analysis on Google Play Store Data Using Deep Learning. In: Johri, P., Verma, J., Paul, S. (eds) Applications

of Machine Learning. Algorithms for Intelligent Systems. Springer, Singapore. https://doi.org/10.1007/978-981-15-3357-0_2

[4] S. Shashank and B. Naidu, “Google Play Store Apps- Data Analysis and Ratings Prediction,” Dec. 2020. https://mail.irjet.net/archives/V7/i12/IRJET-V7I1248.pdf

[5] I. Journal, “IRJET- Play Store App Analysis,” IRJET, Jan. 2021. Available: https://www.academia.edu/52123932/IRJET_Play_Store_App_Analysis

[6] Maredia, Rimsha. (2020). Analysis of Google Play Store Data set and predict the popularity of an app on Google Play Store.

[7] Bashir, G. M. M., Hossen, M. S., Karmoker, D., & Kamal, M. J. (2019, December). Android apps success prediction before uploading on google play store. In 2019 International Conference on Sustainable Technologies for Industry 4.0 (STI) (pp. 1-6). IEEE.

[8] MarediaGao, S., Liu, L., Liu, Y., Liu, H., & Wang, Y. (2021). API recommendation for the development of Android App features based on the knowledge mined from App stores. Science of Computer Programming, 202, 102556.

[9] Song, G., Hu, L., & Liu, Y. (2020). Common but innovative: learning features from apps in different domains. IEEE Access, 8, 186904-186918.

[10] Lengkong, O., & Maringka, R. (2020, October). Apps Rating Classification on Play Store Using Gradient Boost Algorithm. In 2020 2nd International Conference on Cybernetics and Intelligent System (ICORIS) (pp. 1-5). IEEE.

[11] Liu, Y., Liu, L., Liu, H., Wang, X., & Yang, H. (2019). Mining domain knowledge from app descriptions. Journal of Systems and Software, 133, 126-144.

**APPENDIX I :**

**https://colab.research.google.com/drive/1uflSkJo0TCua2q55mz_4bZqHP1TdUhIa?usp=sharing**