The University of Chicago Booth School of Business

BUSN 41201 - Big Data - Final Project

PROJECT TITLE

26 May 2024

Yi Cao, Shri Lekkala, Ningxin Zhang

Contents

Note: The full the code used in all the questions can be found in the appendix.

1. Executive Summary

[REDO AFTER WE COMPLETE THE REPORT]

In this report, we present a comprehensive analysis of the "Google Play Store dataset" to gain insights into the characteristics and success factors of mobile applications. By examining various aspects related to app details, including categories, ratings, reviews, sizes, installations, and pricing, we aim to identify patterns and trends that contribute to an app's success on the Google Play Store.

We begin by exploring the general statistics of apps, focusing on the distribution of app categories, ratings, and reviews. This provides a foundational understanding of the data and highlights key areas of interest. Next, we delve into specific analyses to understand the relationship between app size, installs, and pricing, exploring how these factors influence an app's popularity and user engagement.

Our study also includes a sentiment analysis of user reviews, examining the polarity and subjectivity of feedback to understand how user sentiments correlate with app ratings and success. Additionally, we develop predictive models to forecast app ratings based on various features, and we investigate potential causal relationships between app characteristics and their performance metrics.

By leveraging data visualization, feature engineering, and predictive modeling techniques, we aim to provide actionable insights for potential app developers. These insights can help optimize app features, improve user satisfaction, and ultimately enhance the app's visibility and success on the Google Play Store.

2. Introduction

[WE CAN CHANGE THE QUESTIONS, THESE ARE JUST EXAMPLES]

In this paper, we aim to analyze the Google Play Store dataset to gain a comprehensive understanding of the factors that contribute to the success of mobile applications. The dataset includes details of apps such as categories, ratings, reviews, sizes, installations, and pricing, as well as user reviews with sentiment analysis. Our objective is to uncover patterns and trends that can help app developers optimize their offerings and improve user satisfaction.

The Google Play Store dataset, available on Kaggle, consists of two files: `googleplaystore.csv`, which contains detailed information about the apps, and `googleplaystore_user_reviews.csv`, which includes user reviews and sentiment data.

Our analysis will focus on the following research questions:

- **What factors affect the number of installs an app receives?** Specifically, what is the relationship between app size, type (free or paid), price, and the number of installs?
- **What are the key features that influence an app's rating?** How do factors like category, price, and number of reviews contribute to the overall rating of an app?
- **How does user sentiment in reviews correlate with app ratings?**
  Can sentiment analysis of user reviews provide additional insights into user satisfaction and app performance?

We will begin by loading and cleaning the dataset, followed by a thorough exploratory data analysis to uncover initial insights. Subsequently, we will perform detailed analyses to address our research questions, culminating in the development of predictive models and the identification of causal relationships. We will end by making concluding remarks from our research.

3. Dataset

a) Understanding the data

For `googleplaystore.csv` there are the following columns:

- App: Application Name
- Category: Category Type (e.g. Family, Game, Art)
- Rating: User rating review
- Reviews: Number of reviews
- Size: Download size of application
- Installs: Number of user downloads 0.. - Type: Paid or Free
- Price: Price of App
- Content.Rating: Age group that app is targeted at (E.g. Everyone, Teen, Child)
- Genres: Other categories the app belongs to, other than the main category
- Last.Updated: Date when app was last updated
- Current.Ver: Current app version available
- Android.Ver: Minimum required Android version for app

There are a total of 10841 rows (applications).

For `googleplaystore_user_reviews.csv` there are the following columns:

- App: Application Name
- Translated_Review: User review, translated to English
- Sentiment: Positive / Negative / Neutral (Preprocessed)
- Sentiment_Polarity: Sentiment polarity score (Preprocessed)
- Sentiment_Subjectivity: Sentiment subjectivity score (Preprocessed)

This dataset contains the first 100 'most relevant' review for each app, with some prepocessing already done to add the last 3 features.

There are a total of 64295 rows (reviews).

b) Data Cleaning

```r
# Convert the variables to the appropriate data type
googleplaystore <- googleplaystore_raw |>
  mutate(
    # Transform Installs and size to numeric
    Installs = gsub("\\+", "", as.character(Installs)),
    Installs = as.numeric(gsub(",", "", Installs)),
    Size = gsub("M", "", Size),
    # Convert apps with size < 1MB to 0, and transform to numeric
    Size = ifelse(grepl("k", Size), 0, as.numeric(Size)),
    # Transform reviews to numeric
    Reviews = as.numeric(Reviews),
    # Change currency numeric
    Price = as.numeric(gsub("\\$", "", as.character(Price))),
    # Convert Last.Updated to date
    Last.Updated = mdy(Last.Updated),
    # Change version number to 1 decimal, and add NAs where appropriate
    Android.Ver = gsub("Varies with device", NaN, Android.Ver),
    Android.Ver = as.numeric(substr(Android.Ver, start = 1, stop = 3)),
    Current.Ver = gsub("Varies with device", NaN, Current.Ver),
    Current.Ver = as.numeric(substr(Current.Ver, start = 1, stop = 3)),
  ) |>
  # Remove apps with Type 0 or NA
  filter(Type %in% c("Free", "Paid")) |>
  # Convert Category, Type, Content.Rating and Genres to factors
  mutate(
    App = as.factor(App),
    Category = as.factor(Category),
    Type = as.factor(Type),
    Content.Rating = as.factor(Content.Rating),
    Genres = as.factor(Genres)
  ) |>
  # Remove duplicate rows
  distinct()
```

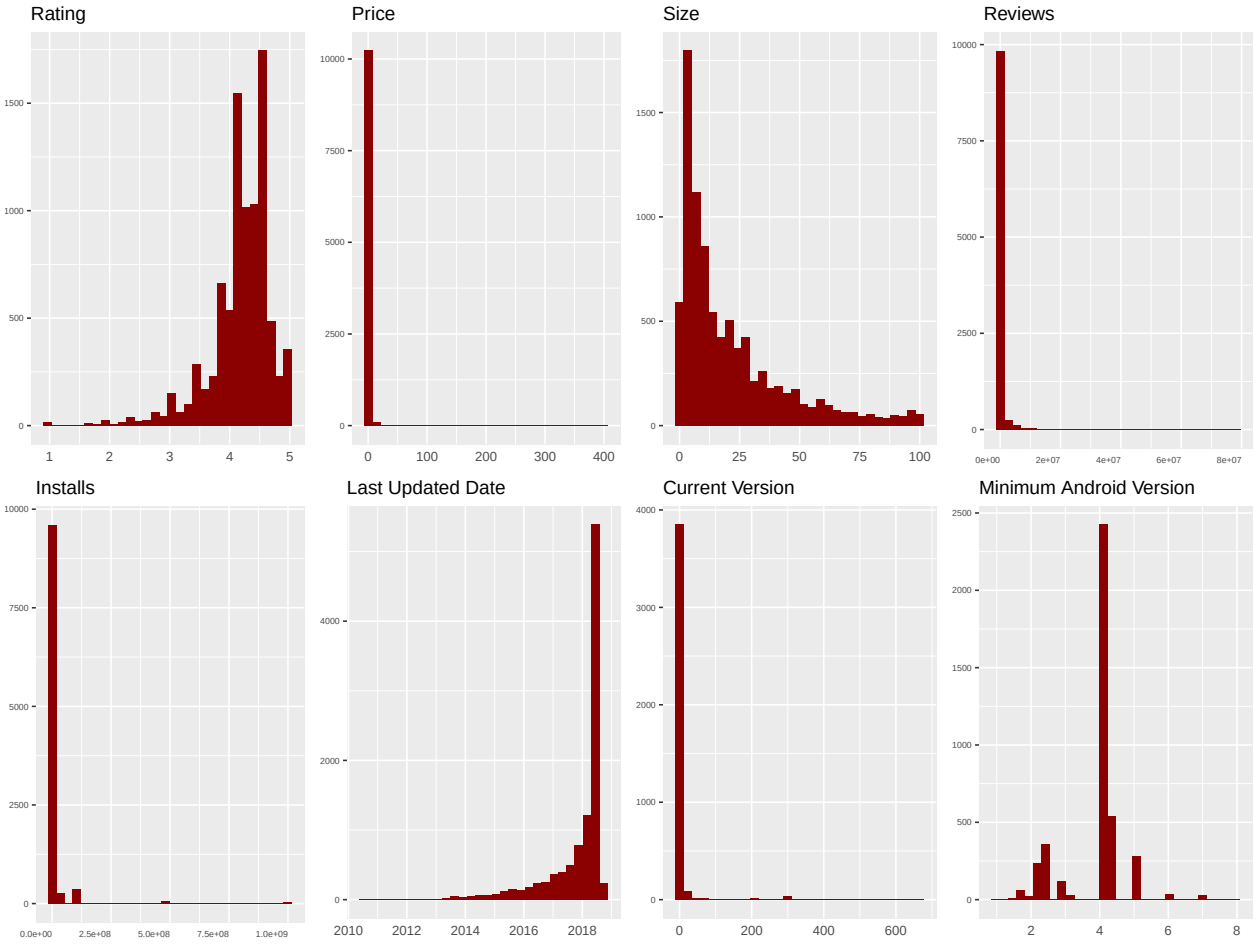
For the `googleplaystore` dataset, we first process the variables by converting columns to the appropriate datatype. For example Installs, Size, Reviews Price, and Android.Ver are converted to numerics, Last.Updated is converted to date. Then we filter out apps with Type 0 or NA, and remove duplicated rows. After this, we are left with 10356 rows.

```r
# Remove all rows with nans
googleplaystore_user_reviews <- googleplaystore_user_reviews_raw |>
  filter(Translated_Review != "nan") |>
  # Convert Sentiment to factor
  mutate(Sentiment = as.factor(Sentiment))
```

With the `googleplaystore_user_reviews` dataset, the variables were already well structured, but we noticed there were many rows with "nan"s. After filtering these out, we were left with 37432 rows.

4. Exploratory Analysis

a) Numerical Features



According to the histograms, most apps have high ratings, peaking around 4 to 5, with fewer apps rated below 3, indicating generally positive user feedback. The majority of apps have a low number of installs, while a few apps have extremely high install numbers, showing a highly skewed distribution. Since installs are highly skewed, we will perform a log transformation on it in later analysis.

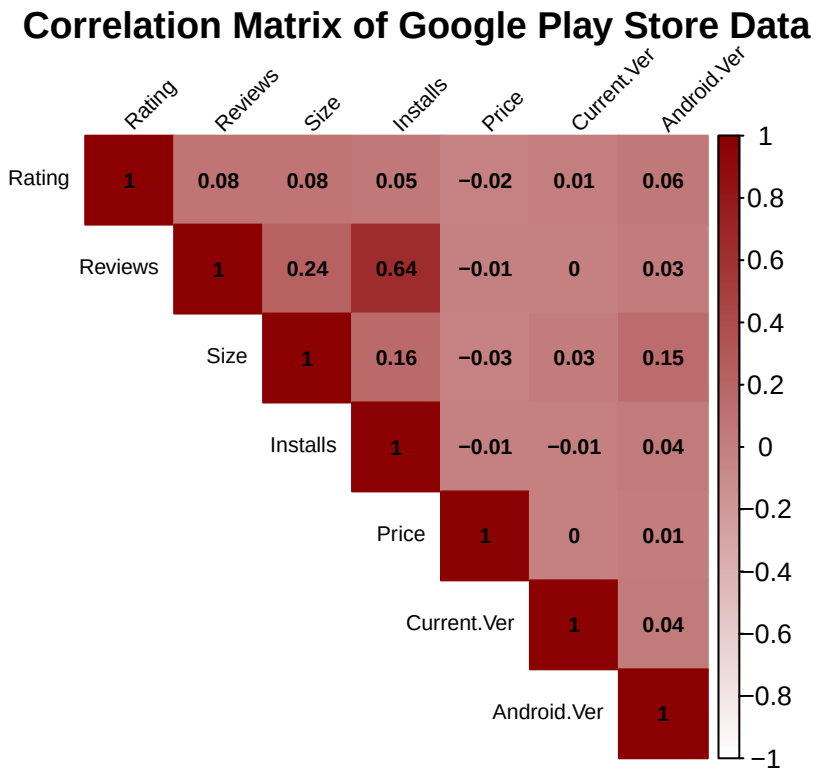
Regarding other numerical variables, the vast majority of apps are free, with the few paid apps showing a wide price range, including some very expensive ones. Most apps are small in size, with a significant drop-off as size increases. The majority are less than 25 MB, with very few exceeding 100 MB. Similarly, most apps have a low number of reviews, with a small number having extremely high reviews, indicating a skewed distribution where a few apps are very popular while many are not widely reviewed. Most apps have been updated recently, with a notable increase in updates around 2018, suggesting the dataset is current and apps are actively maintained. Most apps are on version 1 or 2, with a sharp decrease in the number of apps as

the version number increases, indicating that many apps do not undergo numerous versions.

Most apps require Android version 4 or 4.5, with fewer requiring higher versions, suggesting developers aim for compatibility with older Android versions to reach a wider audience. However, the data for these version variables is not clean and contains extreme outliers even after cleaning, making it difficult to interpret. Therefore, we might exclude these variables from later analysis.

b) Correlation Matrix

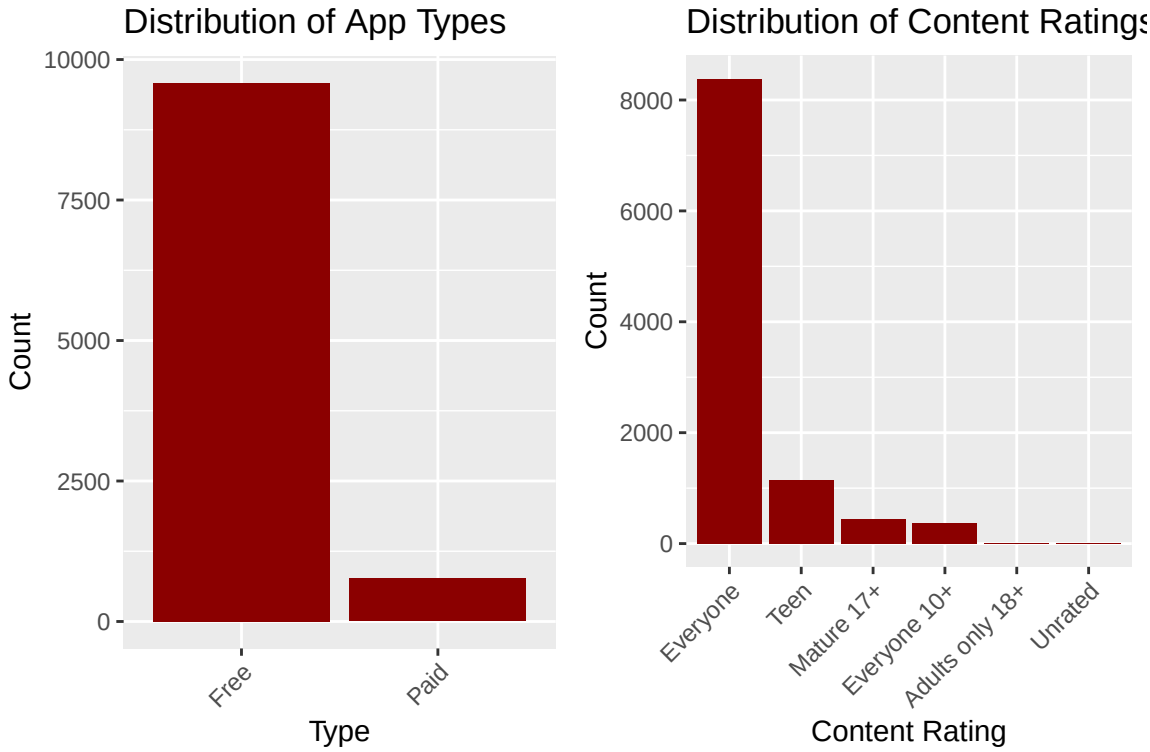
We begin by analysing the correlation matrix of all the numeric variables for googleplaystore:



This seems surprising initially as the variables appear to be fairly uncorrelated with eachother, except for the fact that "Installs" and "Reviews" which are highly correlated with a score of 0.64, which would make sense as one would expect a more popular app with a greater number of installs to also have a higher number of reviews. One surprising variable that is somewhat positively correlated with others is "Size", with small positive correlations with "Reviews" and "Installs". This might perhaps be due to the fact with apps with a larger download size are more 'complicated' and may perform more functions, and thus lead to a greater number of installations and thus reviews too.

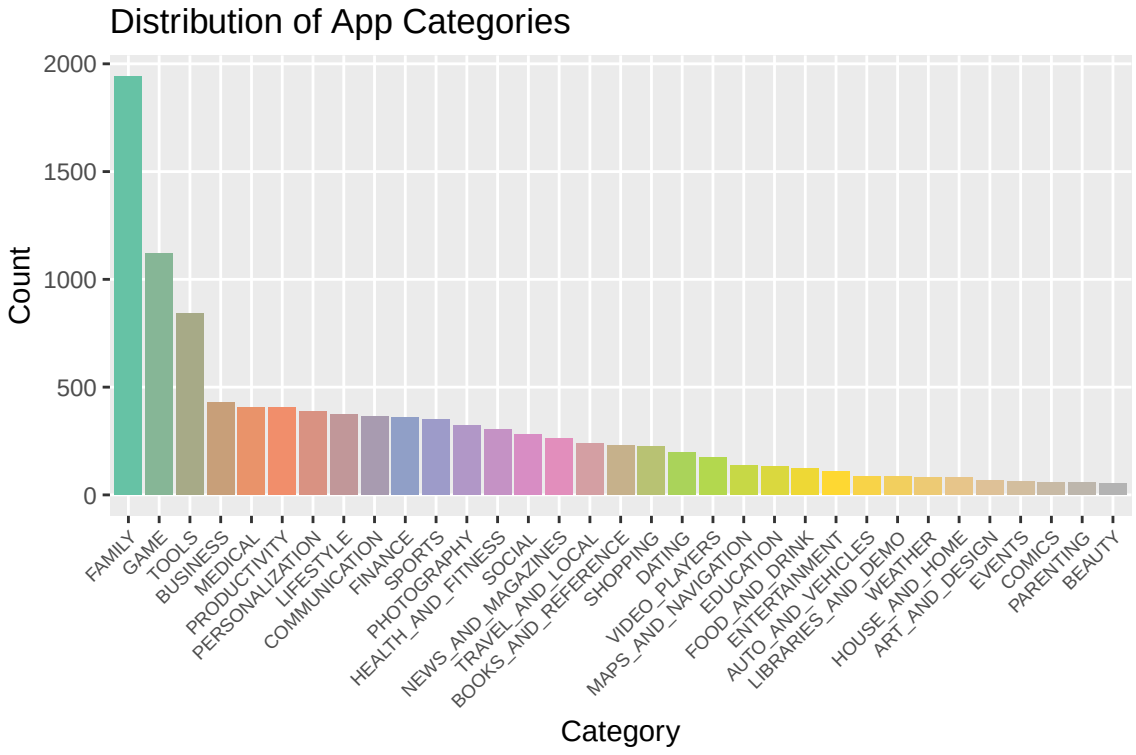
c) Categorical Features

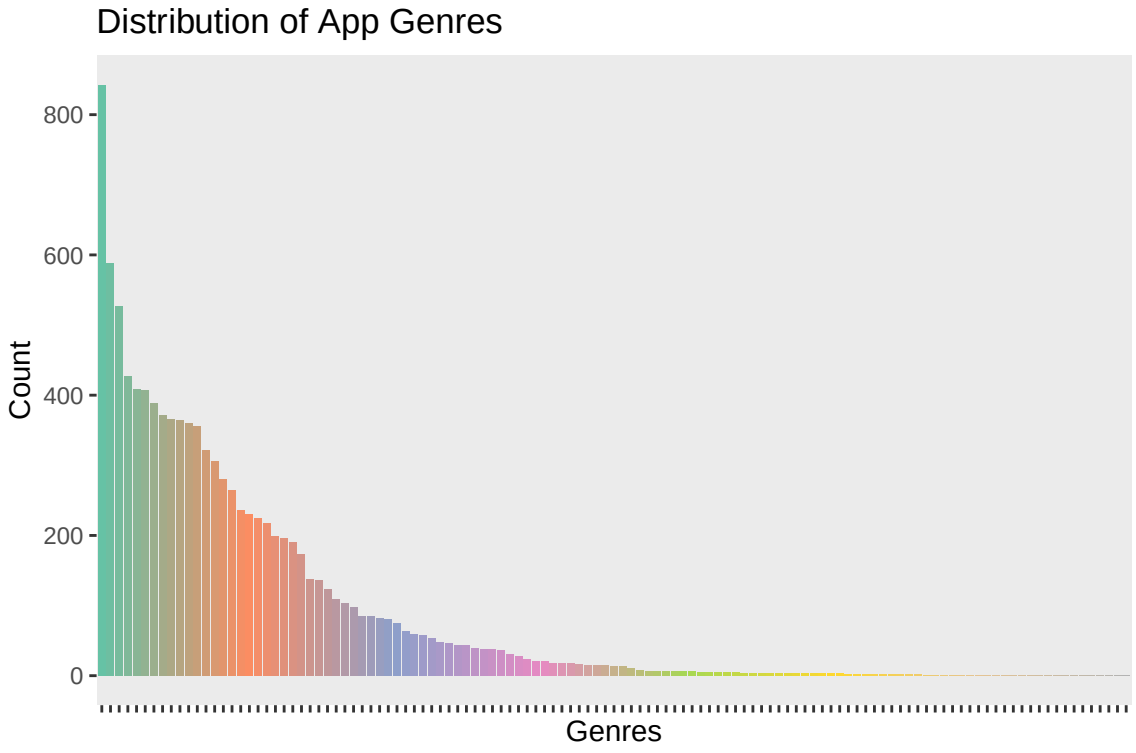
We also look at the distribution of the categorical features in our dataset:



So immediately we observe that there is a much greater proportion of free apps than paid, this aligns with the common "freemium" model where apps are free to download but may offer in-app purchases. This model also lowers the barrier to entry to users.

The content rating distribution shows that the the significant majority of apps are aimed at is "Everyone". This indicates that most apps are designed to be accessible for a general audience, which makes sense if developers want the largest possible user base for their app.





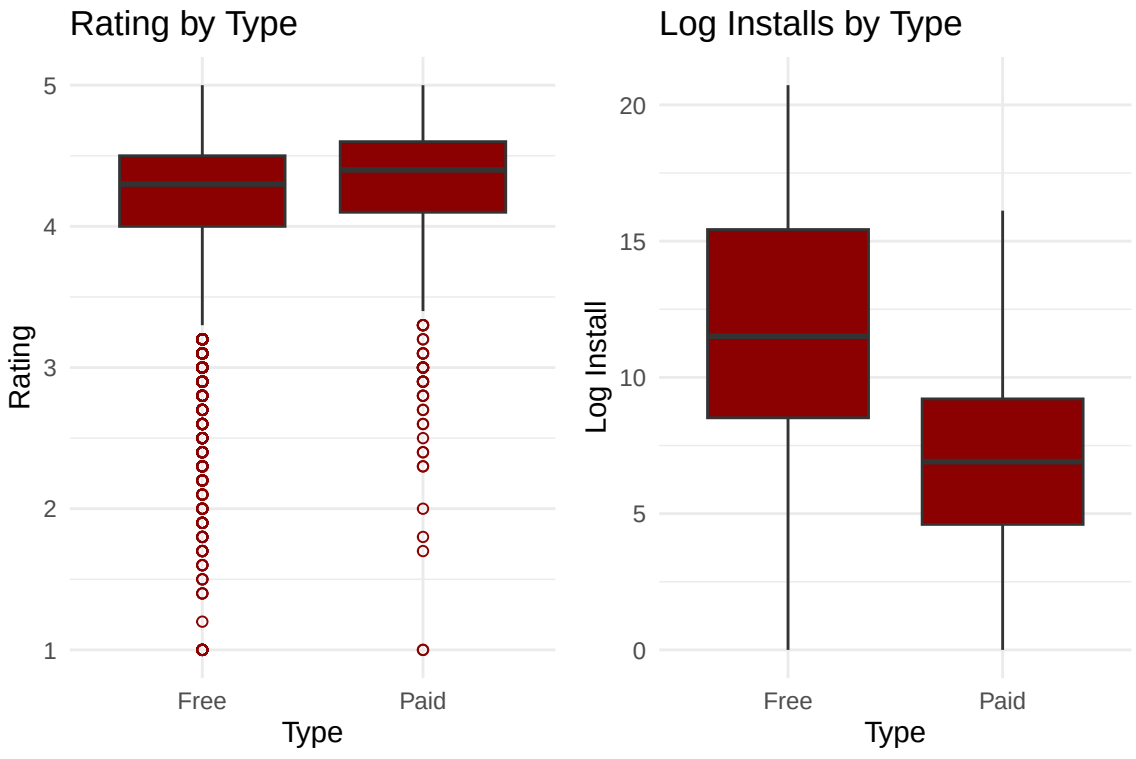
Next, looking at the distribution of category, sorted by count, we see that distribution is very heavily skewed to the right. In particular the first 3 categories (Family, Game, and Tools) have a very large number of apps, after which the count per category drops and falls slowly for the remaining categories.

Secondly, from the genre distribution (recalling that genres are additional categories that apps can be listed as), we observe the same skewness. However the top 30-40% of genres contain most of the count, whereas afterwards the genres listed have a count of almost 0 which suggests that there are many genres with very few apps, suggesting either niche markets or less popular app types.

Table 1: Top 10 Genres and Categories

Rank	Category	Category_Count	Genre	Genre_Count
1	FAMILY	1942	Tools	842
2	GAME	1121	Entertainment	588
3	TOOLS	843	Education	527
4	BUSINESS	427	Business	427
5	MEDICAL	408	Medical	408
6	PRODUCTIVITY	407	Productivity	407
7	PERSONALIZATION	388	Personalization	388
8	LIFESTYLE	373	Lifestyle	372
9	COMMUNICATION	366	Communication	366
10	FINANCE	360	Sports	364

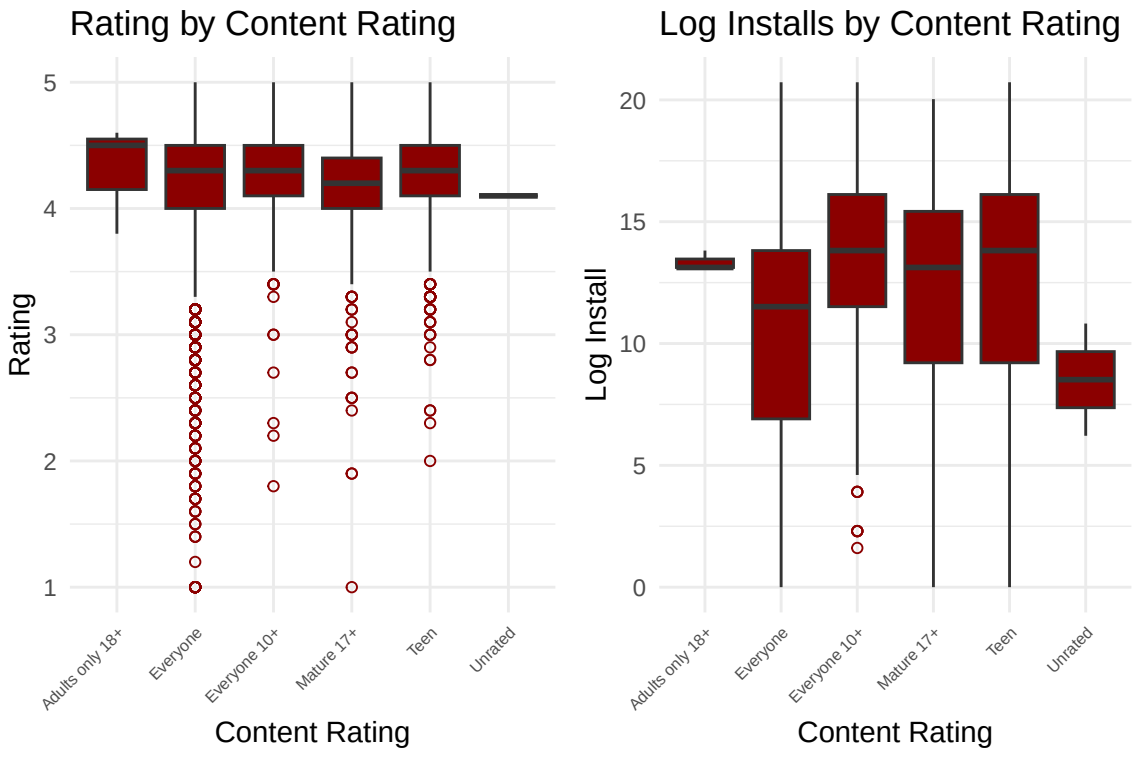
d) Exploring Categorical Features vs Y



The rating of free Apps display a broad range of ratings from 1 to 5, with most ratings clustered around 4. There are many outliers on the lower end, suggesting some free apps are rated poorly. Similar to free

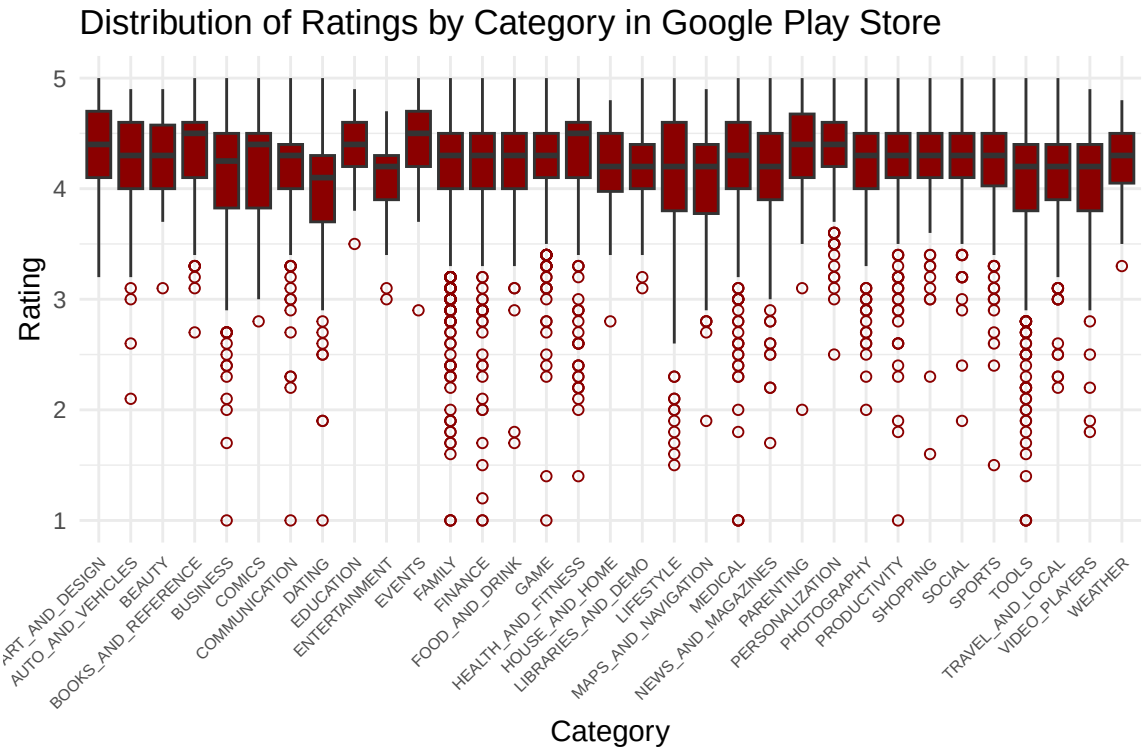
apps, ratings mainly cluster around 4. However, there are fewer lower outliers compared to free apps, indicating generally higher satisfaction among users who purchase apps. Both free and paid apps have a median rating close to 4, showing that overall user satisfaction is high across both app types. The presence of more lower outliers in free apps might indicate variability in quality, where some free apps may not meet user expectations, perhaps due to ads or less functionality.

The rating of free Apps have a broader distribution of log installs, with the median around 15. The range of installs is wide, showing that some free apps achieve significantly higher installs. The distribution of installs is noticeably more constrained and lower than that of free apps. The median log install is lower, and the range (IQR) is narrower, indicating less variation in the number of installs. Thus, free apps tend to reach more users, reflected by the higher median installs and wider distribution. This is expected as the barrier to try a free app is lower than for a paid app. Paid apps, while having fewer installs, tend to have a more consistent range of installs. This could suggest a dedicated user base willing to pay for apps that potentially offer higher quality or unique features not found in free apps.

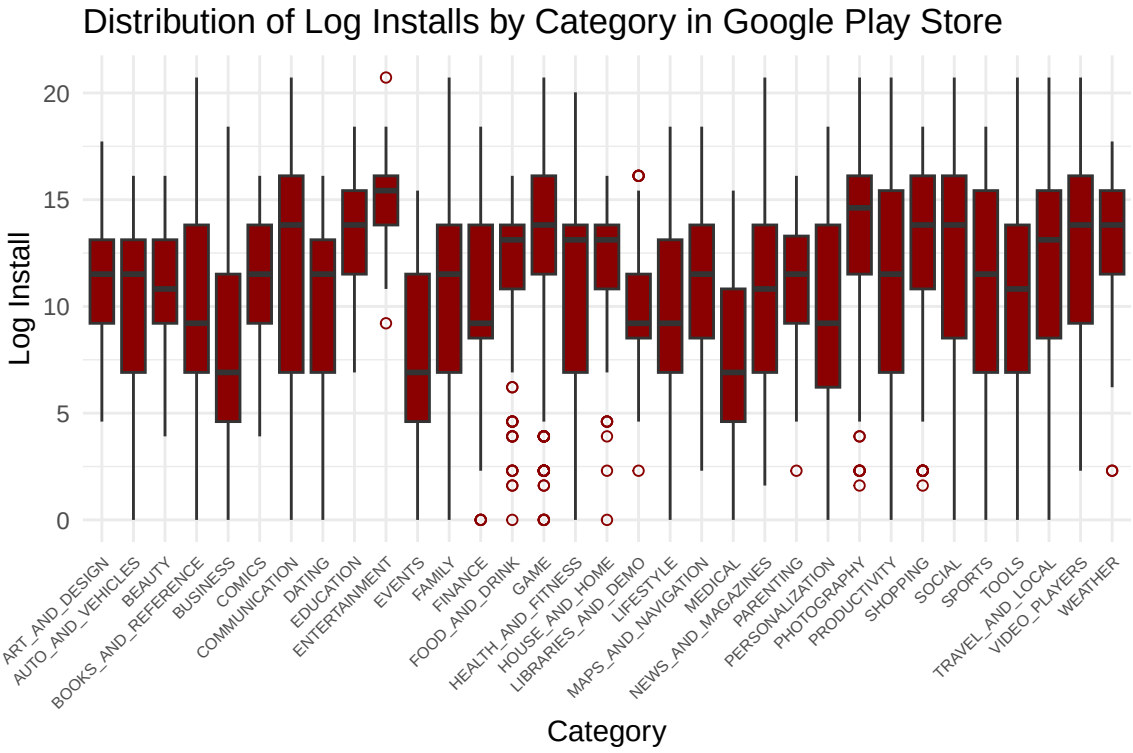


Content ratings such as “Everyone” and “Teen” cover a broad audience, resulting in higher downloads. Categories with restricted audiences like “Adults Only 18+” have both fewer downloads and lower ratings, possibly due to content restrictions or niche market appeal. Thus, Apps aimed at a general audience

("Everyone") might expect higher installations and generally favorable ratings, whereas apps targeted at adults or mature audiences might face more challenges in both downloads and user acceptance.



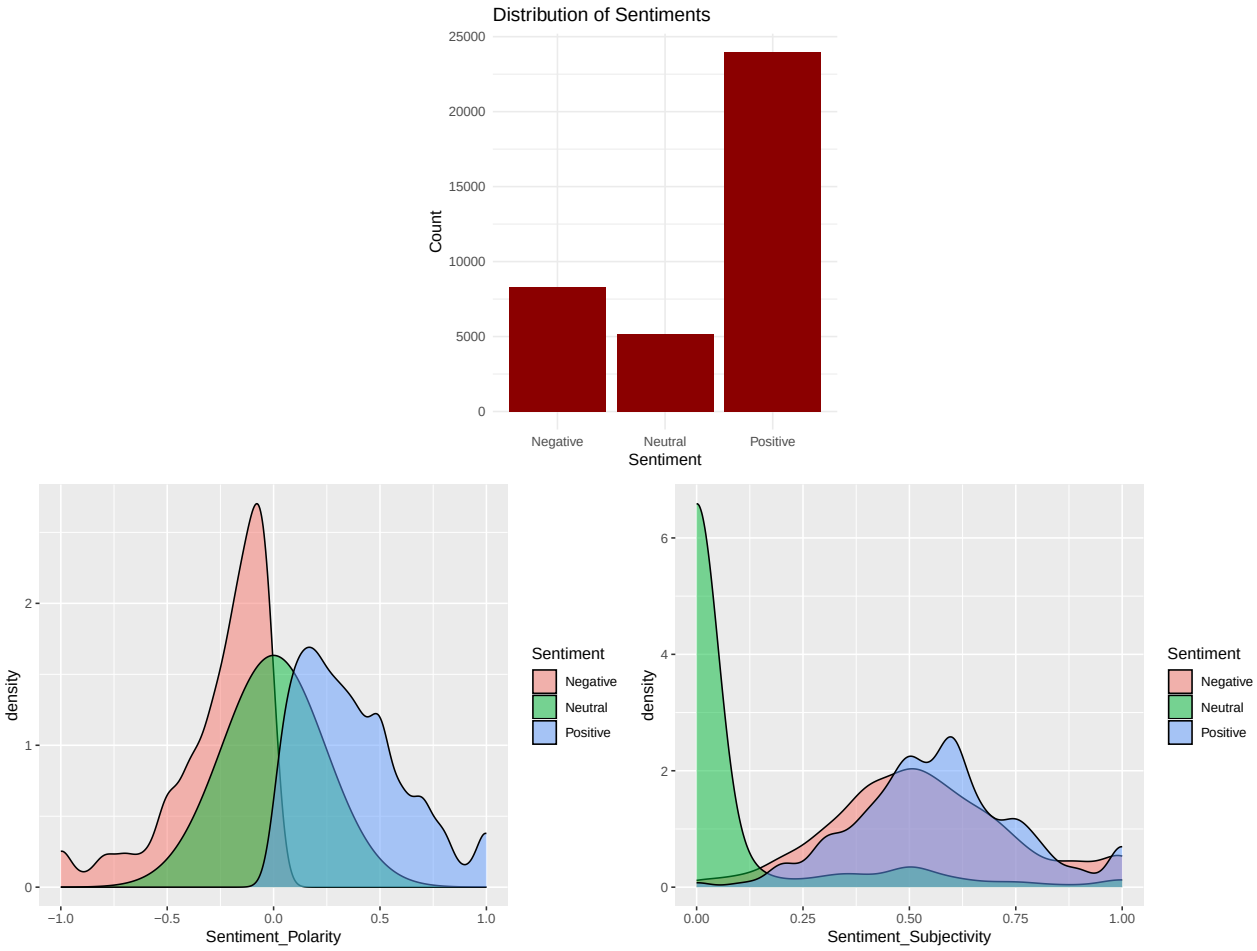
Most categories have median ratings close to 4.0, suggesting a generally positive rating across the board. The boxes are mostly concentrated in the higher rating range (around 3.5 to 4.5), indicating overall good ratings across various categories. Categories like "Art & Design" and "Books & Reference" show less variability in ratings, as indicated by shorter boxes, meaning that ratings in these categories are more consistent. In contrast, categories like "Business" and "Health & Fitness" show wider boxes, indicating more variability in how users rate apps in these categories. Several categories have a significant number of outliers, particularly on the lower side (ratings below 3), such as "Business", "Education", and "Health & Fitness". This could indicate that while many apps in these categories perform well, there are also a considerable number of apps that users are not satisfied with.



The plot shows a wide range of variability in installations across different categories. Categories like "Games" and "Family" show a broad range of installations, evident from the height of their boxes and whiskers, indicating a diverse set of app popularity within these categories. Most categories have their median log installations around the middle of the box, indicating a balanced distribution of data. However, some categories might show a skewed distribution if the median is closer to the top or bottom. Several categories exhibit numerous outliers, especially on the lower side (lower log install counts). This could indicate specific apps in these categories that are significantly less popular than the majority.

e) Sentiment dataset

In the "googleplaystore_user_reviews" dataset, there are already pre-processed features indicating the Sentiment of the review, its polarity, and its subjectivity. Below we visualize the distributions of these features:



Hence, we observe that the majority of reviews have a positive sentiment, which suggests that users tend to leave reviews when they are happy / satisfied with the app. However the number of negative reviews is greater than the number of neutrals which might indicate that users are more likely to leave a review if they feel strongly (either positive or negative) as opposed to being indifferent about it.

The density plot for polarities are as expected, as negative sentiments are clustered around negative polarity values, neutral sentiments around 0, and positive sentiments are spread across positive values.

Finally the subjectivity plot shows that a large right skew for neutral sentiments, which suggests that neutral reviews tend to be more objective. Interestingly, both negative and positive sentiments seem to be centered around a positive subjectivity score of around 0.5, which suggests that these reviews are more subjective and opinion-based.

5. What factors affect the number of installs an app receives?

A. Introduction

B. Analysis

Model 1.

Model 2.

Model 3.

C. Conclusion

6. What are the key features that influence an app's rating?

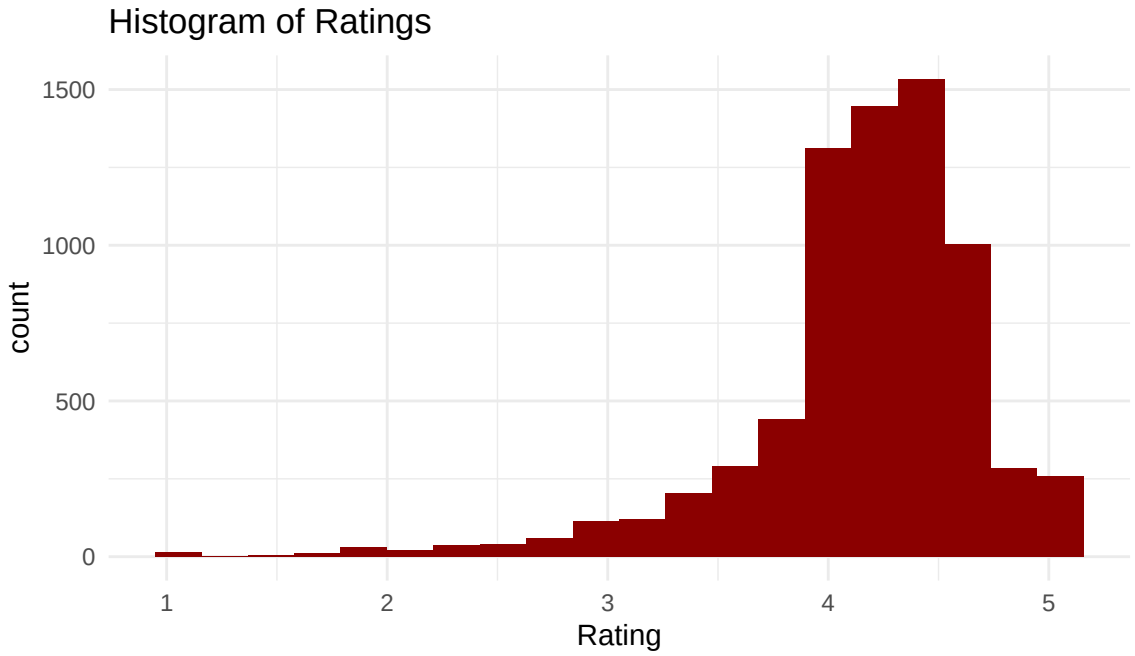
App developers and key stakeholders often have an app's rating as an objective as this influences public perception of the app as well as attracting possible new users. So we decided to investigate our ability to predict an app's rating.

A. Data Preparation and Initial Investigation

Firstly, we process the data by eliminating any rows where the rating is "NaN".
Secondly, we decided to create a new feature called "Days_Since_Update" as this may be more useful than just a specific date of update, and has the added advantage of being a numeric value. The original data was scraped in August 2018, with the latest update for an app being 8th August 2018. The original dataset had no specific day from which it was scraped, so we decided to use 15th August 2018 as an intermediary value, and calculated the different between this date and the "Last.Updated" to create the new feature.
Finally, as we saw in the EDA, some of the numeric variables exhibited skewness, so we log transform Installs and Reviews, and normalize Reviews appropriately (we don't use a log transform here as some apps are listed as size 0).

In addition we also pruned our features to ignore "App" (as app names have added value for our modelling purposes), "Current.Ver" and "Android.Ver" (as again one would assume that these versions hold no significant value on ratings). Out of the remaining categorical variables (Category, Type, Content.Rating, and Genres), we convert these to dummy variables using `model.matrix` which can then be fed into the models.

This subsetted our data into 161 independent variables, and 1 dependent variable (the rating score).

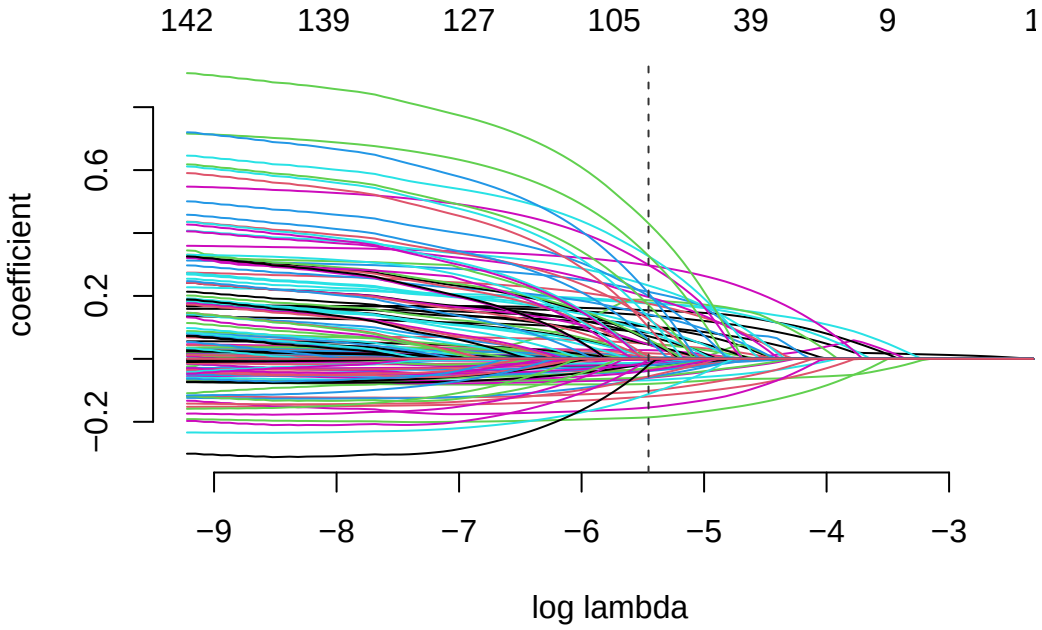


By looking at the histogram of the ratings, it is interesting to observe that the greatest density of ratings is between 4 and 5, which suggests that most users tend to leave a positive rating. Surprisingly the distribution is very heavily skewed to the left, so there are very few ratings close to 1.

It will be interesting to observe if our models are better at predicting high ratings correctly compared to low ratings.

B. Analysis

LASSO We first fit a LASSO model using x and y prepared above, and select a lambda using the AICc.



Above we can see the regularization paths for the penalized β and the minimum AICc selection marked.

```
##
## gaussian gamlr with 162 inputs and 100 segments.
```

Table 2: Highest and Lowest Coefficients from LASSO Model

Feature	Coefficient	Impact
GenresBoard;Pretend Play	0.4306371	Positive
GenresComics;Creativity	0.3278592	Positive
GenresEducation;Creativity	0.3262821	Positive
GenresParenting;Music & Video	0.2986915	Positive
CategoryEVENTS	0.2972870	Positive
CategoryMAPS_AND_NAVIGATION	-0.1018166	Negative
GenresMusic	-0.1117586	Negative
log_Installs	-0.1194121	Negative
GenresEducational	-0.1546060	Negative
CategoryDATING	-0.1858781	Negative

From looking at the coefficients with the largest positive and negative values, we observe that apps with the Genres: Board;Pretend Play, Comics;Creativity, Education;Creativity,all have strong positive association with higher ratings. This could suggest that apps with combined genres, and ones that promote creativity, play, and fun tend to be well-received by users and are could provide higher user-satisfaction

Interestingly, while the genre “education” seems to have a positive impact on rating , the genre “educational” appears to have the most negative impact in our model. This might suggest that apps related to

education are polarizing to users, and they tend to either be satisfied and leave a high rating, or otherwise they do not meet user expectations and negatively influence the rating.

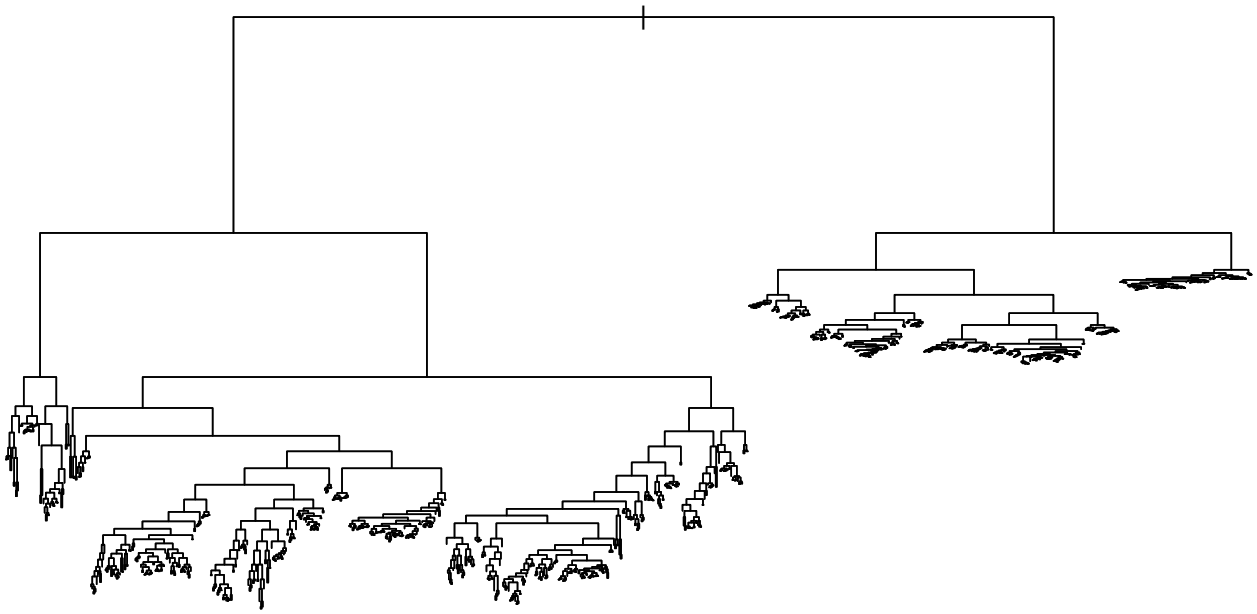
Also, another category of note is "Dating" which also has a large negative impact, and again this might be due to user dissatisfaction with the service, and makes sense considering the complicated and competitive nature of such apps.

Finally, the number of installs also seems to be a feature of note (log_Installs) as it somewhat counter-intuitively negatively impacts ratings. This might be explained by the idea that more popular apps are used by a broader audience with diverse expectations, and thus receive more critical reviews.

```
## [1] "In-sample R^2: 0.159926644162399"
```

The in-sample R^2 for the AICc slice of the LASSO path is ≈ 0.16, which suggests that this model is not a very good fit for our data and only about 16% of the variance in app ratings is explained by our predictors. This might suggest that the relationship between the features and ratings is not linear and more complex than this model can capture. So, next we aim to look at non linear models such as decision trees and random forests.

Decision Tree We build a regression tree model using all the features from above.

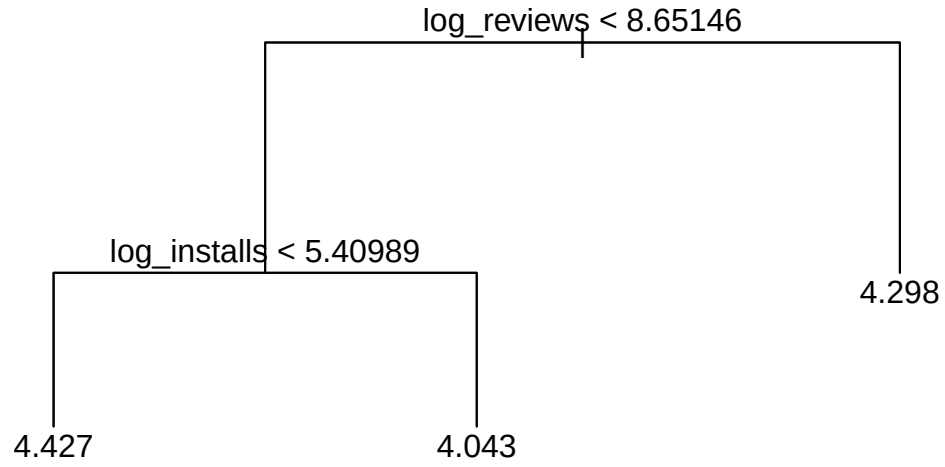


```
## In-sample R^2: 0.5576114
```

```
## Mean Squared Error: 0.1341204
```

The dendogram for the chosen tree is as above, with labels of variables left out for clarity of visualization. Looking at the R^2 score and mean square error, it is clear that the tree performs very well, however given that there are 1126 terminal nodes, it is very likely that overfitting is occuring, and the tree might not perform well out of sample.

To address this, we prune the tree using cross-validation:



The deviance for the cross-validated trees was minimal and the same for the number of leaves = 3 onwards.

So, choosing this as our best tree size, we obtain the tree as above. This has the added advantage of being more interpretable, as now the predictions only depend on two variables: Reviews and Installs.

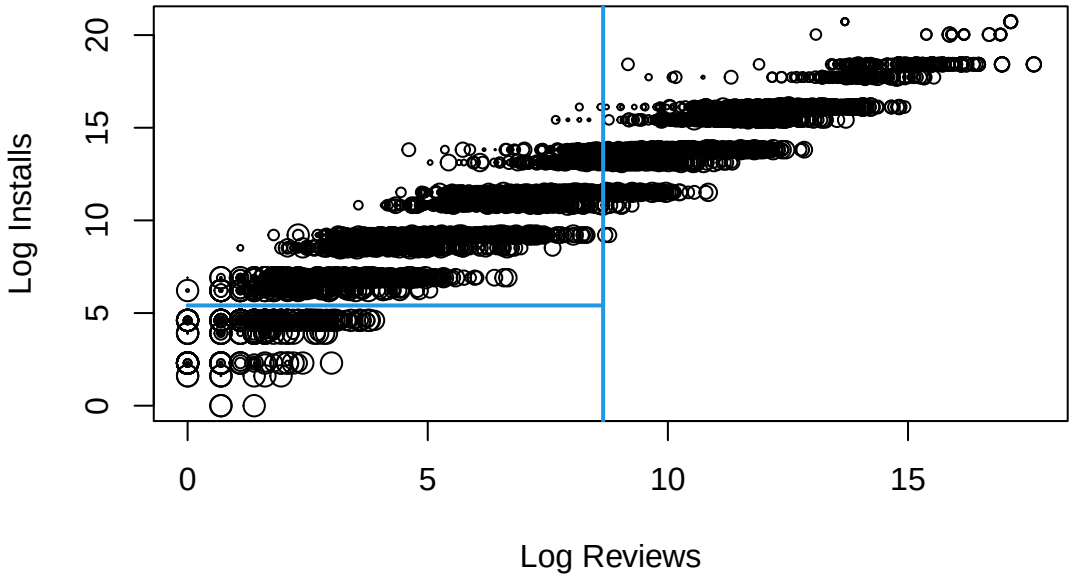
It is interesting to note that having more installs does not necessarily lead to a higher rating. As we see that in the left branch, having log_installs < 5.40989 (equivalent to a threshold of < 223.607 installs), leads to a higher predicted rating than otherwise. This aligns with our finding in the LASSO model above where log_installs had a significant negative coefficient.

Finally we also note that all the leaves of the pruned tree lie between 4.0 and 4.5, and this makes sense as this is where most of the ratings in our dataset lie (as seen on the histogram above). However this may mean that this model does not perform well when predicting lower ratings.

```
## In-sample R^2: 0.06286469
```

```
## Mean Squared Error: 0.2841143
```

The in-sample R^2 shows that when pruning the tree, we have sacrificed some explanation in variability. However the mean-squared error does not significantly increase which is a good sign.



The pruned tree splits the data in the feature space as visualized above, where the point size is proportional to their rating.

Finally, in order to try and improve upon the tree model, we turn to random forests.

Random Forest

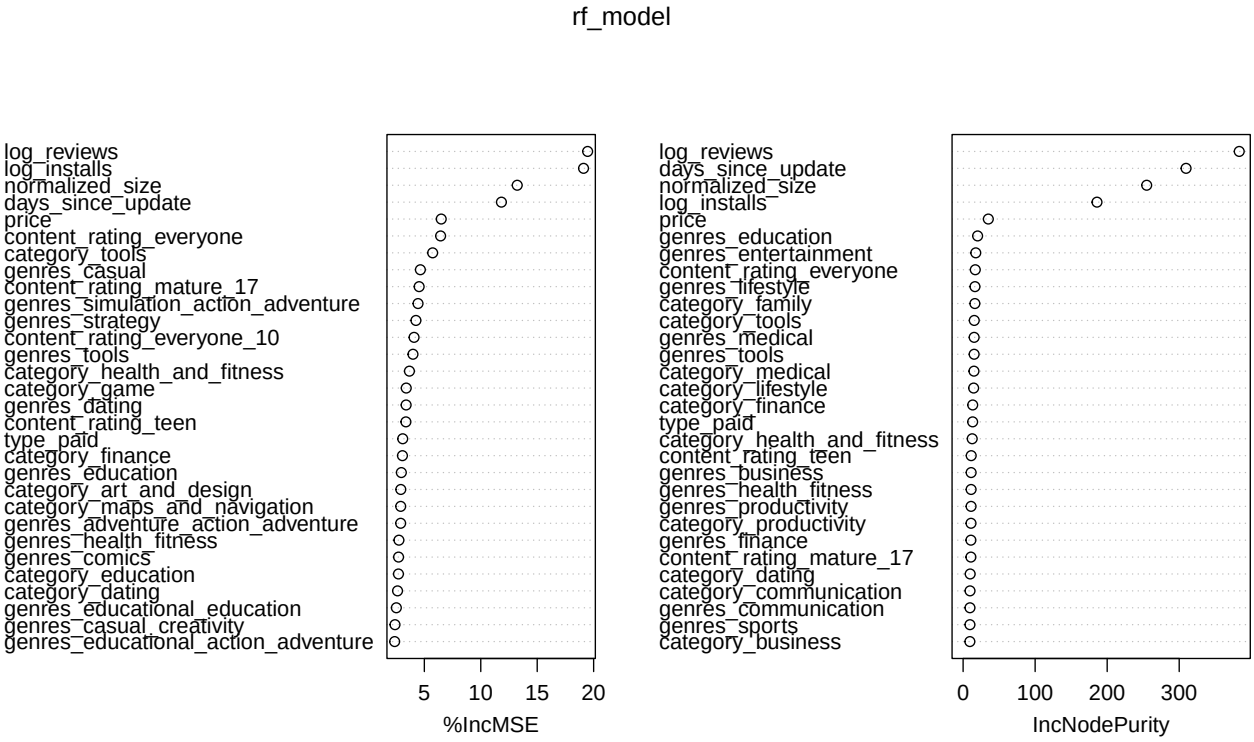
```
## In-sample R^2: 0.7530393
## Mean Squared Error: 0.07487186
```

It is clear to see that this model fits the dataset the best, with an in sample R^2 of ≈ 0.75. This is expected as Random Forests are essentially a type of "model averaging" algorithm which are very flexible. However, we lose the interpretability of a single decision tree.

Despite this, we can still look at the variable importance of each feature, which is measured when constructing the model:

Table 3: Variable Importances from Random Forest

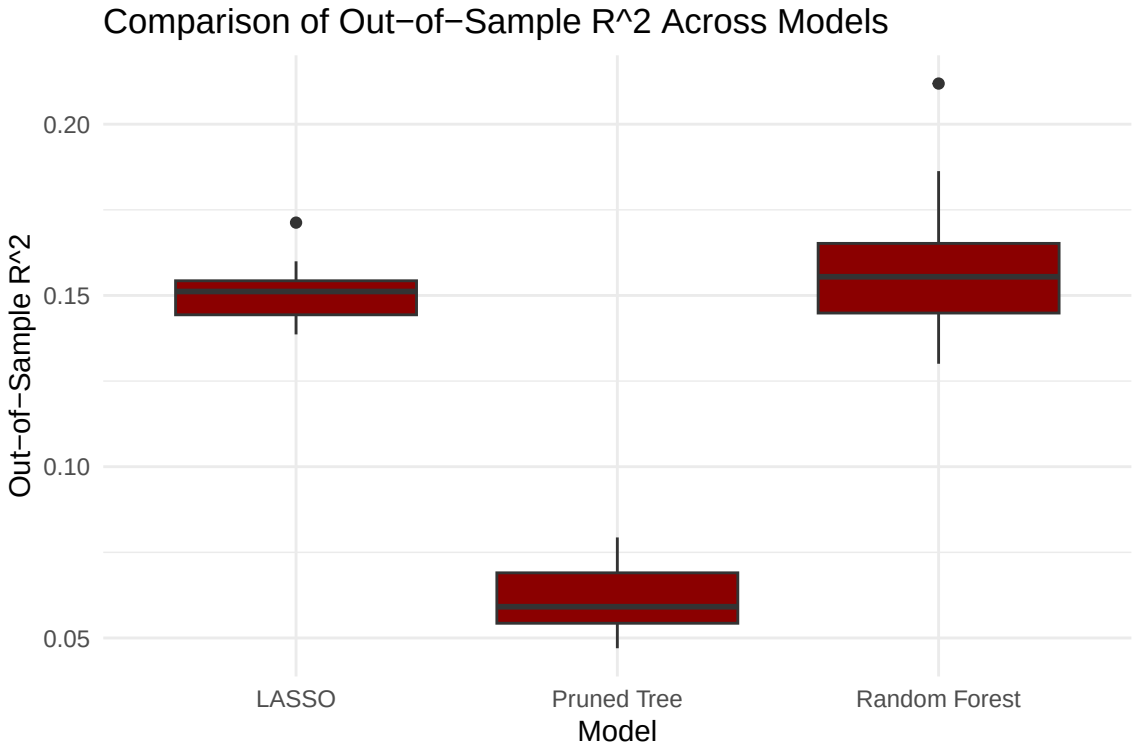
	%IncMSE	IncNodePurity
log_reviews	19.4674421	383.351639
normalized_size	13.2304531	254.862059
log_installs	19.1053739	185.868775
price	6.4970255	35.013352
days_since_update	11.8290876	309.536994
category_art_and_design	2.9151620	1.425252
category_auto_and_vehicles	0.4783673	4.486800
category_beauty	-0.7259967	1.628395
category_books_and_reference	1.8529145	5.651883
category_business	1.0650146	9.391971



Interestingly, the top features ranked in terms of importance for both mean squared error and node purity are numerical features as opposed to categorical ones: Reviews, Installs, Size, Days since Last Update, and Price. This is was not apparent when fitting a LASSO model or a simple decision tree. However the two most important variables for MSE (log_reviews, and log_installs) are consistent with the main splitting features selected by the pruned decision tree earlier.

C. Comparison and Conclusion

To assess the predictive ability of our three models, we randomly split our full dataset into training and test data (in an 80:20 split). We then train each of our three models (LASSO, Pruned Decision Tree, Random Forest), on our training data, and evaluate them on the test data to compute the out of sample R^2 score. To mitigate any randomness we repeat this process 20 times and collect the out of sample R^2 for each model. Our findings our summarised in the boxplot below:



So it is clear to see that our pruned decision tree performs the worst on unseen data, but LASSO and Random Forest perform relatively well. This makes sense as decision trees, even when pruned, are highly sensitive to initial data and are prone to over-fitting so they are not very predictors for unseen applications. Our linear model with LASSO regularization, and penalty parameter chosen by the AICc performs moderately well, but also has a tight interquartile range which indicates stable performance across the test sets which may be desirable in some cases. However the best model is Random Forests, with the highest median OOS R^2 of 0.17308. Although this model does have a broader range of values, which suggests it might be sensitive on the initial data split, it reaches the highest R^2 values.

Thus overall, as a predictor for the rating, our analysis shows that the Random Forest model is the most capable in terms of maximum potential performance. However potential developers and stakeholders using

such a model would have to be wary of the higher variability in the model, and the lack of interpretability compared to other models. This is consistent with results in the literature where Random Forest models have time again proved to good machine learning models for prediction due to their flexibility and ability to capture complex patterns in data.

However one should also be wary that all these models are sensitive to hyper parameters, and often in practice further analysis is required to consider feature engineering and hyper parameter tuning to obtain the best possible model.

7. How does user sentiment in reviews correlate with app ratings?

A. Introduction

B. Analysis

Model 1.

Model 2.

Model 3.

C. Conclusion

8. Conclusion

9. Appendix

```
########################################
# Setup
########################################

knitr::opts_chunk$set(
    echo = FALSE,
    fig.height = 4,
    fig.width = 6,
    warning = FALSE,
    cache = TRUE,
    digits = 3,
    width = 48
)

# Required Packages
library(tidyverse)
library(ggplot2)
library(dplyr)
library(corrplot)
library(grid)
library(gridExtra)
library(RColorBrewer)
library(kableExtra)
library(gamlr)
library(bestNormalize)
library(tree)
library(janitor)
library(randomForest)
library(rsample)
########################################
# 3. a) Understanding the datasets
########################################
# Load the datasets
googleplaystore_raw <- read.csv("data/googleplaystore.csv")
googleplaystore_user_reviews_raw <- read.csv("data/googleplaystore_user_reviews.csv")

# Check the column names
colnames(googleplaystore_raw)
colnames(googleplaystore_user_reviews_raw)

# Check the dimensions
dim(googleplaystore_raw)
dim(googleplaystore_user_reviews_raw)
########################################
# 3. b) Data Cleaning
########################################
# Convert the variables to the appropriate data type
googleplaystore <- googleplaystore_raw |>
  mutate(
```

```
    # Transform Installs and size to numeric
    Installs = gsub("\\+", "", as.character(Installs)),
    Installs = as.numeric(gsub(",", "", Installs)),
    Size = gsub("M", "", Size),
    # Convert apps with size < 1MB to 0, and transform to numeric
    Size = ifelse(grepl("k", Size), 0, as.numeric(Size)),
    # Transform reviews to numeric
    Reviews = as.numeric(Reviews),
    # Change currency numeric
    Price = as.numeric(gsub("\\$", "", as.character(Price))),
    # Convert Last.Updated to date
    Last.Updated = mdy(Last.Updated),
    # Change version number to 1 decimal, and add NAs where appropriate
    Android.Ver = gsub("Varies with device", NaN, Android.Ver),
    Android.Ver = as.numeric(substr(Android.Ver, start = 1, stop = 3)),
    Current.Ver = gsub("Varies with device", NaN, Current.Ver),
    Current.Ver = as.numeric(substr(Current.Ver, start = 1, stop = 3)),
  ) |>
  # Remove apps with Type 0 or NA
  filter(Type %in% c("Free", "Paid")) |>
  # Convert Category, Type, Content.Rating and Genres to factors
  mutate(
    App = as.factor(App),
    Category = as.factor(Category),
    Type = as.factor(Type),
    Content.Rating = as.factor(Content.Rating),
    Genres = as.factor(Genres)
  ) |>
  # Remove duplicate rows
  distinct()
# Remove all rows with nans
googleplaystore_user_reviews <- googleplaystore_user_reviews_raw |>
  filter(Translated_Review != "nan") |>
  # Convert Sentiment to factor
  mutate(Sentiment = as.factor(Sentiment))
##########################################
# 4. a) Overall Histogram Overview
##########################################
common_theme <- theme(
  axis.ticks.x = element_blank(), # Optional: Remove x-axis ticks if not needed
  axis.title.x = element_blank(), # Removes x-axis title for cleaner look
  axis.text.y = element_text(size = 6), # Y-axis text size for uniformity
  axis.title.y = element_blank(), # Removes x-axis title for cleaner look
)

# Determine the top 10 values for categorical data
top_categories <- googleplaystore %>%
  count(Category) %>%
  top_n(10) %>%
  pull(Category)

filtered_google <- googleplaystore %>%
  filter(Category %in% top_categories) %>%
```

```
    mutate(Category = factor(Category, levels = names(sort(table(Category), decreasing = TRUE))))

p1 <- ggplot(filtered_google, aes(x = Category)) +
  geom_bar(fill = "darkred") +
  ggtitle("Top 10 of 33 Categories")+
  theme(axis.text.x = element_text(size = 8,angle = 45, hjust = 1)) +
  common_theme
#########
p2 <- ggplot(googleplaystore, aes(x = Rating)) +
  geom_histogram(bins = 30, fill = "darkred") +
  ggtitle("Rating")+common_theme
#########
p3 <- ggplot(googleplaystore, aes(x = Reviews)) +
  geom_histogram(bins = 30, fill = "darkred") +
  ggtitle("Reviews")+
  theme(axis.text.x = element_text(size = 6,angle = 0, hjust = 1, vjust = 0.5)) +
  common_theme
#########
p4 <- ggplot(googleplaystore, aes(x = Size)) +
  geom_histogram(bins = 30, fill = "darkred") +
  ggtitle("Size")+common_theme
#########
p5 <- ggplot(googleplaystore, aes(x = Installs)) +
  geom_histogram(bins = 30, fill = "darkred") +
  ggtitle("Installs")+
  theme(axis.text.x = element_text(size = 6,angle = 0, hjust = 1, vjust = 0.5)) +
  common_theme
#########
p6 <- ggplot(filtered_google, aes(x = Type)) +
  geom_bar(fill = "darkred") +
  ggtitle("Type")+common_theme
#########
p7 <- ggplot(googleplaystore, aes(x = Price)) +
  geom_histogram(bins = 30, fill = "darkred") +
  ggtitle("Price")+common_theme
#########
filtered_google <- googleplaystore %>%
    mutate(Content.Rating = factor(Content.Rating,
                                   levels = names(sort(table(Content.Rating),
                                                       decreasing = TRUE))))
p8 <- ggplot(filtered_google, aes(x = Content.Rating)) +
  geom_bar(fill = "darkred") +
  ggtitle("Content Rating")+
  theme(axis.text.x = element_text(size = 10,angle = 45, hjust = 1)) +
  common_theme
#########
top_genres <- googleplaystore %>%
  count(Genres) %>%
  top_n(10) %>%
  pull(Genres)
filtered_google <- googleplaystore %>%
  filter(Genres %in% top_genres) %>%
  mutate(Genres = factor(Genres, levels = names(sort(table(Genres), decreasing = TRUE))))
```

```
p9 <- ggplot(filtered_google, aes(x = Genres)) +
  geom_bar(fill = "darkred") +
  ggtitle("Top 10 of 119 Genres") +
  theme(axis.text.x = element_text(size = 10,angle = 45, hjust = 1))+common_theme
#########
p10 <- ggplot(googleplaystore, aes(x = Last.Updated)) +
  geom_histogram(bins = 30, fill = "darkred") +
  ggtitle("Last Updated Date")+common_theme
#########
p11 <- ggplot(filtered_google, aes(x = Current.Ver)) +
  geom_histogram(bins = 30, fill = "darkred") +
  ggtitle("Current Version") +common_theme
#########
p12 <- ggplot(filtered_google, aes(x = Android.Ver)) +
  geom_histogram(bins = 30, fill = "darkred") +
  ggtitle("Minimum Android Version")+common_theme
grid.arrange(p2, p7, p4, p3, p5, p10, p11, p12,
             nrow = 2, ncol = 4, heights = rep(1, 2), widths = rep(1, 4))
##########################################
# 4. b) Correlation Matrix
##########################################
# google_cleaned <- googleplaystore %>%
#   select(Rating, Reviews, Size, Installs, Price)
#
# # Calculate correlation matrix
# cor_matrix <- cor(google_cleaned, use = "complete.obs")  # using complete observations
#
# # Plot the correlation matrix
# corrplot(cor_matrix, method = "color", col = colorRampPalette(c("white", "darkred"))(200),
#          type = "upper", order = "hclust",
#          addCoef.col = "black", # Adding correlation coefficients
#          tl.col = "black", tl.srt = 45, # Text label color and rotation
#          diag = FALSE) # Remove diagonal
##########################################
# 4. b) Correlation Matrix
##########################################
# Select only the numeric columns for the correlation matrix
numeric_columns <- googleplaystore[, sapply(googleplaystore, is.numeric)]

# Compute the correlation matrix
cor_matrix <- cor(numeric_columns, use = "complete.obs")

# Visualize the correlation matrix using a heatmap
corrplot(cor_matrix, method = "color", type = "upper",
         col = colorRampPalette(c("white", "darkred"))(200),
         tl.col = "black", tl.srt = 45,
         addCoef.col = "black", number.cex = 0.7,tl.cex = 0.7,
         title = "Correlation Matrix of Google Play Store Data",
         mar = c(0, 0, 1, 0))
##########################################
# 4. c) Categorical Features
##########################################
# Distribution of Types (Free vs. Paid)
```

```
p1 <- ggplot(googleplaystore, aes(x = Type, fill = Type)) +
  geom_bar(fill = "darkred") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Distribution of App Types", x = "Type", y = "Count") +
  theme(legend.position = "none")

# Distribution of Content Ratings
filtered_google <- googleplaystore %>%
    mutate(Content.Rating = factor(Content.Rating,
                                   levels = names(sort(table(Content.Rating), decreasing = TRUE))))
p2 <- ggplot(filtered_google, aes(x = `Content.Rating`, fill = `Content.Rating`)) +
  geom_bar(fill = "darkred") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Distribution of Content Ratings", x = "Content Rating", y = "Count") +
  theme(legend.position = "none")

# Arrange the plots in a grid
grid.arrange(p1, p2, ncol = 2)
# Count the number of apps in each category
category_counts <- googleplaystore |>
  count(Category) |>
  arrange(desc(n))

# Convert Category to a factor with levels ordered by count
category_counts$Category <- factor(category_counts$Category,
                                   levels = category_counts$Category)

# Plot the distribution of app categories sorted by count
p3 = ggplot(category_counts, aes(x = n, y = Category, fill = Category)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 7)) +
  labs(title = "Distribution of App Categories", x = "Count", y = "Category") +
  theme(legend.position = "none") +
  scale_fill_manual(values = colorRampPalette(brewer.pal(8, "Set2"))(nrow(category_counts))) +
  theme(panel.grid.minor = element_blank()) +
  coord_flip()

# Count the number of apps in each genre
genre_counts <- googleplaystore |>
  count(Genres) |>
  arrange(desc(n))

# Convert Genres to a factor with levels ordered by count
genre_counts$Genres <- factor(genre_counts$Genres, levels = genre_counts$Genres)

# Plot the distribution of app genres sorted by count
p4 = ggplot(genre_counts, aes(x = n, y = Genres, fill = Genres)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Distribution of App Genres", x = "Count", y = "Genres") +
  theme(legend.position = "none") +
  scale_fill_manual(values = colorRampPalette(brewer.pal(8, "Set2"))(nrow(genre_counts))) +
  theme(panel.grid.minor = element_blank(),
```

```
        panel.grid.major = element_blank(),
        axis.text.x = element_blank()) +
  coord_flip()

p3
p4
# Combine the dataframes
combined_df <- data.frame(
  Rank = 1:10,
  Category = category_counts[1:10,]$Category,
  Category_Count = category_counts[1:10,]$n,
  Genre = genre_counts[1:10,]$Genres,
  Genre_Count = genre_counts[1:10,]$n
)

# Print the combined dataframe using kable
kable(combined_df, caption = "Top 10 Genres and Categories", align = 'c') %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed"))
########################################
# 4. d) Categorical Features
########################################
# Create the box plot
b1<-ggplot(googleplaystore, aes(x = Type, y = Rating)) +
  geom_boxplot(outlier.color = "darkred", outlier.shape = 1
               ,fill = "darkred") +  # Red for outliers
  labs(title = "Rating by Type",
       x = "Type",
       y = "Rating") +
  theme_minimal()

b2<-ggplot(googleplaystore, aes(x = Type, y = log(Installs))) +
  geom_boxplot(outlier.color = "darkred", outlier.shape = 1,
               fill = "darkred") +  # Red for outliers
  labs(title = "Log Installs by Type",
       x = "Type",
       y = "Log Install") +
  theme_minimal()

grid.arrange(b1, b2, ncol = 2)
b1<-ggplot(googleplaystore, aes(x = Content.Rating, y = Rating)) +
  geom_boxplot(outlier.color = "darkred", outlier.shape = 1,
               fill = "darkred") +  # Red for outliers
  labs(title = "Rating by Content Rating",
       x = "Content Rating",
       y = "Rating") +
  theme_minimal() +
  theme(axis.text.x = element_text(size = 6,angle = 45, hjust = 1, vjust = 1))

b2<-ggplot(googleplaystore, aes(x = Content.Rating, y = log(Installs))) +
  geom_boxplot(outlier.color = "darkred", outlier.shape = 1,
               fill = "darkred") +  # Red for outliers
  labs(title = "Log Installs by Content Rating",
       x = "Content Rating",
```

```
       y = "Log Install") +
  theme_minimal() +
  theme(axis.text.x = element_text(size = 6,angle = 45, hjust = 1, vjust = 1))

grid.arrange(b1, b2, ncol = 2)
# Create the box plot
ggplot(googleplaystore, aes(x = Category, y = Rating)) +
  geom_boxplot(outlier.color = "darkred", outlier.shape = 1,
               fill = "darkred") +  # Red for outliers
  labs(title = "Distribution of Ratings by Category in Google Play Store",
       x = "Category",
       y = "Rating") +
  theme_minimal() +
  theme(axis.text.x = element_text(size = 6,angle = 45, hjust = 1, vjust = 1))
# Create the box plot
ggplot(googleplaystore, aes(x = Category, y = log(Installs))) +
  geom_boxplot(outlier.color = "darkred", outlier.shape = 1,
               fill = "darkred") +  # Red for outliers
  labs(title = "Distribution of Log Installs by Category in Google Play Store",
       x = "Category",
       y = "Log Install") +
  theme_minimal() +
  theme(axis.text.x = element_text(size = 6, angle = 45, hjust = 1, vjust = 1))
########################################
# 4. e) Sentiment dataset
########################################

# Distribution of Sentiments
p1 = ggplot(googleplaystore_user_reviews, aes(x = Sentiment)) +
  geom_bar(fill = "darkred") +
  labs(title = "Distribution of Sentiments", x = "Sentiment", y = "Count") +
  theme_minimal()

p2 = googleplaystore_user_reviews |>
  ggplot(aes(x=Sentiment_Polarity, group=Sentiment, fill=Sentiment)) +
  geom_density(adjust=1.5, alpha=0.5)

p3 = googleplaystore_user_reviews |>
  ggplot(aes(x=Sentiment_Subjectivity, group=Sentiment, fill=Sentiment)) +
  geom_density(adjust=1.5, alpha=0.5)

# Arrange the plots in a grid
# Arrange the plots in the specified layout
p1_centered <- arrangeGrob(nullGrob(), p1, nullGrob(), ncol = 3)
p2_p3_row <- arrangeGrob(p2, p3, ncol = 2)
grid.arrange(p1_centered, p2_p3_row, nrow = 2)
########################################
# 6. A. Data Preparation and Initial Investigation
########################################
# Data Processing
# Filter out rows where Rating is NaN
ratings_data <- googleplaystore |> na.omit()
```

```r
# The data is scraped from August 2018
# Create a feature called Days Since Last Update
ratings_data$Days_Since_Update <- as.numeric(as.Date("2018-08-15")
                                             - ratings_data$Last.Updated)

# Log transform Installs, Size, and Reviews to remove skewness
ratings_data <- ratings_data |>
  mutate(log_Installs = log(ratings_data$Installs),
         normalized_Size = bestNormalize(ratings_data$Size)$x.t,
         log_Reviews = log(ratings_data$Reviews))

# Create dummy variables using model.matrix
x <- model.matrix(Rating ~ log_Reviews + normalized_Size + log_Installs +
                    Price + Days_Since_Update + Category + Type +
                    Content.Rating + Genres - 1, data = ratings_data)
# Response variable
y <- ratings_data$Rating
ggplot(ratings_data, aes(x = Rating)) +
  geom_histogram(bins = 20, fill = "darkred") +
  ggtitle("Histogram of Ratings") +
  theme_minimal()
##########################################
# 6. B. i) LASSO
##########################################
# Fit the LASSO model using gamlr
set.seed(1024)
lasso_model <- gamlr(x, y, lambda.min.ratio=1e-3)
plot(lasso_model)
# Find the index with lowest AICc
summary_output = summary(lasso_model)
best_aicc_index <- which.min(summary_output$aicc)

coefficients_lasso <- lasso_model$beta[, best_aicc_index]
highest_coefs <- head(sort(coefficients_lasso, decreasing = TRUE), 5)
lowest_coefs <- head(sort(coefficients_lasso, decreasing = FALSE), 5)

# Convert them to data frames
highest_df = data.frame(Feature = names(highest_coefs),
                        Coefficient = highest_coefs, Impact = "Positive")
lowest_df = data.frame(Feature = names(lowest_coefs),
                       Coefficient = lowest_coefs, Impact = "Negative")
coefficients_df <- rbind(highest_df, lowest_df)

# Ordering the dataframe by coefficient magnitude for clearer interpretation
coefficients_df <- coefficients_df[order(coefficients_df$Coefficient, decreasing=TRUE),]
kable(coefficients_df,
      caption = "Highest and Lowest Coefficients from LASSO Model",
      row.names = FALSE) |>
  kable_styling(bootstrap_options = c("striped", "hover"))
# Find the R2 of the lowest AICc slice
best_r2 <- summary_output$r2[best_aicc_index]
print(paste("In-sample R^2:", best_r2))
##########################################
```

```r
# 6. B. ii) Decision Tree
########################################
# Use names without spaces for tree package
x = clean_names(as.data.frame(x))
tree_model <- tree(y ~ ., data=x, mindev=0.00001)

plot(tree_model)
evaluate_tree_model <- function(model, x, y) {
    # Predict ratings using the tree model
    predictions <- predict(model, x)

    # Calculate SSR (Sum of Squares of Residuals)
    SSR <- sum((y - predictions)^2)

    # Calculate SST (Total Sum of Squares)
    mean_rating <- mean(y)
    SST <- sum((y - mean_rating)^2)

    # Compute R^2
    R_squared <- 1 - (SSR / SST)

    # Calculate Mean Squared Error
    mse <- mean((y - predictions)^2)

    # Print the results
    cat("In-sample R^2:", R_squared, "\n")
    cat("Mean Squared Error:", mse, "\n")
}

evaluate_tree_model(tree_model, x, y)
# Cross Validation
cv_tree_model <- cv.tree(tree_model, K=50)

# Find the last index corresponding to minimum deviance
tree_index = max(which(cv_tree_model$dev == min(cv_tree_model$dev)))

# Find the tree size
tree_size = cv_tree_model$size[tree_index]

# Prune the tree
pruned_tree <- prune.tree(tree_model, best=tree_size)
plot(pruned_tree)
text(pruned_tree, pretty = 1)
evaluate_tree_model(pruned_tree, x, y)
plot(x$log_reviews, x$log_installs, cex=exp(y)*.01,
     xlab = "Log Reviews", ylab = "Log Installs")
abline(v=8.65146, col=4, lwd=2)
lines(x=c(0,8.65146), y=c(5.40989,5.40990), col=4, lwd=2)
########################################
# 6. B. iii) Random Forest
########################################
rf_model <- randomForest(y ~ ., data=x, importance = TRUE, ntree=50)
load("model_data/rf_model.RData")
```

```r
evaluate_tree_model(rf_model, x, y)
kable(head(importance(rf_model), n = 10),
      caption = "Variable Importances from Random Forest",
      format = 'markdown') |>
  kable_styling(bootstrap_options = c("striped", "hover"))

varImpPlot(rf_model)
########################################
# 6. C. Comparison and Conclusion
########################################
# Function to compute out of sample R2 for given models
compute_OOS_R2 <- function(model, test_x, test_y) {
    # Predict ratings using the model
    predictions <- predict(model, test_x)

    # Calculate SSR (Sum of Squares of Residuals)
    SSR <- sum((test_y - predictions)^2)

    # Calculate SST (Total Sum of Squares)
    mean_rating <- mean(test_y)
    SST <- sum((test_y - mean_rating)^2)

    # Compute R^2
    R_squared <- 1 - (SSR / SST)

    return (R_squared)
}

# Function to return a trained tree model
get_tree_model <- function(train_x, train_y) {
    # Intial Tree
    tree_model <- tree(y ~ ., data=x, mindev=0.00001)

    # Cross Validation
    cv_tree_model <- cv.tree(tree_model, K=50)

    # Find the last index corresponding to minimum deviance
    tree_index = max(which(cv_tree_model$dev == min(cv_tree_model$dev)))

    # Find the tree size
    tree_size = cv_tree_model$size[tree_index]

    # Prune the tree
    pruned_tree <- prune.tree(tree_model, best=tree_size)

    return(pruned_tree)
}
set.seed(2048)

# Initialize a data frame to store R-squared results
results_df <- data.frame(iteration = integer(),
                         R2_LASSO = numeric(),
                         R2_Tree = numeric(),
```

```
                          R2_rf = numeric())


# Loop for 20 iterations
for (i in 1:20) {
    # Progress Counter
    print(paste("Iteration: ", i, "/20"))

    # Randomly split data into training and testing sets
    n <- length(y)
    split <- sample(c(TRUE, FALSE), n, replace = TRUE, prob = c(0.8, 0.2))
    x_train <- x[split, ]
    x_test <- x[!split, ]
    y_train <- y[split]
    y_test <- y[!split]

    # Fit models on the training data
    print("Training LASSO")
    model_lasso <- gamlr(x_train, y_train, lambda.min.ratio=1e-3)
    print("Training Tree")
    model_tree <- get_tree_model(x_train, y_train)
    print("Training Random Forest")
    model_rf <- randomForest(y_train ~ ., data=x_train, ntree=50)

    # Compute out-of-sample R2 for each model
    R2_LASSO <- compute_OOS_R2(model_lasso, x_test, y_test)
    R2_Tree <- compute_OOS_R2(model_tree, x_test, y_test)
    R2_rf <- compute_OOS_R2(model_rf, x_test, y_test)

    # Store the results in the dataframe
    results_df <- rbind(results_df,
                        data.frame(iteration = i,
                                   R2_LASSO = R2_LASSO,
                                   R2_Tree = R2_Tree,
                                   R2_rf = R2_rf))
}
load("model_data/results_df.RData")
# Plot the results
results_long <- pivot_longer(results_df,
                             cols = c(R2_LASSO, R2_Tree, R2_rf),
                             names_to = "Model",
                             values_to = "R2")
results_long$Model <- factor(results_long$Model, levels = c("R2_LASSO", "R2_Tree", "R2_rf"))


# Create a boxplot to compare the R2 scores for each model
ggplot(results_long, aes(x = Model, y = R2, fill = Model)) +
    geom_boxplot(fill = "darkred") +
    labs(title = "Comparison of Out-of-Sample R^2 Across Models",
         x = "Model",
         y = "Out-of-Sample R^2") +
    theme_minimal() +
    scale_x_discrete(labels = c("LASSO", "Pruned Tree", "Random Forest"))
```