The University of Chicago Booth School of Business

BUSN 41201 - Big Data - Final Project

Don't Play Me!

Analyzing Play Store Data

26 May 2024

Yi Cao, Shri Lekkala, Ningxin Zhang

Honor Code

We pledge our honor that we have not violated the Booth Honor Code during this assignment.

Contents

1. Executive Summary . . . 3
2. Introduction . . . 4
3. Dataset . . . 5
    a) Understanding the data . . . 5
    b) Data Cleaning . . . 6
4. Exploratory Analysis . . . 7
    a) Numerical Features . . . 7
    b) Correlation Matrix . . . 8
    c) Categorical Features . . . 9
    d) Exploring Categorical Features vs Y . . . 11
        Type . . . 11
        Content Rating . . . 12
        Category . . . 13
    e) Sentiment dataset . . . 14
5. What factors affect the number of installs an app receives? . . . 16
    A. Data Preparation and Initial Investigation . . . 16
    B. Analysis . . . 16
        Lasso . . . 16
        Decision Tree . . . 18
        Random Forest . . . 20
    C. Conclusion . . . 21
6. What are the key features that influence an app's rating? . . . 23
    A. Data Preparation and Initial Investigation . . . 23
    B. Analysis . . . 24
        LASSO . . . 24
        Decision Tree . . . 26
        Random Forest . . . 28
    C. Conclusion . . . 30
7. How does user sentiment in reviews correlate with app ratings? . . . 32
    A. Introduction . . . 32
    B. Analysis . . . 33
        Top 20 Words in Each Sentiment . . . 33
        Regression Analysis . . . 35
        Enumerating Sentiment Scores . . . 38
    C. Conclusion . . . 45
8. Conclusion and Improvements . . . 47
9. Appendix . . . 48

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

1. Executive Summary

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

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[1], 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. All the information is scraped from the Play Store in August 2018.

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.

1: https://www.kaggle.com/datasets/lava18/google-play-store-apps

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

```
# 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)) |>
  distinct() # Remove duplicate rows
```

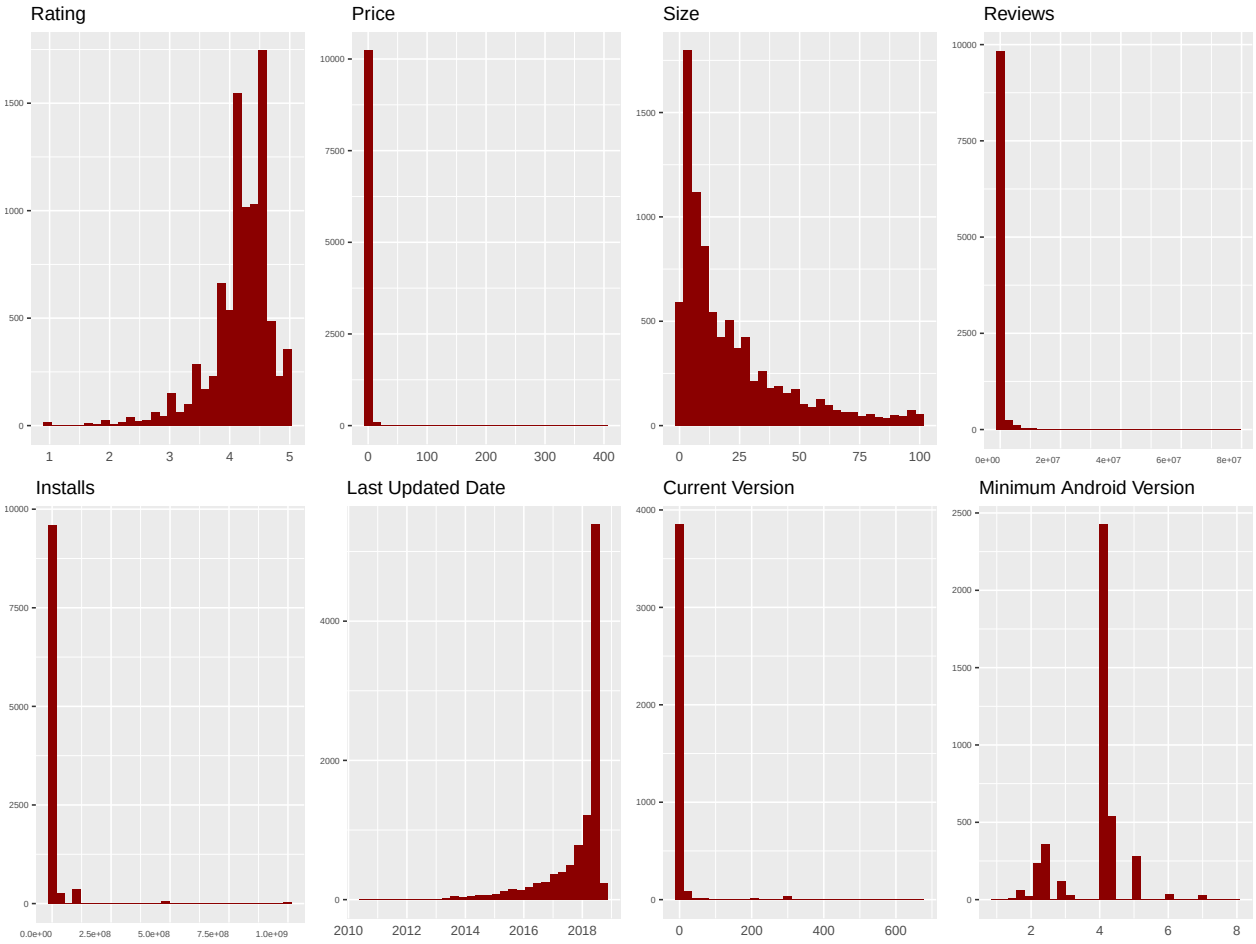
For the `googleplaystore` dataset, we first process the variables by converting columns to the appropriate datatype. For example Installs, Size, Reviews Price, and Android.Ver are converted to numerics, Last.Updated is converted to date. Then we filter out apps with Type 0 or NA, and remove duplicated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

```
# 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.

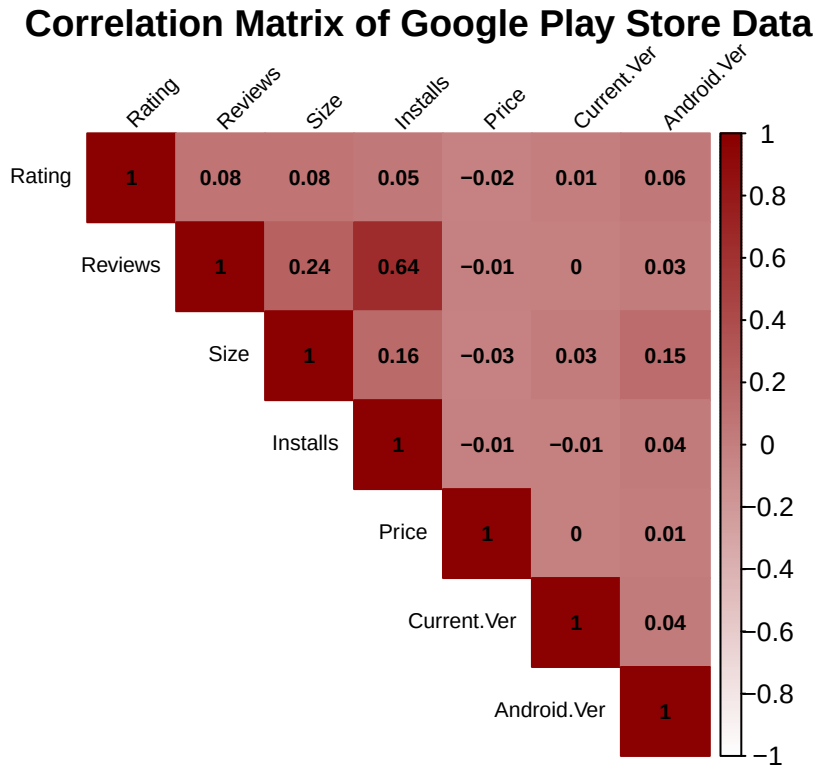
Regarding other numerical variables, the vast majority of apps are free, with the few paid apps showing a wide price range, including some very expensive ones. Most apps are small in size, with a significant drop-off as size increases. The majority are less than 25 MB, with very few exceeding 100 MB. 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

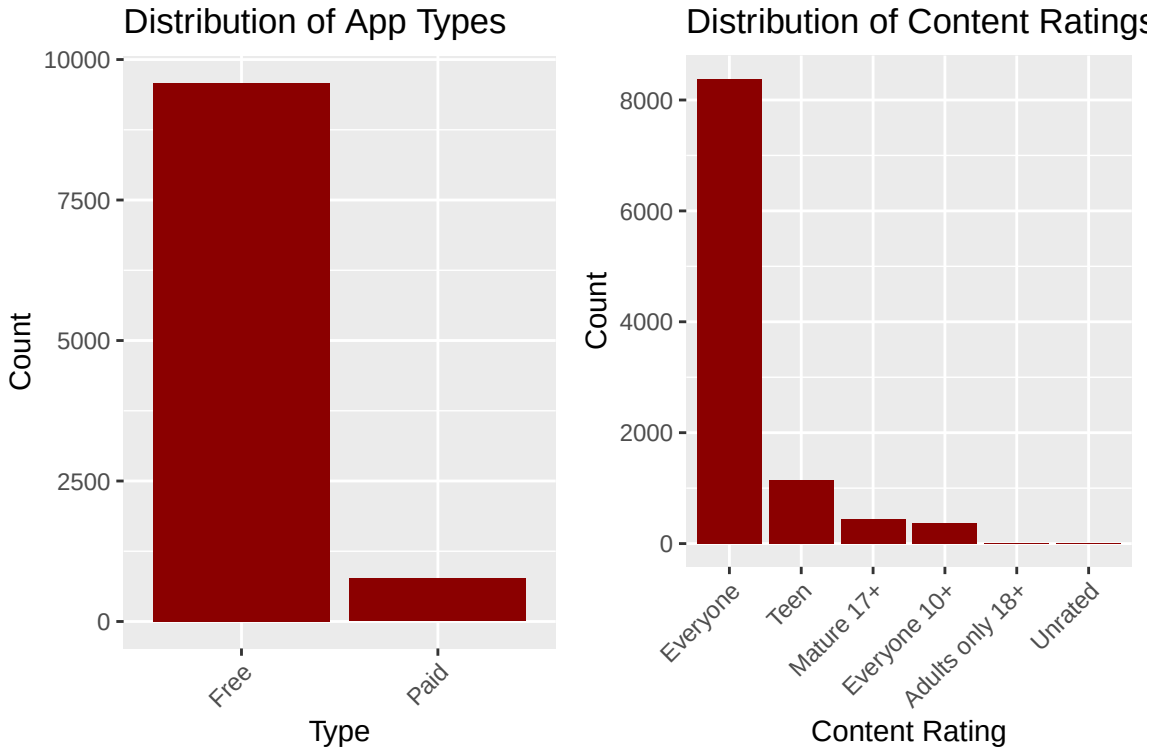
We begin by analysing the correlation matrix of all the numeric variables for googleplaystore:



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.

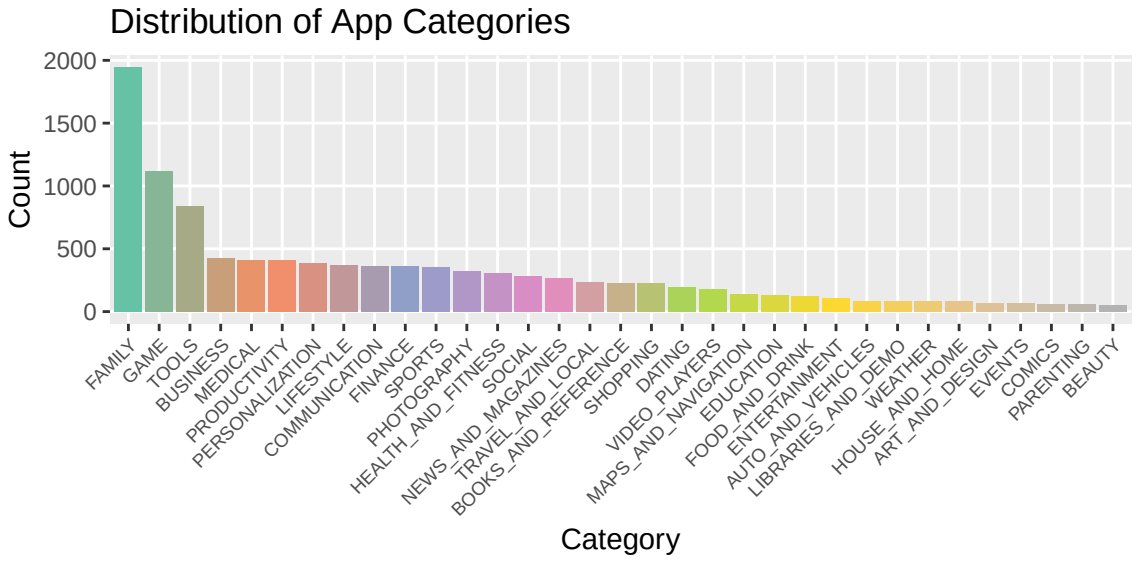
c) Categorical Features

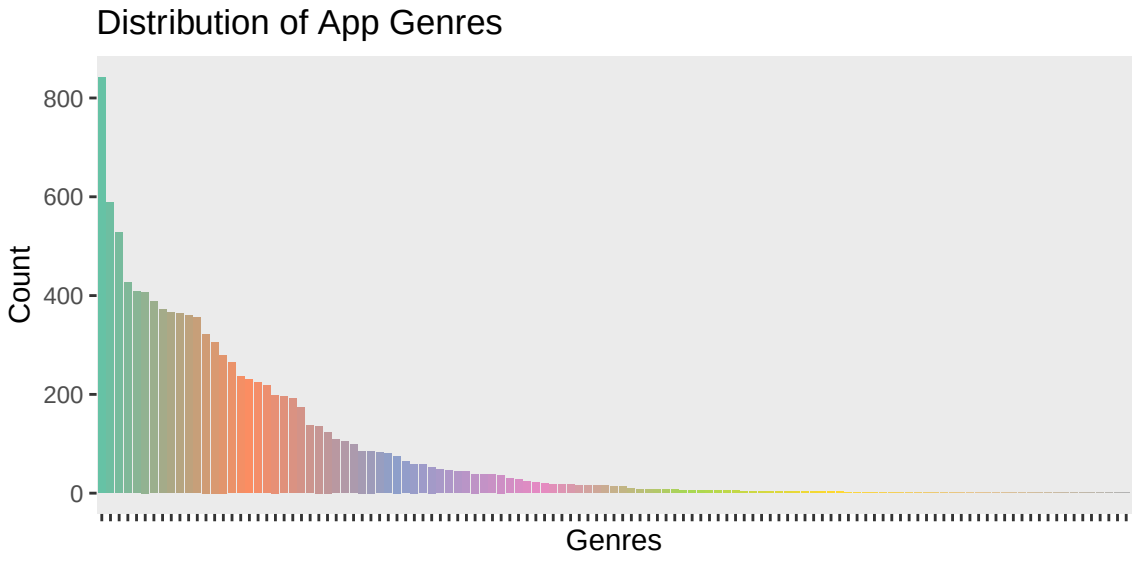
We also look at the distribution of the categorical features in our dataset:



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.

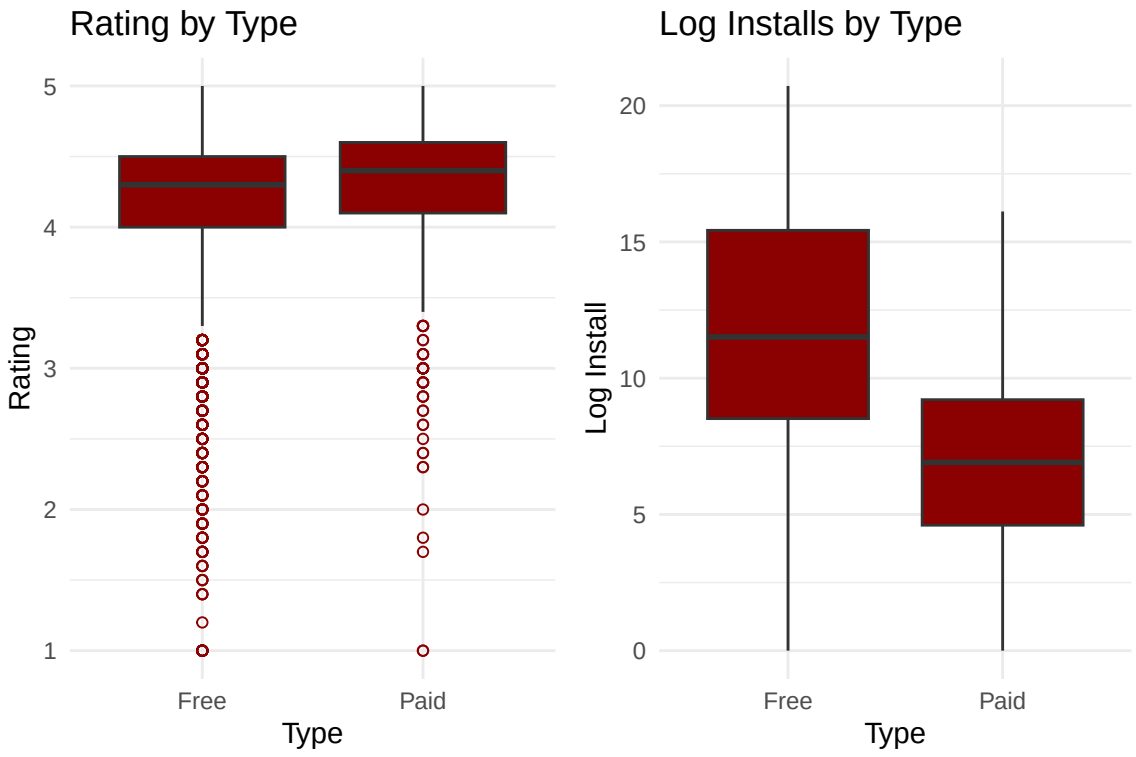
Table 1: Top 10 Genres and Categories

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

Now we understand the distribution of categorical variables, we make box plots group by these categorical features, trying to find out the potentail relationship between them and our Y variables.

Type

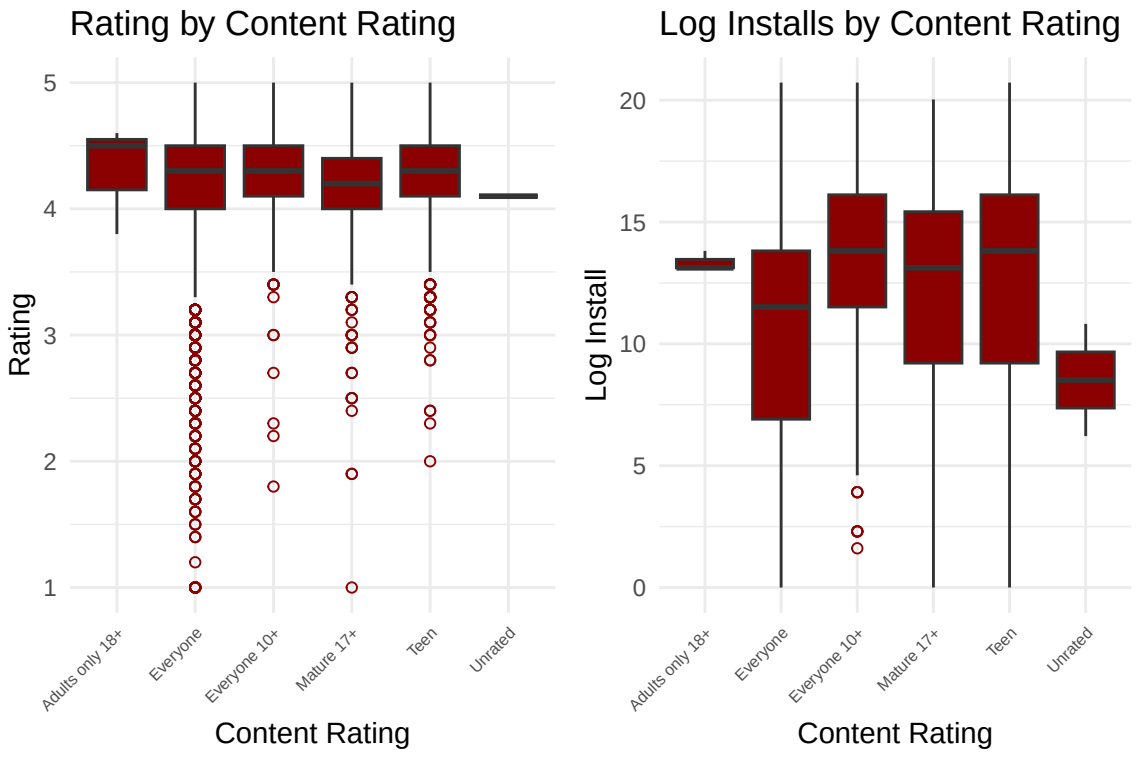


The rating of free Apps display a broad range of ratings from 1 to 5, with most ratings clustered around 4. There are many outliers on the lower end, suggesting some free apps are rated poorly. Similar to free

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.

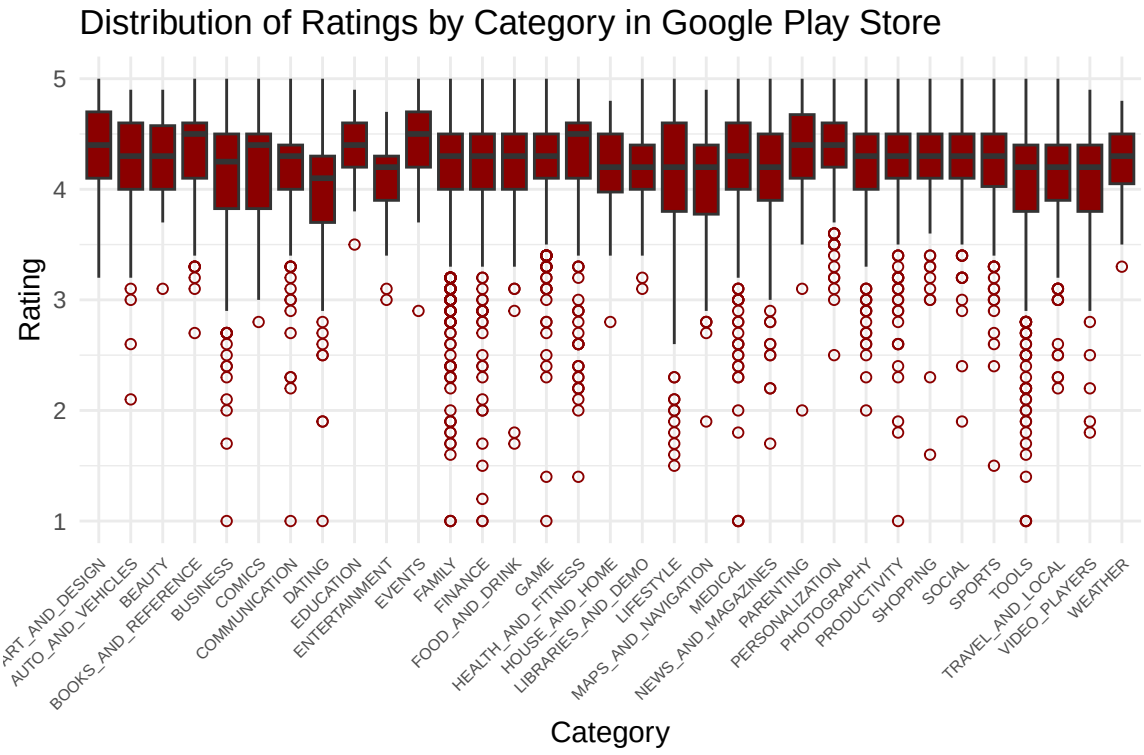
Content Rating



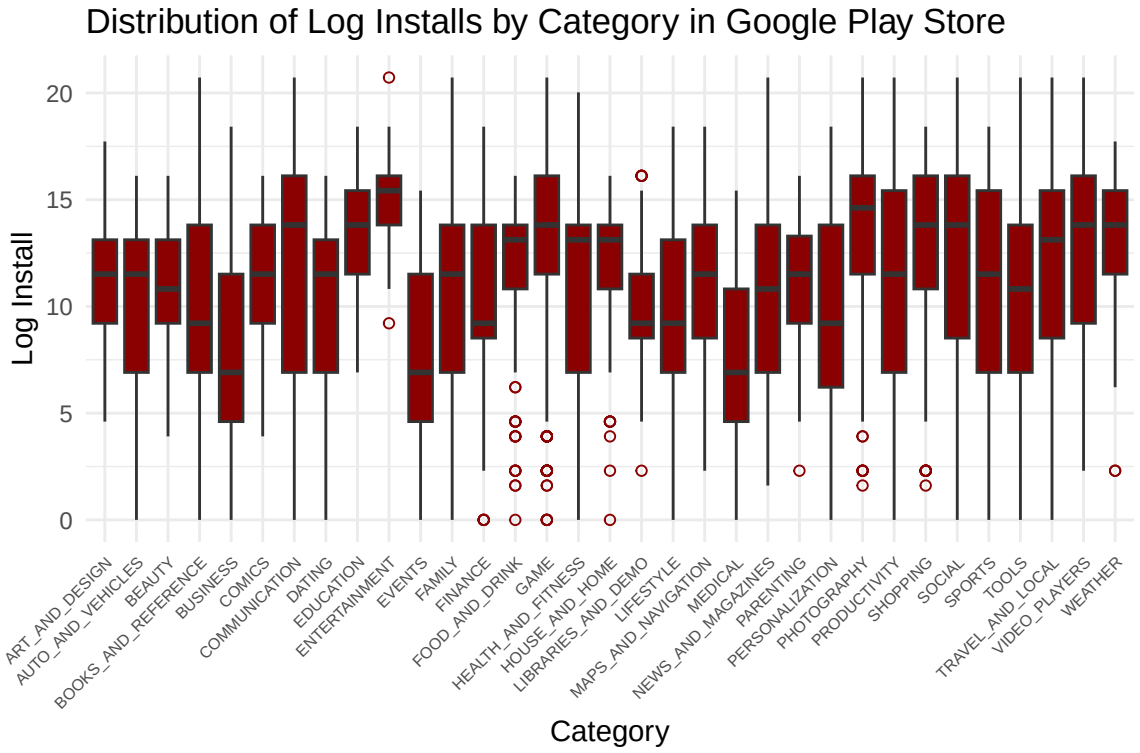
Content ratings such as “Everyone” and “Teen” cover a broad audience, resulting in higher downloads. Categories with restricted audiences like “Adults Only 18+” have both fewer downloads and lower ratings,

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.

Category



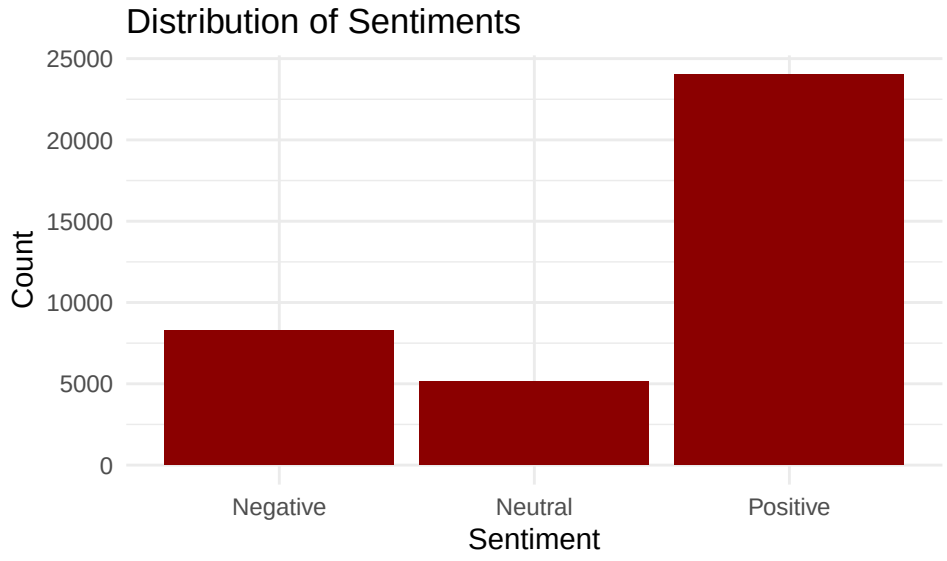
Most categories have median ratings close to 4.0, suggesting a generally positive rating across the board. The boxes are mostly concentrated in the higher rating range (around 3.5 to 4.5), indicating overall good ratings across various categories. Categories like "Art & Design" and "Books & Reference" 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.



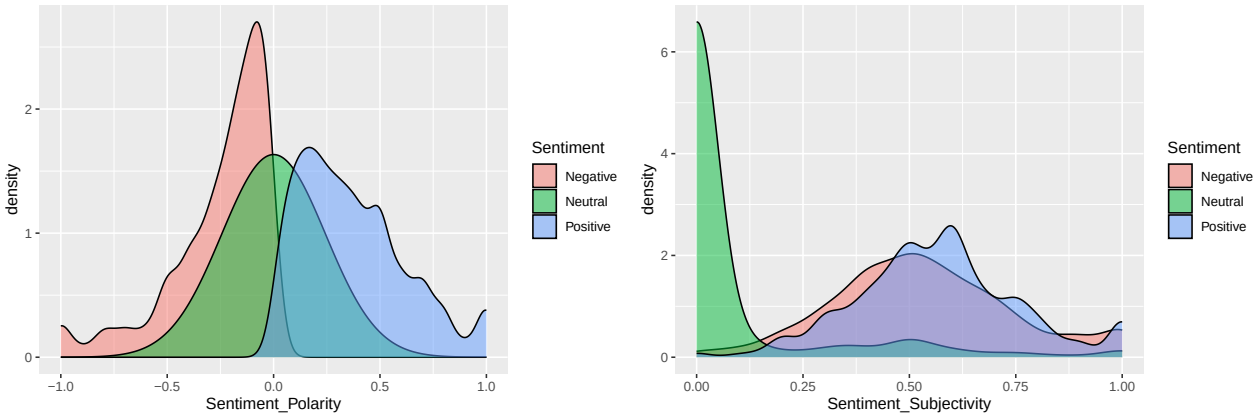
The plot shows a wide range of variability in installations across different categories. Categories like "Games" and "Family" show a broad range of installations, evident from the height of their boxes and whiskers, indicating a diverse set of app popularity within these categories. Most categories have 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.



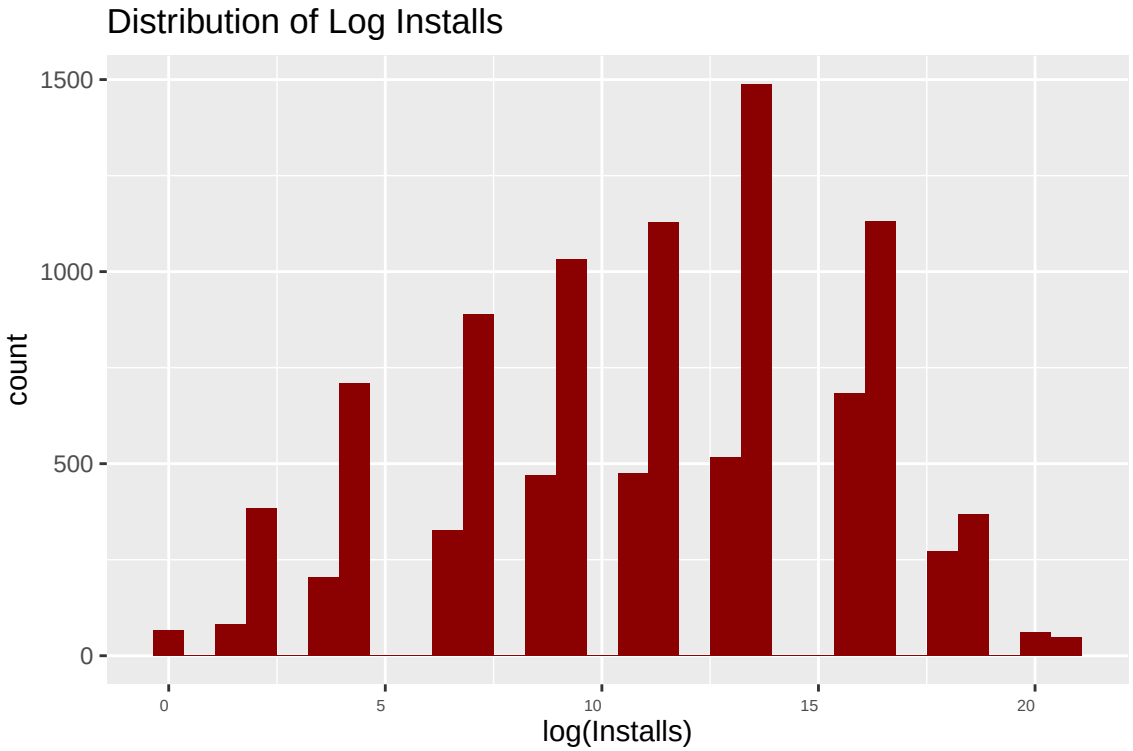
The density plot for polarities are as expected, as negative sentiments are clustered around negative polarity values, neutral sentiments around 0, and positive sentiments are spread across positive values.

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?

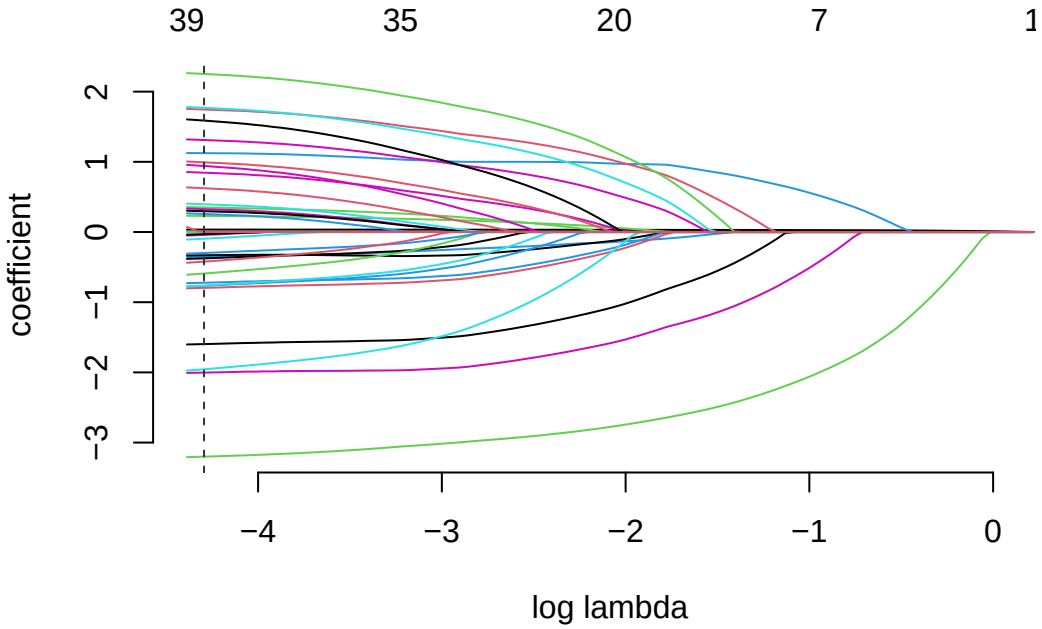
A. Data Preparation and Initial Investigation

Firstly, we eliminate all the null values. Then, 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. As we mention in the EDA section, we perform log transformation on the Installs variable. We result in a much more normal distribution after transformation.



B. Analysis

Lasso We start with using Lasso model using dataset we prepare, and select the lambda using AICc. The lasso selected 37 explanatory variables and rejected 7 variables.




```
##
## gaussian gamlr with 44 inputs and 100 segments.
```

Table 2: Highest and Lowest Coefficients from LASSO Model

Feature	Coefficient	Impact
CategoryENTERTAINMENT	2.2522446	Positive
CategoryEDUCATION	1.7695382	Positive
CategoryPHOTOGRAPHY	1.7456409	Positive
CategoryWEATHER	1.5878266	Positive
CategorySHOPPING	1.3106124	Positive
CategoryFINANCE	-0.7959939	Negative
CategoryBUSINESS	-1.5968459	Negative
CategoryEVENTS	-1.9552108	Negative
CategoryMEDICAL	-2.0030610	Negative
TypePaid	-3.2005830	Negative

The LASSO model analysis reveals distinct patterns regarding the impact of app categories and monetization strategies on app performance within the Google Play Store. Categories such as Entertainment,

Education, Photography, Weather, and Shopping positively influence app performance, indicating high user engagement or downloads, with Entertainment apps showing the highest positive effect.

Conversely, Medical, Events, Business, and Finance apps demonstrate negative impacts, suggesting challenges in user acceptance or market competition, particularly for Medical apps, which show the most significant negative influence.

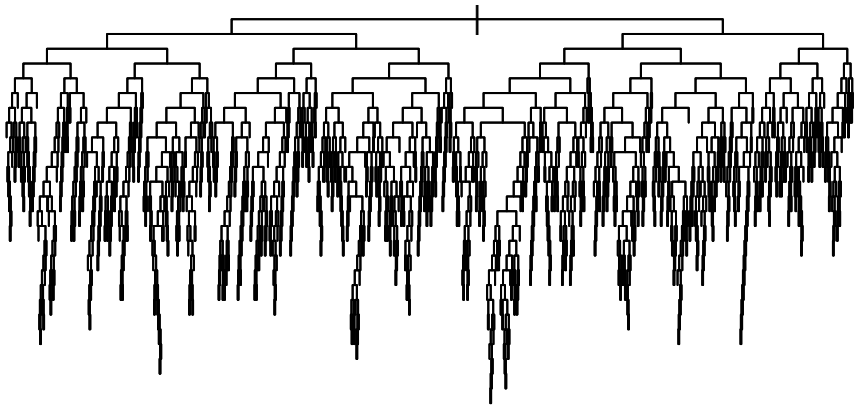
The model also highlights a strong negative effect associated with paid apps, suggesting a distinct user preference for free apps, likely due to hesitancy to incur upfront costs without guaranteed value.

These insights provide valuable guidance for app developers and marketers, emphasizing the importance of category choice and the critical impact of pricing strategies on market success.

```
## [1] "In-sample R^2: 0.298913804277731"
```

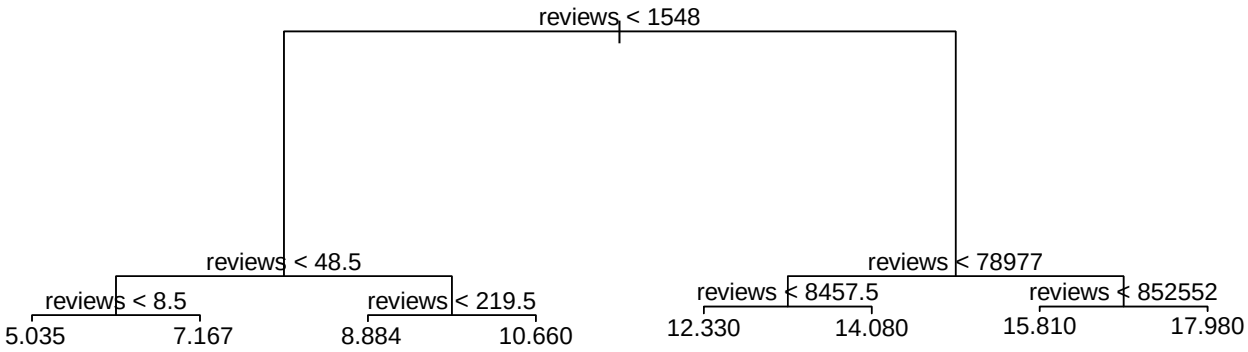
The value of 0.299 suggests a moderate level of explanatory power. This is neither particularly high nor low but indicates that while the model has captured a significant portion of the available explanatory information, there remains a substantial amount of variability that is not explained by the model. Despite not explaining more than half of the variance, the model could still be useful depending on the context and the complexity of the data.

Decision Tree Next, we tried a decision tree model to predict log installs, setting the initial minimum node size to 1 and a minimum deviance requirement of 0.00001 to proceed with a new split.



```
## In-sample R^2: 0.9793926
## Mean Squared Error: 0.2809792
```

The in-sample R^2 value of 0.9793926 and a Mean Squared Error (MSE) of 0.2809792 demonstrate strong performance of the model. An R^2 value close to 1, such as 0.9793926, indicates that the model explains approximately 97.9% of the variance within the training dataset, which points to a very good fit. Similarly, the low MSE corroborates the model's effectiveness, as it signifies a minimal average squared difference between the predicted and actual values, enhancing confidence in the model's predictive accuracy. However, the combination of a high R^2 value and the complex structure of our tree model could lead to concerns about overfitting, where the model might be too closely fitted to the nuances of the training data, potentially limiting its generalizability to new data. Additionally, the structure of the decision tree makes it difficult to interpret, which could obscure meaningful insights from the model. To mitigate these issues and enhance the model's robustness, 50-fold Cross Validation will be performed to appropriately prune the tree, aiming to simplify the model and improve its interpretive ease without sacrificing accuracy.



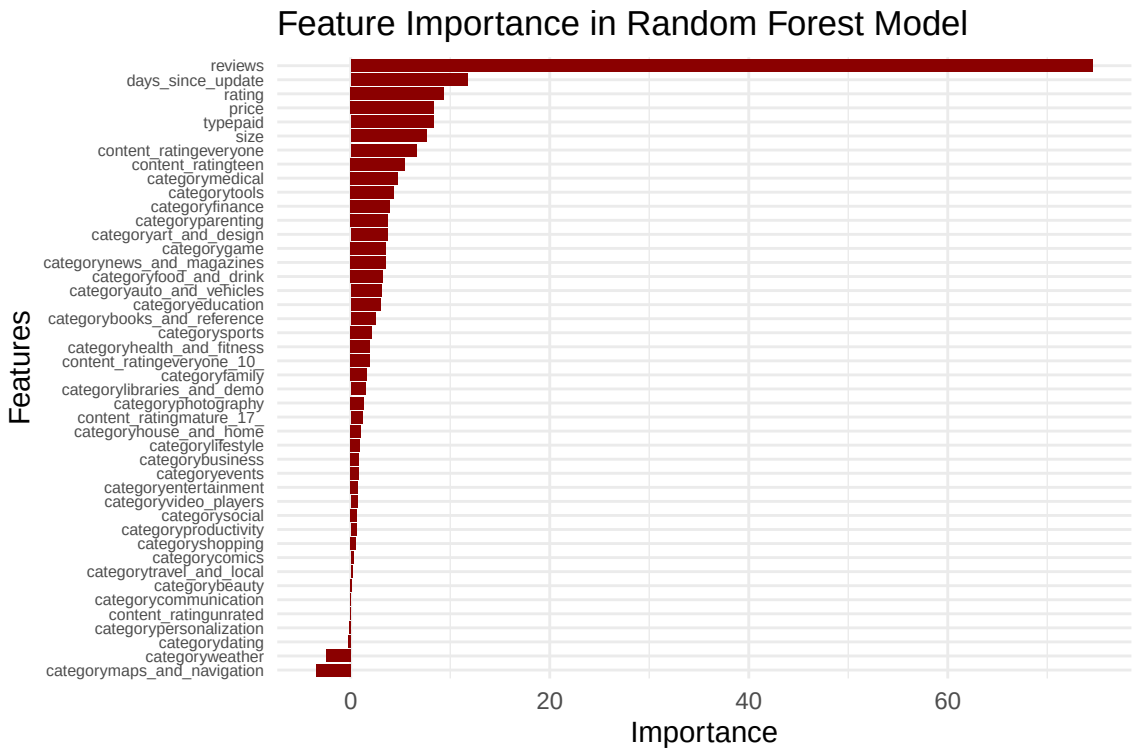
The decision tree's structure, which categorizes outputs based on the number of reviews, underscores the critical role of user reviews in influencing the model's predictions. This not only highlights the importance of this feature but also reflects how critical user engagement metrics are in predicting app performance.

```
## In-sample R^2: 0.8981282
## Mean Squared Error: 1.389012
```

The pruned decision tree model exhibits a strong fit, as indicated by an in-sample R^2 of 0.8981282 from the cross-validation process. This value demonstrates that the model explains approximately 89.81% of the variance in the dependent variable, confirming its effectiveness in capturing the underlying relationships between the predictors and the response variable. However, the Mean Squared Error (MSE) has increased to 1.389012. While this still represents a relatively low error rate, the increase compared to earlier results suggests some variability in predictive accuracy when the model is subjected to different subsets of data.

This could be indicative of a slight overfitting to the training data, where the model performs exceptionally well on training data but less consistently on unseen data.

Random Forest While the tree model is useful for interpretation, it can be improved upon when it comes to prediction with a random forest. A 50 tree forest yields the following important factors:



Similarly with decision tree, the random forest model reveals that the number of reviews is the most significant determinant, indicating that user engagement, as measured by review volume, plays a pivotal role in app success. This suggests that apps with higher review counts likely see greater visibility and popularity, which significantly impacts the model's predictions.

Following reviews, the 'days_since_update' feature stands out as the second most important factor, highlighting the importance of recent updates in app performance. This feature's prominence suggests that apps regularly updated with new features or bug fixes tend to be favored by users, reflecting ongoing development commitment and app reliability.

Other notable features include app size, user ratings, and whether an app is free or paid. These aspects moderately influence app performance, with size and ratings likely affecting user download and retention decisions, and the app's type (paid or free) reflecting user purchasing behavior. Additionally, pricing and

specific content ratings like 'everyone' and 'teen' show varying degrees of impact, indicating differences in target demographics and their preferences.

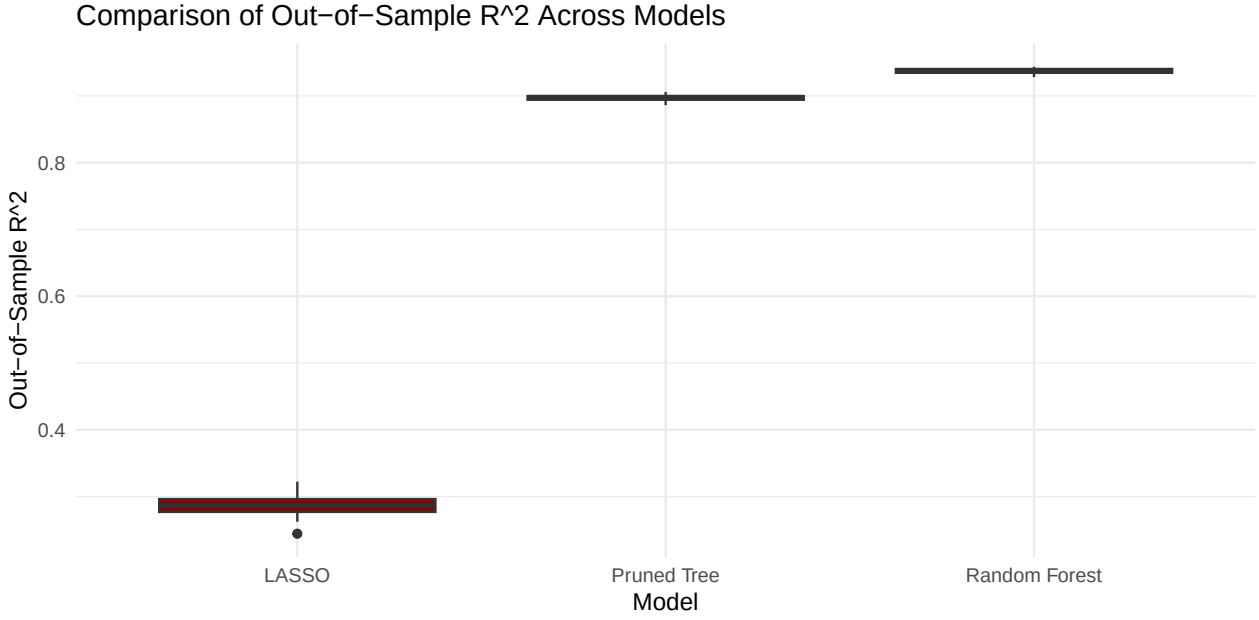
The importance of various app categories such as 'finance', 'medical', 'sports', and 'game' also varies, which may reflect distinct market dynamics, user base sizes, and usage patterns inherent to each category. This differentiation underscores the need for developers to consider category-specific strategies when designing and marketing their apps.

```
## In-sample R^2: 0.9825123
## Mean Squared Error: 0.2384428
```

The results for the Random Forest model, featuring an in-sample R^2 value of 0.9829158 and a Mean Squared Error (MSE) of 0.2329418, illustrate its high predictive performance and accuracy.

C. Conclusion

To evaluate the predictive performance of our three models, we divided our complete dataset into training and test sets using an 80:20 split. We then trained each model—LASSO, Pruned Decision Tree, and Random Forest—on the training data and assessed them on the test data to calculate the out-of-sample R^2 score. To minimize the effects of randomness, we repeated this process 20 times and collected the out-of-sample R^2 scores for each model. Our findings are summarized in the boxplot below:



The boxplot shows that the LASSO model consistently achieves the lowest R^2 scores among the three models. Its median is below 0.2, indicating that it explains less than 20% of the variance in the dataset when used for prediction. The interquartile range (IQR) is tight, suggesting that the model's performance is quite stable across different splits of the data, but unfortunately, it is consistently low. The Pruned Decision Tree model has a slightly higher median R^2 score than the LASSO, hovering around 0.3, and a slightly wider IQR, which suggests more variability in performance. This increase in variability could be due to the model structure or the specific splits of the dataset. The Random Forest model outperforms both LASSO and the Pruned Decision Tree by a significant margin. Its median R^2 is close to 0.8, indicating it explains about 80% of the variance in the test data. The IQR is also relatively narrow, suggesting that this model performs robustly across different data splits. The whiskers, which extend close to the top of the plot, indicate that the lowest performance is still relatively high.

Overall, the Random Forest model is the best performer in terms of out-of-sample predictive ability. Its high R^2 values suggest that it is the most reliable model for predicting the outcome variable.

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

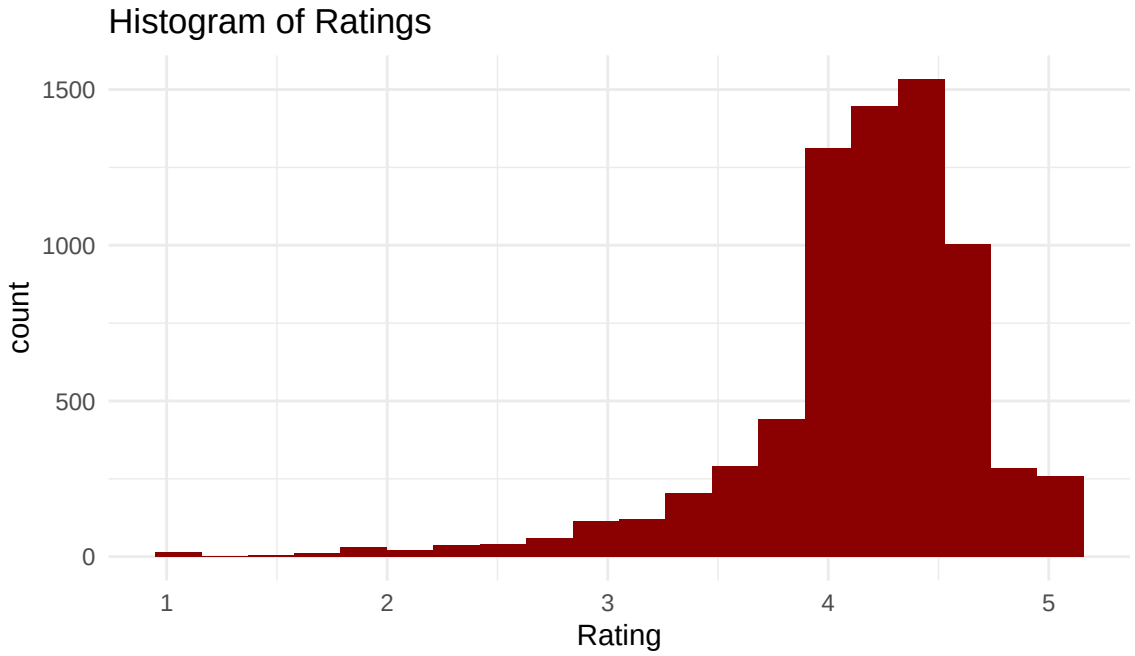
App developers and key stakeholders often have an app's rating as an objective as this influences public perception of the app as well as attracting possible new users. So we decided to investigate our ability to predict an app's rating.

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).



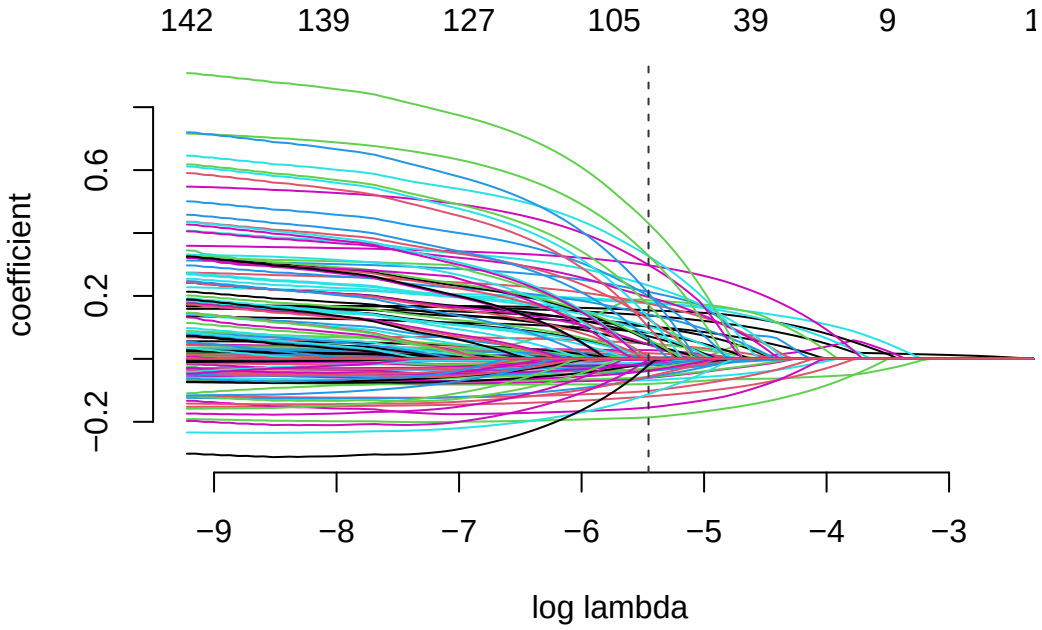
By looking at the histogram of the ratings, it is interesting to observe that the greatest density of ratings is between 4 and 5, which suggests that most users tend to leave a positive rating. Surprisingly the distribution is very heavily skewed to the left, so there are very few ratings close to 1.

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 3: 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.

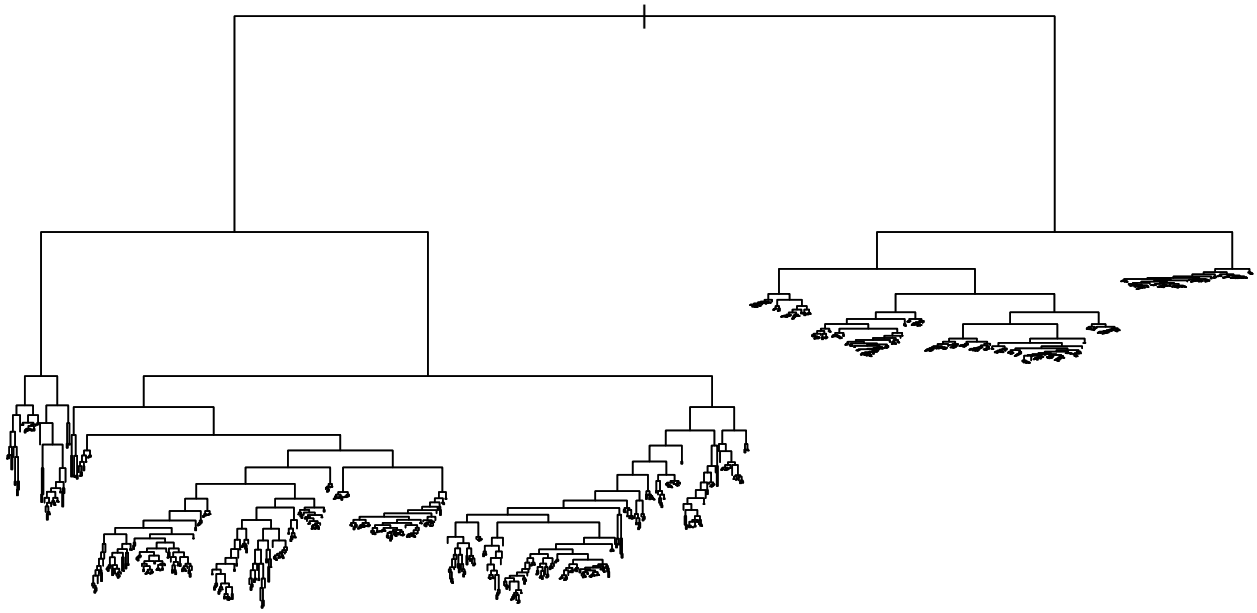
Finally, the number of installs also seems to be a feature of note (log_Installs) as it somewhat counter-intuitively negatively impacts ratings. This might be explained by the idea that more popular apps are used by a broader audience with diverse expectations, and thus receive more critical reviews.

```
## [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.

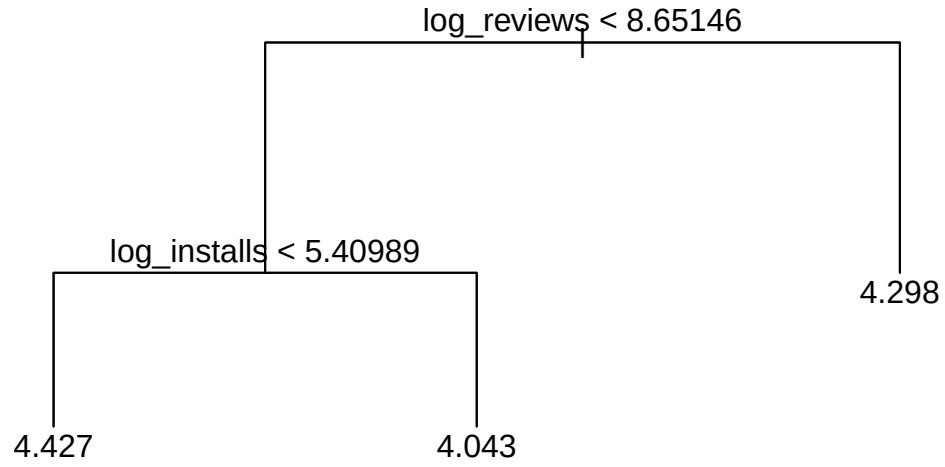


```
## In-sample R^2: 0.5576114
```

```
## 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.

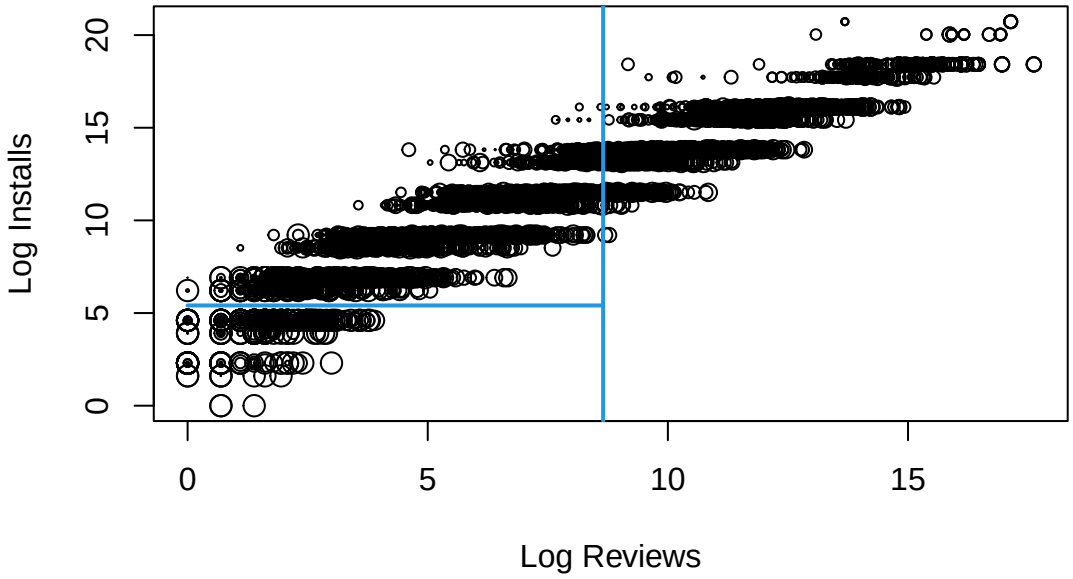
It is interesting to note that having more installs does not necessarily lead to a higher rating. As we see that in the left branch, having log_installs < 5.40989 (equivalent to a threshold of < 223.607 installs), leads to a higher predicted rating than otherwise. This aligns with our finding in the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

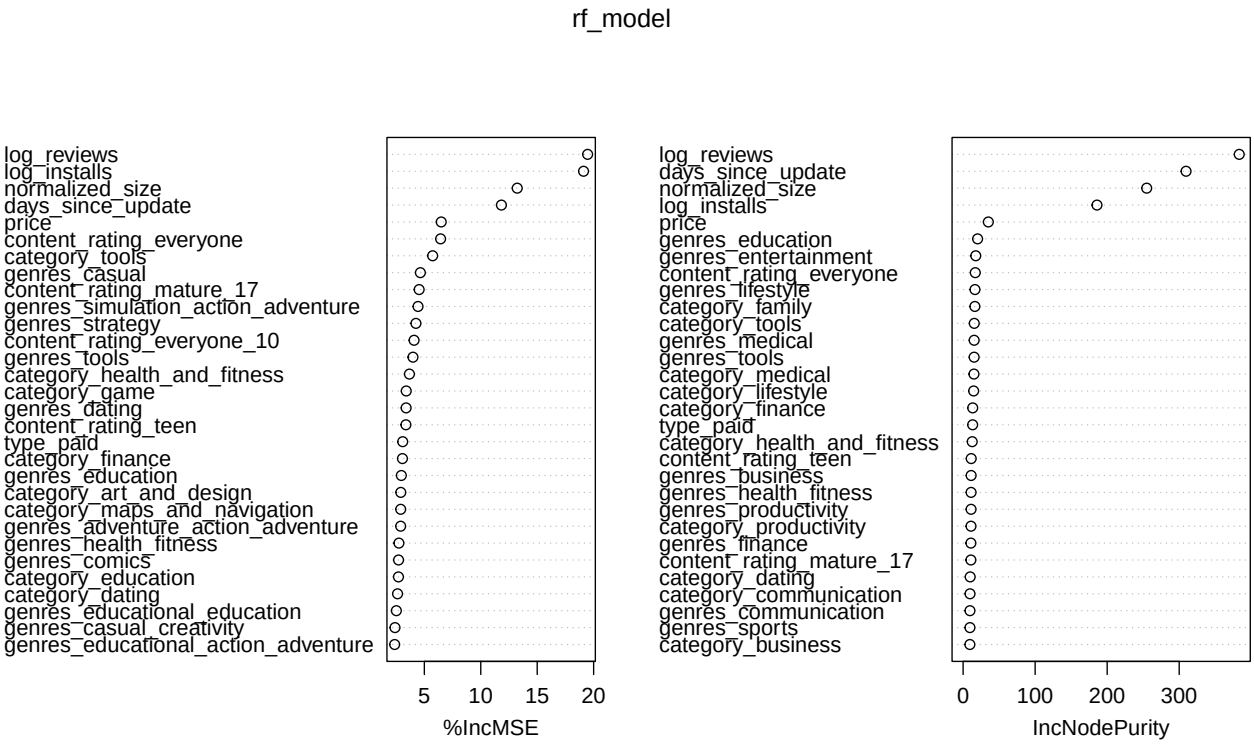
```
## In-sample R^2: 0.7530393
## Mean Squared Error: 0.07487186
```

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 4: Variable Importances from Random Forest

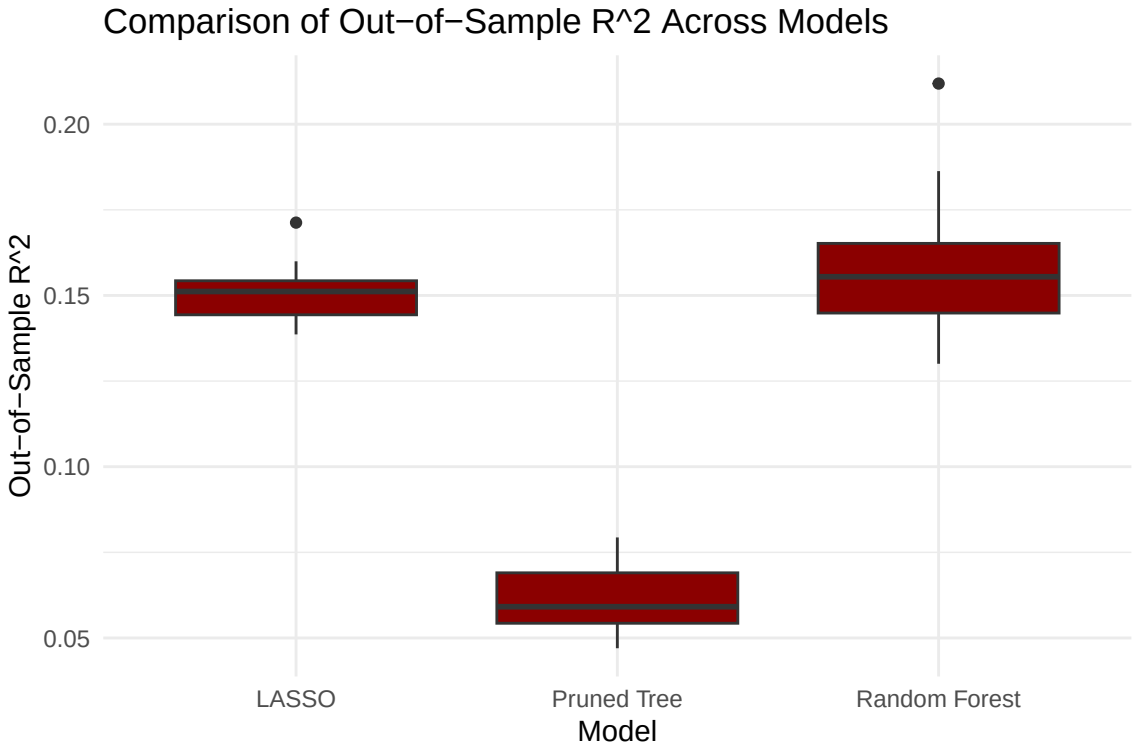
	%IncMSE	IncNodePurity
log_reviews	19.4674421	383.351639
normalized_size	13.2304531	254.862059
log_installs	19.1053739	185.868775
price	6.4970255	35.013352
days_since_update	11.8290876	309.536994
category_art_and_design	2.9151620	1.425252
category_auto_and_vehicles	0.4783673	4.486800
category_beauty	-0.7259967	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. 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?

User sentiment in reviews plays a pivotal role in determining an app's rating, significantly influencing public perception and potential users' decisions. Understanding the intricate relationship between user sentiment and app ratings not only aids in enhancing the app's features but also serves as a crucial gauge of user satisfaction and engagement. Consequently, our investigation centers on intricately examining how sentiment expressed in user reviews, measured through factors such as sentiment polarity and subjectivity, correlates with the overall app ratings. This analytical approach offers valuable insights into user preferences and pain points, empowering developers to refine their applications to better align with user expectations and elevate the overall user experience. Through meticulous analysis of sentiment data, our objective is to uncover discernible patterns that can forecast the impact of user reviews on app ratings, thereby providing strategic guidance for improvements and marketing endeavors.

A. Introduction

In our examination of the `googleplaystore_user_reviews` dataset, we've curated relevant data to delve into the dynamics among `reviews`, `sentiment`, `sentiment_polarity`, and `sentiment_subjectivity`. Firstly, we'll analyze the dataset to identify the top 20 words that have the most significant impact on sentiment, exploring their effects within the categories of `positive`, `neutral`, and `negative` sentiment. Secondly, we'll investigate the correlations among `sentiment`, `sentiment_polarity`, and `sentiment_subjectivity`, conducting analyses on subsets of data categorized by sentiment to delve deeper into these relationships. Lastly, in our third model, we plan to introduce sentiment analysis packages to recalibrate the emotional value of each review. By leveraging this numerical representation, we'll enhance the dataset and treat it as a variable to further explore the relationships among `sentiment`, `sentiment_polarity`, and `sentiment_subjectivity`. Utilizing the `glm` function, we'll determine relevant coefficients and provide comprehensive explanations, aiming to gain a more holistic understanding of the dynamic nature of sentiment in reviews.

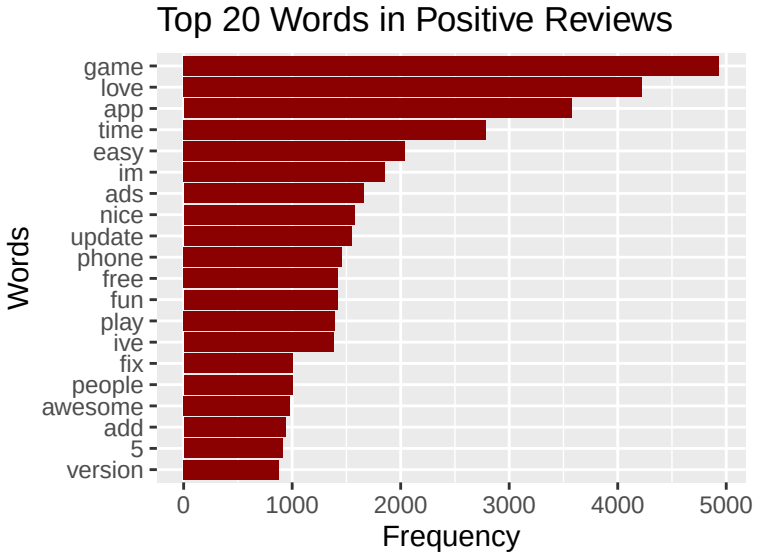
B. Analysis

Top 20 Words in Each Sentiment

By preprocessing the text and extracting the most frequent words associated with different sentiments (positive, neutral, and negative), we aim to gain insights into the overall sentiment distribution and key themes present in the reviews. This analysis is valuable for understanding customer perceptions and experiences with the Google Play Store apps. Additionally, visualizing the top words for each sentiment category provides a clear and concise representation of the most common sentiments expressed by users. We anticipate observing distinct patterns and trends in the words used across different sentiment categories, which can inform app developers and stakeholders about areas for improvement and potential strengths of their apps. Ultimately, this analysis helps to enhance our understanding of user sentiment and preferences, enabling us to make data-driven decisions to optimize app performance and user satisfaction.

Positive Sentiment

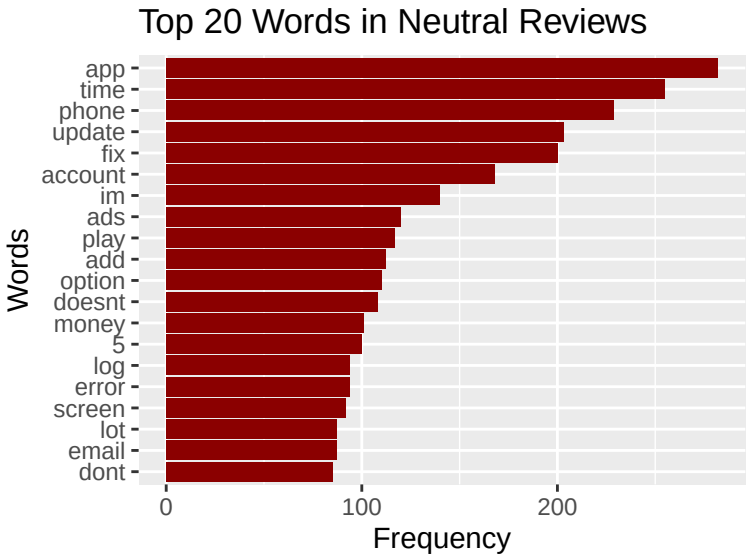
The bar chart representing positive app reviews highlights prevalent themes of enjoyment and functionality, with words like `game, love, fun,` and `easy` indicating user satisfaction and pleasure. Additionally, terms such as `update` and `phone` suggest positive feedback on app updates and compatibility, enhancing user satisfaction. These insights imply that positive reviews often stem from users finding engaging or useful features in the apps, particularly in gaming or functional applications.



Neutral Sentiment

In neutral reviews, practical concerns and minor issues are apparent, as indicated by words like `update,`

`account`, and `option`. The presence of terms such as `money` and `ads` suggests neutrality based on app monetization strategies or advertising presence, which neither significantly pleases nor annoys users. These reviews often focus on specific features or issues needing improvement but are not necessarily dissatisfying enough to be negative.



Negative Sentiment

Negative app reviews center around frustration and problems, with words like `bad`, `annoying`, and `worst` expressing dissatisfaction. Terms such as `update`, `ads`, and `money` indicate negative experiences, likely related to intrusive ads, poor updates, or perceived poor value for money. The presence of `fix` suggests users are encountering bugs or glitches that negatively impact their app experience.

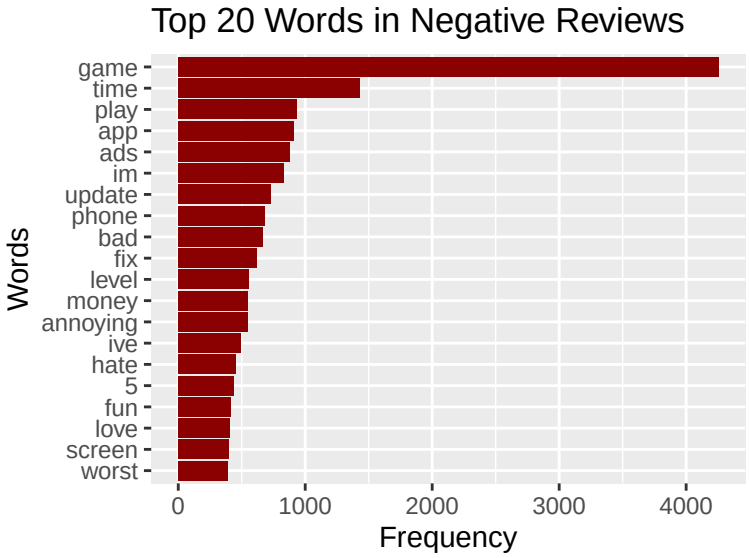


Table 5: Summary Statistics for Different Sentiment Categories

Row	Positive.Sentiment.Polarity	Positive.Sentiment.Subjectivity	Neutral.Sentiment.Polarity	Neutral.Sentiment.Subjectivity	Negative.Sentiment.Polarity	Negative.Sentiment.Subjectivity
Min.	0.0000000	0.0000000	0	0.0000000	-1.0000000	0.0000000
1st Qu.	0.1666667	0.4500000	0	0.0000000	-0.3645833	0.4000000
Median	0.3300000	0.5642857	0	0.0000000	-0.1833333	0.5180952
Mean	0.3724021	0.5669494	0	0.0797091	-0.2561728	0.5350868
3rd Qu.	0.5000000	0.6750000	0	0.0000000	-0.0812500	0.6666667
Max.	1.0000000	1.0000000	0	1.0000000	0.0000000	1.0000000

Regression Analysis

Based on exploratory data analysis (EDA), we can observe the following insights. Firstly, positive and negative comments tend to be more subjective, as they may stem from personal experiences and emotions, while neutral comments are generally more objective, focusing on features without strong emotional content. Secondly, developers and marketers can glean guidance from these charts; understanding polarity helps identify the pros and cons of comments, guiding customer service responses, while comprehending subjectivity aids in distinguishing types of feedback, guiding product improvements and feature development. To better ascertain the effectiveness of these comments, it's essential to probe `sentiment_polarity` and `sentiment_subjectivity` within each sentiment category.

The coefficient for Sentiment_Subjectivity is 0.601763, indicating a strong positive relationship with Sentiment_Polarity. This suggests that as subjectivity in positive reviews increases, the polarity also tends to be more positive. The model explains about 21.32% of the variance in sentiment polarity (Adjusted R-squared: 0.2132), which is moderate but significant, indicating that while other factors also affect polarity, subjectivity is an important predictor.

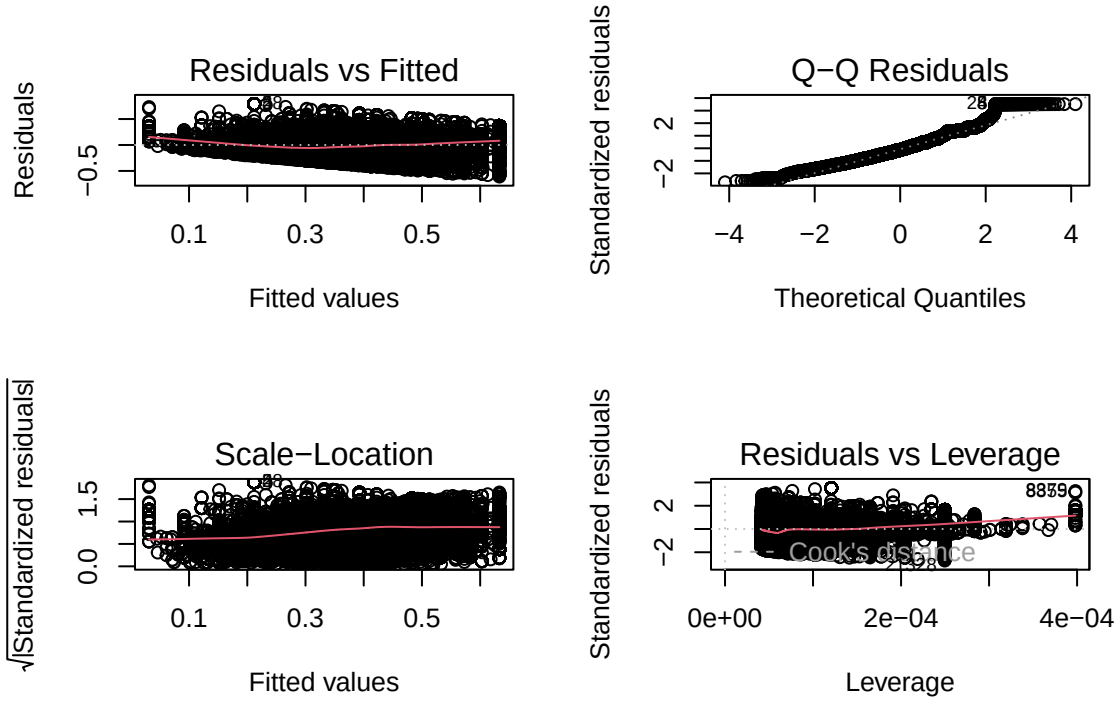
```
Call:
lm(formula = Sentiment_Polarity ~ Sentiment_Subjectivity, data = data_positive)

Residuals:
     Min       1Q   Median       3Q      Max
-0.60800 -0.16049 -0.03167  0.12806  0.78824

Coefficients:
                       Estimate Std. Error t value Pr(>|t|)
(Intercept)            0.031233   0.004472   6.985 2.93e-12 ***
Sentiment_Subjectivity 0.601763   0.007463  80.633  < 2e-16 ***
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2241 on 23996 degrees of freedom
Multiple R-squared:  0.2132,    Adjusted R-squared:  0.2132
F-statistic:  6502 on 1 and 23996 DF,  p-value: < 2.2e-16
```

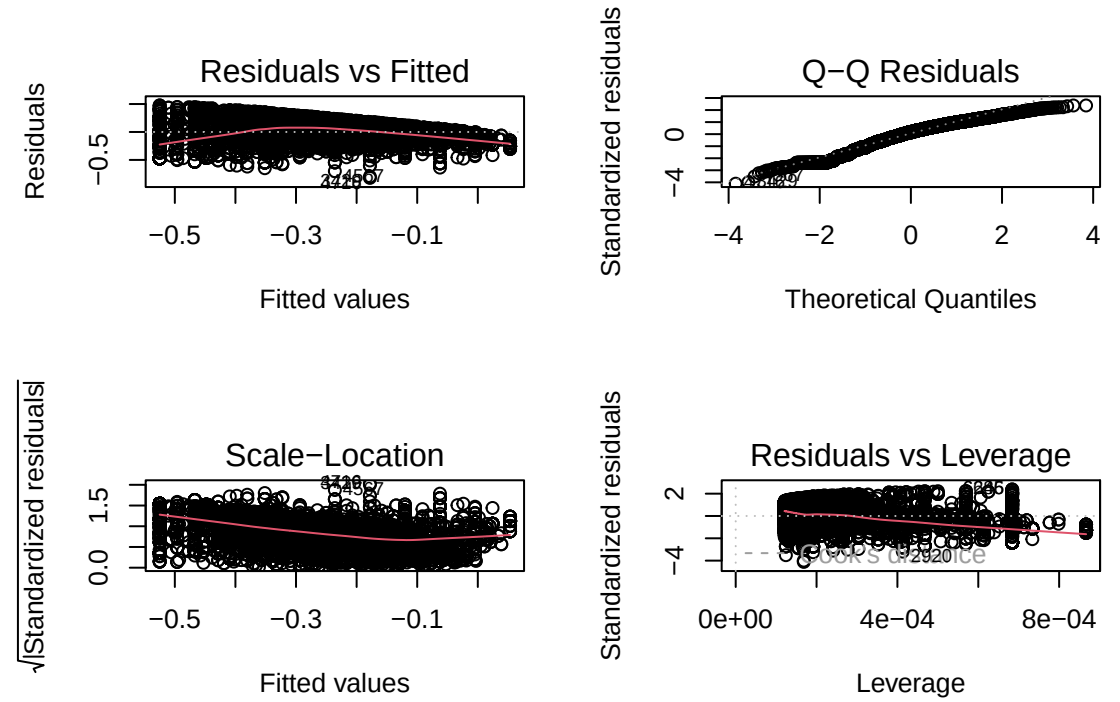
The coefficient for Sentiment_Subjectivity is -0.578264, showing a significant negative relationship with Sentiment_Polarity in negative reviews. This suggests that higher subjectivity in negative reviews correlates with more negative polarity. The model explains about 27.99% of the variance in sentiment polarity (Adjusted R-squared: 0.2799), which is somewhat higher than in positive reviews, indicating a slightly stronger influence of subjectivity on polarity in negative contexts.

```
Call:
lm(formula = Sentiment_Polarity ~ Sentiment_Subjectivity, data = data_negative)

Residuals:
     Min       1Q   Median       3Q      Max
-0.82194 -0.12194  0.03242  0.14648  0.47502

Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)             0.053249   0.005883   9.052   <2e-16 ***
Sentiment_Subjectivity -0.578264   0.010198 -56.704   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1998 on 8269 degrees of freedom
Multiple R-squared:   0.28, Adjusted R-squared:  0.2799
F-statistic:  3215 on 1 and 8269 DF,  p-value: < 2.2e-16
```

Positive Sentiment Model Plots:



Negative Sentiment Model Plots:



Residuals vs Fitted Plots: Both models show some patterns in residuals, which could indicate non-linearity or other underlying effects not captured by the model. There's no apparent funnel shape, which suggests homoscedasticity (constant variance of errors).

Q-Q Plots: The plots for both models indicate some deviation from normality, especially at the tails. This suggests that the residuals are not perfectly normally distributed, which could impact inference from the model.

Scale-Location Plots: The spread of residuals in these plots is fairly constant, supporting the assumption of equal variances across fitted values.

Residuals vs Leverage Plots: There are a few influential points, as indicated by Cook's distance lines, but they do not appear to unduly influence the model's overall fit.

The linear models for both positive and negative sentiments show that subjectivity is a significant predictor of sentiment polarity, with subjectivity having a positive impact in positive reviews and a negative impact in negative reviews. The diagnostic checks reveal some concerns with normality and potential influence from a few data points, but overall, the models are robust and provide valuable insights.

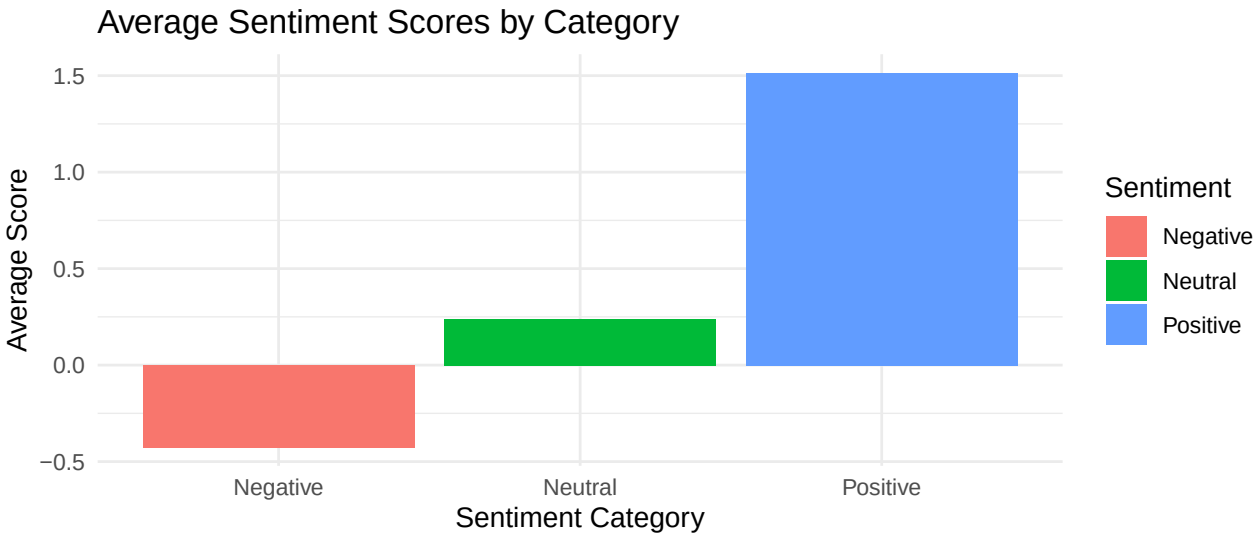
Given the evidence of some non-linearity and potential outliers, it may be beneficial to explore more complex models or transformations of variables. Non-linear models or the inclusion of interaction terms could better capture the underlying dynamics of the data. Additionally, considering the substantial proportion of variance unexplained by the models, incorporating additional predictors could enhance the models' explanatory power.

Overall, these findings highlight the nuanced role of subjectivity in shaping user perceptions expressed in reviews. By acknowledging these dynamics, businesses can better understand the underpinnings of user feedback and refine their strategies for addressing user concerns and enhancing overall user experience.

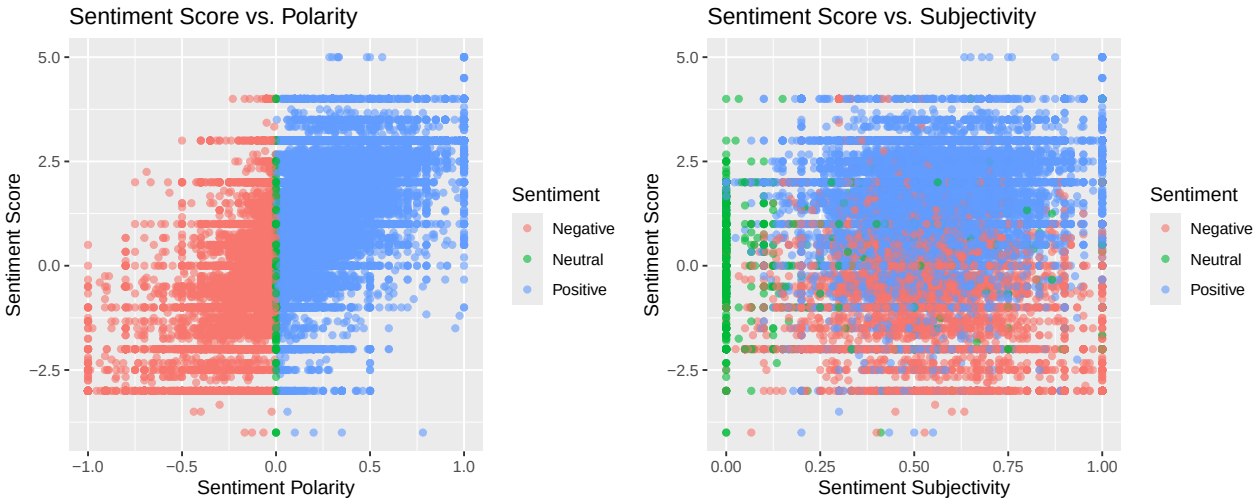
Enumerating Sentiment Scores

Enumerating sentiment scores from text is a crucial method that converts subjective elements of language into numerical data, facilitating a deeper and more structured understanding of textual content. This approach enables quantifying sentiment trends over time, comparing sentiment across different groups, and tracking changes in response to events. By computing sentiment scores, we engage in quantitative analysis, enhancing decision-making processes, improving products, adjusting marketing strategies, and elevating customer service based on feedback sentiment. Furthermore, the automation and scalability of sentiment

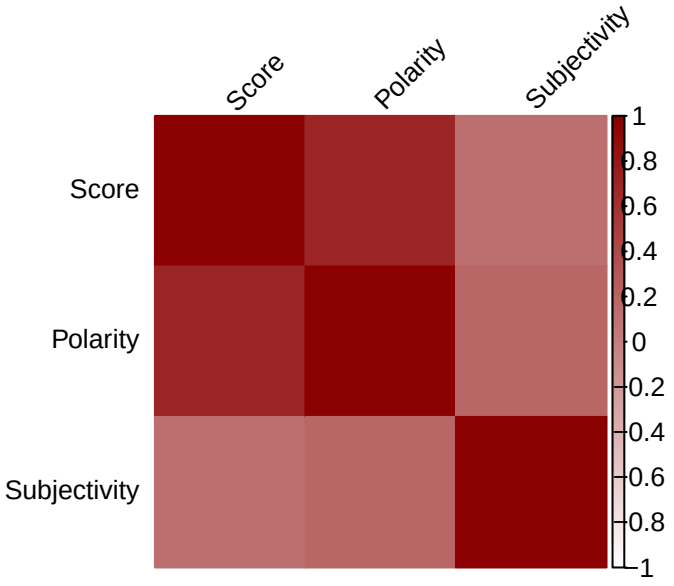
scoring enable efficient handling of large text volumes, benefiting applications like social media monitoring, market research, and customer feedback analysis. Additionally, sentiment scoring captures subtle nuances in sentiment expression that may be overlooked in broader analyses, and analyzing scores at the sentence level provides contextual insights, leading to more accurate interpretations of overall sentiment trends.



The presented plot serves as a comprehensive visualization illustrating the average sentiment scores across different sentiment categories, namely Negative, Neutral, and Positive. Through this visual representation, we glean valuable insights into the nuanced sentiment dynamics inherent within each category. Notably, positive reviews emerge with a notably elevated average score, indicative of the predominantly favorable sentiments expressed within this segment. In contrast, neutral reviews gravitate towards a sentiment score hovering around the neutral midpoint, suggesting a balance between positive and negative sentiments or a lack of strong emotional expression. Conversely, negative reviews exhibit an average score situated below zero, underscoring the prevalence of adverse sentiments within this segment. This graphical depiction provides compelling evidence of the efficacy of the sentiment scoring mechanism in accurately discerning and quantifying the diverse spectrum of sentiments encapsulated within each category, thereby offering a robust foundation for subsequent analytical endeavors and informed decision-making processes.



These scatter plots serve to illustrate the connection between computed sentiment scores and their corresponding polarity and subjectivity. Across various levels of polarity and subjectivity, the plots reveal a diverse distribution of sentiment scores. Notably, concerning polarity, a discernible correlation emerges, indicating that elevated polarity tends to align with more positive sentiment scores, while diminished polarity correlates with more negative sentiment scores. However, when examining subjectivity, the relationship appears less defined, implying that subjectivity alone may not wield significant predictive power over sentiment scores. These visualizations offer valuable insights into the nuanced interplay between sentiment attributes, informing a deeper comprehension of sentiment dynamics within the dataset.



The heatmap provides a comprehensive overview of the correlations among sentiment score, sentiment polarity, and sentiment subjectivity, indicating the strength of their relationships through varying color intensities. Notably, it illustrates a significant positive correlation between sentiment score and sentiment polarity, suggesting that as polarity becomes more positive, sentiment scores tend to increase accordingly. Conversely, the correlation between sentiment score and sentiment subjectivity appears weaker, implying that while subjectivity influences sentiment, its impact on overall sentiment scores may be less pronounced. This visualization offers valuable insights into the complex interplay of sentiment attributes, aiding in a deeper understanding of sentiment dynamics within the dataset and guiding more informed decision-making processes.

In employing the linear regression model, we first identify a set of predictor variables that we believe may influence the outcome variable of interest, which in this case is sentiment polarity. These predictors could include sentiment scores, subjectivity levels, and other relevant features extracted from textual data. We then train the model using a dataset where sentiment polarity is known, allowing the model to learn the relationship between the predictors and the target variable. Once trained, we validate the model's performance using separate validation data, ensuring its ability to generalize to unseen data. Through this process, we gain insights into how different factors contribute to sentiment polarity, enabling us to understand the nuanced dynamics of sentiment expression and potentially informing strategies for sentiment modification or more refined sentiment analysis.

```
Call:
lm(formula = Sentiment_Polarity ~ Sentiment_Subjectivity + sentiment_score,
    data = data_positive_scored)

Residuals:
     Min       1Q   Median       3Q      Max
-0.64553 -0.13105 -0.01642  0.11382  0.90340

Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)            -0.037624   0.004903  -7.674 1.76e-14 ***
Sentiment_Subjectivity  0.465757   0.008131  57.281  < 2e-16 ***
sentiment_score         0.079350   0.001236  64.177  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1967 on 16557 degrees of freedom
Multiple R-squared:  0.3635,    Adjusted R-squared:  0.3634
F-statistic:  4728 on 2 and 16557 DF,  p-value: < 2.2e-16
```

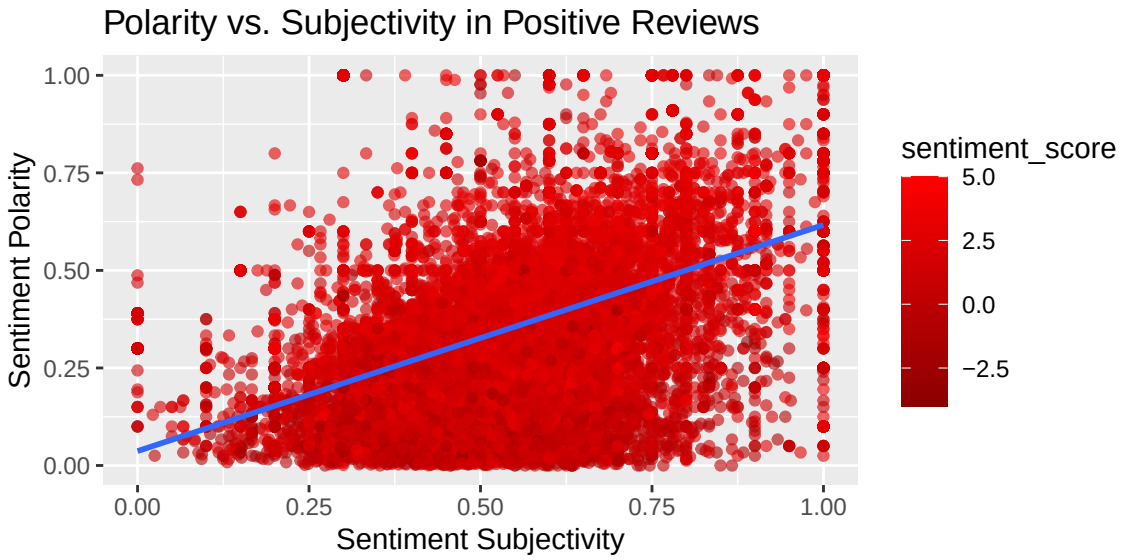
The linear model for positive sentiments reveals a strong positive correlation between sentiment subjectivity and polarity, suggesting that more subjective positive reviews tend to exhibit stronger positive

sentiments. This association is reinforced by the significant positive coefficient for sentiment scores, indicating their contribution to higher polarity values. The model's explanatory power, represented by the Adjusted R-squared of 0.3633, implies that approximately 36.33% of the variation in sentiment polarity is accounted for by the predictors. Both Sentiment_Subjectivity and sentiment_score play significant roles, with positive coefficients of 0.465701 and 0.079301, respectively, affirming their influence on sentiment polarity. Despite the model's overall goodness of fit, potential outliers or high-leverage points suggest the need for further robustness checks or alternative model considerations to enhance accuracy and interpretation.

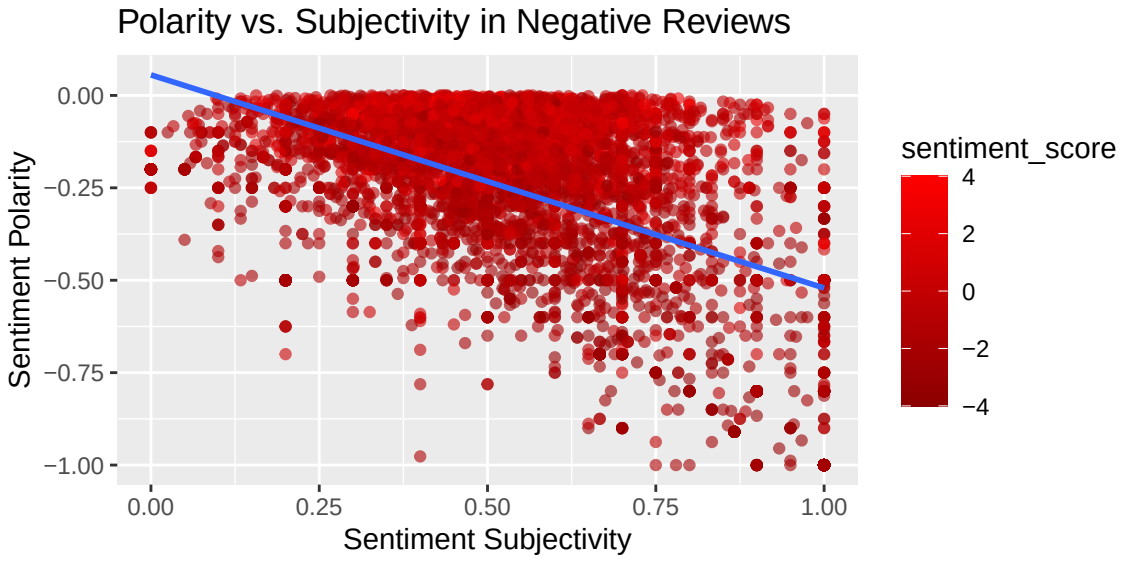
```
Call:
lm(formula = Sentiment_Polarity ~ Sentiment_Subjectivity + sentiment_score,
    data = data_negative_scored)

Residuals:
     Min       1Q   Median       3Q      Max
-0.73738 -0.10602  0.02464  0.12806  0.48040

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)
(Intercept)              0.052021   0.006581   7.905 3.19e-15 ***
Sentiment_Subjectivity -0.490074   0.011463 -42.752  < 2e-16 ***
sentiment_score          0.063452   0.001638  38.739  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1811 on 5626 degrees of freedom
Multiple R-squared:  0.4232,    Adjusted R-squared:  0.423
F-statistic:  2064 on 2 and 5626 DF,  p-value: < 2.2e-16
```

The negative sentiment model reveals a negative coefficient (-0.491287) for sentiment subjectivity, indicating an inverse relationship between subjectivity and sentiment polarity in negative reviews; more subjective negative reviews tend to have less negative polarity. Conversely, the coefficient for sentiment scores is positive (0.063526), suggesting that even within negative reviews, higher usage of negative words (higher sentiment

scores) correlates with less negative polarity. The model exhibits slightly higher explanatory power than the positive sentiment model, with an adjusted R-squared value of 0.4239, indicating that approximately 42.39% of the variance in sentiment polarity is explained by the predictors. This linear model uncovers an intriguing negative relationship between sentiment subjectivity and polarity in negative sentiments, suggesting that more subjective negative reviews often have less negative polarity, perhaps reflecting nuanced criticism or emotional content that does not translate directly into straightforward negativity. The positive coefficient for sentiment scores aligns with this trend, as lower negative scores correlate with higher polarity values. The model explains a substantial portion of the variability, as indicated by the R-squared value.



This scatter plot visually reinforces the findings from the linear model, displaying a clear positive trend between sentiment subjectivity and polarity among positive reviews. The density of points increases as subjectivity and polarity rise, supporting the notion that highly subjective positive reviews are often more emphatically positive. The color gradient representing sentiment scores shows that higher scores generally coincide with higher polarity, aligning with the model's coefficients. This plot effectively illustrates the positive relationship and provides a visual confirmation of the statistical findings, highlighting the consistency across different data representation methods.



Contrasting with the positive reviews, the scatter plot for negative reviews shows a negative trend, where increased subjectivity correlates with higher polarity, suggesting a decrease in negativity. This plot visually supports the negative coefficient for subjectivity found in the linear model and illustrates how subjective negative reviews often express criticisms that are less harsh in terms of sentiment polarity. The plot also reveals a broad spread of sentiment scores across different levels of subjectivity, suggesting a complex interaction between these factors in shaping the sentiment expressed in negative reviews. This complexity is crucial for understanding how different aspects of sentiment interact in user feedback and underscores the need for nuanced analysis in handling negative reviews.

C. Conclusion

Across all sentiments, common words like “app,” “update,” and “phone” indicate their central role in user reviews, regardless of sentiment. Positive reviews tend to emphasize enjoyment and utility, neutral reviews discuss functionality and minor concerns, while negative reviews highlight faults and negative experiences. Developers can prioritize areas for improvement and enhance user satisfaction by understanding these themes, adjusting features, or strategies based on the most frequent criticisms or praises. This underscores the importance of monitoring user feedback to drive better user experiences and app development.

In the basic regression model, the analysis of the Positive Sentiment Linear Model reveals a significant positive correlation between Sentiment_Subjectivity and Sentiment_Polarity, indicating that increased subjectivity in positive reviews generally correlates with more positive expressions of sentiment. This model accounts for approximately 21.32% of the variance in sentiment polarity, suggesting that while subjectivity

is notable, other variables also influence sentiment in positive reviews. Conversely, the Negative Sentiment Linear Model demonstrates a significant negative relationship between Sentiment_Subjectivity and Sentiment_Polarity, suggesting that higher subjectivity in negative reviews tends to be associated with more intensely negative sentiments. This model explains about 27.99% of the variance in sentiment polarity, slightly higher than in the positive model, indicating a stronger influence of subjectivity on sentiment polarity in negative reviews. These findings underscore the importance of subjectivity in shaping sentiment expressions, with its impact manifesting differently across positive and negative contexts, offering valuable insights for businesses seeking to understand and respond to customer feedback effectively.

Moreover, by introducing a new parameter, sentiment scores, and examining the relationship among the variables, the analysis unveils distinct dynamics in how subjectivity and computed sentiment scores influence the polarity of positive versus negative reviews. Positive reviews exhibit a reinforcing effect, where both subjectivity and positive sentiment scores push polarity higher. Conversely, in negative reviews, increased subjectivity tends to mitigate the negativity reflected in the polarity scores, a counterintuitive yet crucial aspect for comprehending nuances in customer feedback. These insights are critical for businesses aiming to effectively parse through customer sentiments to enhance service quality or product offerings. The differences in model fits and residual patterns also prompt careful consideration of potential data issues like outliers or influential points, which could impact the robustness and interpretation of the models.

8. Conclusion and Improvements

In this comprehensive analysis of the Google Play Store dataset, we have explored the multifaceted relationships between app characteristics, user sentiment, and market performance. Our findings illuminate the profound impact of user reviews and app updates on app success metrics such as install counts and ratings. Through rigorous statistical modeling, including techniques like LASSO, Decision Trees, and Random Forest, we've identified key predictors of app success and provided a nuanced understanding of the factors that enhance user engagement and satisfaction.

Key Findings:

- **User Reviews:** The volume and sentiment of user reviews are strongly correlated with app visibility and popularity, underscoring the importance of maintaining high user satisfaction.
- **App Updates:** Frequent updates, which often reflect active development and responsiveness to user feedback, are associated with higher user ratings, suggesting that timely content renewal is critical.
- **Sentiment Analysis:** Positive sentiment in user reviews generally leads to higher ratings, highlighting the need for apps to elicit and sustain positive user experiences.

Recommendations for Improvement:

1. **Deepening Feature Analysis:** To refine the predictive accuracy of our models, further development of feature engineering techniques is recommended. Integrating more detailed user interaction metrics and app performance indicators could uncover deeper insights.
2. **Model Enhancement:** Continued optimization of the existing models, especially through advanced hyperparameter tuning and the exploration of ensemble methods, could improve their predictive strength and reliability.
3. **Expanding Sentiment Metrics:** Enhancing the sentiment analysis framework to capture broader emotional scales could provide a more comprehensive picture of user sentiment, aiding in the nuanced understanding of user feedback.
4. **Demographic Considerations:** Including demographic data in the analysis could reveal essential differences in app preferences and usage patterns across user groups, supporting targeted app development strategies.

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)
library(wordcloud)
library(textdata)
library(tidytext)
library(reshape2)
library(rpart)
library(rpart.plot)
########################################
# 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)) |>
  distinct() # Remove duplicate rows
# 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
p2_p3_row <- arrangeGrob(p2, p3, ncol = 2)
grid.arrange(p2_p3_row, nrow = 1)
########################################
# 5. A. Data Preparation and Initial Investigation
########################################
```

```r
installs_data <- googleplaystore |> na.omit()

# The data is scraped from August 2018
installs_data$Days_Since_Update <- as.numeric(as.Date("2018-08-15")
                                              - installs_data$Last.Updated)

# Log transform Installs, Size, and Reviews to remove skewness
installs_data <- installs_data |>
  mutate(log_Installs = log(installs_data$Installs))

# Create dummy variables using model.matrix
x <- model.matrix(log_Installs ~ Reviews + Size  + Price + Days_Since_Update +
                    Category + Type + Content.Rating + Rating - 1, data = installs_data)
# Response variable
y <- installs_data$log_Installs
ggplot(googleplaystore, aes(x = log(Installs))) +
  geom_histogram(bins = 30, fill = "darkred") +
  ggtitle("Distribution of Log Installs")+
  theme(axis.text.x = element_text(size = 6,angle = 0, hjust = 1, vjust = 0.5))
########################################
# 5. B. i) LASSO
########################################
library(gamlr)
set.seed(1024)
lasso_model <- gamlr(x, y, alpha = 1, nfolds = 100, type.measure = "mse")
plot(lasso_model)
# Find the index with lowest AICc
summary_output = summary(lasso_model)
best_aicc_index <- which.min(summary_output$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$Coefficient, decreasing=TRUE),]
kable(coefficients_df,
      caption = "Highest and Lowest Coefficients from LASSO Model",
      row.names = FALSE) |>
  kable_styling(bootstrap_options = c("striped", "hover"))
print(paste("In-sample R^2:", summary_output$r2[best_aicc_index]))
########################################
# 5. B. ii) Decision Tree
########################################
clean_column_names <- function(data) {
    names(data) <- tolower(names(data))
```

```
    names(data) <- gsub("[^[:alnum:]_]", "_", names(data))
    names(data) <- make.names(names(data), unique = TRUE)
    return(data)
}

x <- clean_column_names(as.data.frame(x))

set.seed(1234)
tree_model <- tree(y ~ ., data = x, mincut = 1, mindev = 0.00001)
plot(tree_model, type="uniform")
evaluate_tree_model <- function(model, x, y) {
    predictions <- predict(model, x)
    SSR <- sum((y - predictions)^2)
    mean_rating <- mean(y)
    SST <- sum((y - mean_rating)^2)
    R_squared <- 1 - (SSR / SST)
    mse <- mean((y - predictions)^2)
    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)
########################################
# 5. B. iii) Random Forest
########################################
set.seed(5678)
rf_model <- randomForest(y ~ ., data=x, importance = TRUE, ntree=50)
load("model_data/installs_rf_model.RData")
importance <- importance(installs_rf_model)  # Get importance
importance_df <- data.frame(Feature = rownames(importance), Importance = importance[,1])
ggplot(importance_df, aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_col(fill = "darkred") +
  labs(title = "Feature Importance in Random Forest Model",
       x = "Features", y = "Importance") +
  theme_minimal() +
  coord_flip()+
  theme(axis.text.y = element_text(size = 6))
evaluate_tree_model(installs_rf_model, x, y)
########################################
```

```r
# 5. 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) {
    # Initial 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"))
```

```r
    # 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/installs_results_df.RData")
# Plot the results
results_long <- pivot_longer(installs_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"))
##########################################
# 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")
```

```
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"))
```

```
########################################
# 7. B. i) Top 20 Words in Each Sentiment
########################################
# Load the dataset
data <- googleplaystore_user_reviews

# Preprocess the text: remove punctuation, convert to lowercase, remove stopwords
data_clean <- data |>
  mutate(Review_Text = tolower(Translated_Review)) |>
  mutate(Review_Text = gsub("[[:punct:]]", "", Review_Text))

# Tokenize the text
data_tokens <- data_clean |>
  unnest_tokens(word, Review_Text)

# Remove stopwords
data_tokens <- data_tokens |>
  anti_join(stop_words, by = "word")

# Function to get top N words for a given sentiment
get_top_words <- function(data, sentiment, n = 20) {
  data |>
    filter(Sentiment == sentiment) |>
    count(word, sort = TRUE) |>
    slice_max(order_by = n, n = n) |>
    arrange(desc(n))
}

# Get top 20 words for each sentiment
top_words_positive <- get_top_words(data_tokens, "Positive")
top_words_neutral <- get_top_words(data_tokens, "Neutral")
top_words_negative <- get_top_words(data_tokens, "Negative")

# Function to plot top words
plot_top_words <- function(top_words, sentiment) {
  ggplot(top_words, aes(x = reorder(word, n), y = n)) +
    geom_bar(stat = "identity", fill = "darkred") +
    coord_flip() +
    labs(title = paste("Top 20 Words in", sentiment, "Reviews"),
         x = "Words",
         y = "Frequency")
}

plot_top_words(top_words_positive, "Positive")
plot_top_words(top_words_neutral, "Neutral")
plot_top_words(top_words_negative, "Negative")
########################################
# 7. B. ii) Regression Analysis
########################################

# Separate the data using filter
data_positive <- data |> filter(Sentiment == "Positive")
data_neutral <- data |> filter(Sentiment == "Neutral")
```

```r
data_negative <- data |> filter(Sentiment == "Negative")

# Summary statistics for positive sentiment
positive_polarity = unclass(summary(data_positive$Sentiment_Polarity))
positive_subjectivity = unclass(summary(data_positive$Sentiment_Subjectivity))

# Summary statistics for neutral sentiment
neutral_polarity = unclass(summary(data_neutral$Sentiment_Polarity))
neutral_subjectivity = unclass(summary(data_neutral$Sentiment_Subjectivity))

# Summary statistics for negative sentiment
negative_polarity = unclass(summary(data_negative$Sentiment_Polarity))
negative_subjectivity = unclass(summary(data_negative$Sentiment_Subjectivity))

# Combine all into a data frame
df_stats <- data.frame(
  Row = c("Min.", "1st Qu.", "Median", "Mean", "3rd Qu.", "Max."),
  "Positive Sentiment Polarity" = unname(positive_polarity),
  "Positive Sentiment Subjectivity" = unname(positive_subjectivity),
  "Neutral Sentiment Polarity" = unname(neutral_polarity),
  "Neutral Sentiment Subjectivity" = unname(neutral_subjectivity),
  "Negative Sentiment Polarity" = unname(negative_polarity),
  "Negative Sentiment Subjectivity" = unname(negative_subjectivity)
)

# Transpose the data frame
transposed_df <- as.data.frame(t(df_stats))
colnames(transposed_df) <- transposed_df[1, ]
transposed_df <- transposed_df[-1, ]

# Output the table
kable(df_stats, format="latex",  booktabs=TRUE,
      caption = "Summary Statistics for Different Sentiment Categories", align = 'c') |>
  kable_styling(latex_options="scale_down")
# Apply linear model for positive sentiment
lm_positive <- lm(Sentiment_Polarity ~ Sentiment_Subjectivity, data = data_positive)
summary(lm_positive)
# Apply linear model for negative sentiment
lm_negative <- lm(Sentiment_Polarity ~ Sentiment_Subjectivity, data = data_negative)
summary(lm_negative)
par(mfrow = c(2, 2))
plot(lm_positive)
par(mfrow = c(2, 2))
plot(lm_negative)
########################################
# 7. B. iii) Enumerating Sentiment Scores
########################################

data_positive <- filter(data, Sentiment == "Positive")
data_neutral <- filter(data, Sentiment == "Neutral")
data_negative <- filter(data, Sentiment == "Negative")

# Load AFINN sentiment lexicon
```

```
afinn <- get_sentiments("afinn")

# Function to process and calculate scores
calculate_scores <- function(df) {
  df |>
    unnest_tokens(word, Translated_Review) |> # Tokenizing the text
    mutate(word = tolower(word)) |>           # Convert words to lower case
    inner_join(afinn, by = "word") |>         # Join with AFINN lexicon
    group_by(Sentiment) |>
    summarize(average_score = mean(value, na.rm = TRUE), .groups = 'drop')}

# Calculate scores for each sentiment dataset
positive_scores <- calculate_scores(data_positive)
neutral_scores <- calculate_scores(data_neutral)
negative_scores <- calculate_scores(data_negative)

# Combining all scores into one data frame for plotting
all_scores <- bind_rows(
  positive_scores |> mutate(Sentiment = "Positive"),
  neutral_scores |> mutate(Sentiment = "Neutral"),
  negative_scores |> mutate(Sentiment = "Negative")
)
# Plotting the average sentiment scores
ggplot(all_scores, aes(x = Sentiment, y = average_score, fill = Sentiment)) +
  geom_col() +
  labs(title = "Average Sentiment Scores by Category",
       x = "Sentiment Category",
       y = "Average Score") +
  theme_minimal()
library(ggplot2)

# Assuming 'data_scored' has been created, if not, follow similar steps as before
data_scored <- data %>%
  unnest_tokens(word, Translated_Review) %>%
  mutate(word = tolower(word)) %>%
  inner_join(afinn, by = "word") %>%
  group_by(App, Sentiment, Sentiment_Polarity, Sentiment_Subjectivity) %>%
  summarize(sentiment_score = mean(value, na.rm = TRUE), .groups = 'drop')

# Scatter plot for Sentiment Score vs. Polarity
a1 <- ggplot(data_scored, aes(x = Sentiment_Polarity,
                              y = sentiment_score, color = Sentiment)) +
  geom_point(alpha = 0.6) +
  labs(title = "Sentiment Score vs. Polarity",
       x = "Sentiment Polarity", y = "Sentiment Score")

# Scatter plot for Sentiment Score vs. Subjectivity
a2 <- ggplot(data_scored, aes(x = Sentiment_Subjectivity,
                              y = sentiment_score, color = Sentiment)) +
  geom_point(alpha = 0.6) +
  labs(title = "Sentiment Score vs. Subjectivity", x = "Sentiment Subjectivity", y = "Sentiment Score")

grid.arrange(a1, a2, ncol = 2)
```

```r
# Calculate correlation matrix
cor_matrix <- cor(data_scored[, c("sentiment_score", "Sentiment_Polarity",
                                  "Sentiment_Subjectivity")], use = "complete.obs")
colnames(cor_matrix) <- c("Score", "Polarity", "Subjectivity")
rownames(cor_matrix) <- c("Score", "Polarity", "Subjectivity")


# Plot the correlation matrix
corrplot(cor_matrix, method = "color",
         col = colorRampPalette(c("white", "darkred"))(200),
         tl.col = "black",
         tl.srt = 45,
         tl.cex = 0.8,
         number.cex = 0.7)
# Load AFINN sentiment lexicon
afinn <- get_sentiments("afinn")

# Modified function to calculate sentiment scores and return detailed dataframe
calculate_scores <- function(df) {
  df %>%
    unnest_tokens(word, Translated_Review) %>%  # Tokenizing the text
    mutate(word = tolower(word)) %>%           # Convert words to lower case
    inner_join(afinn, by = "word") %>%
    group_by(Sentiment, App, Sentiment_Polarity, Sentiment_Subjectivity) %>%
    summarize(sentiment_score = mean(value, na.rm = TRUE), .groups = 'drop')
}

# Applying function to each sentiment dataset
data_positive_scored <- calculate_scores(filter(data, Sentiment == "Positive"))
data_negative_scored <- calculate_scores(filter(data, Sentiment == "Negative"))

# Ensure data is available for modeling
data_positive_scored <- data_positive_scored %>% ungroup() %>% distinct()
data_negative_scored <- data_negative_scored %>% ungroup() %>% distinct()


# Linear model for positive reviews
lm_positive <- lm(Sentiment_Polarity ~ Sentiment_Subjectivity + sentiment_score,
                  data = data_positive_scored)

# Linear model for negative reviews
lm_negative <- lm(Sentiment_Polarity ~ Sentiment_Subjectivity + sentiment_score,
                  data = data_negative_scored)
# Summarize the models
summary(lm_positive)  # Model for positive reviews
summary(lm_negative)  # Model for negative reviews
# Visualization for Positive Reviews
ggplot(data_positive_scored, aes(x = Sentiment_Subjectivity, y = Sentiment_Polarity,
                                 color = sentiment_score)) +
  geom_point(alpha = 0.6) +
  scale_color_gradient(low = "darkred", high = "red")+
  geom_smooth(method = "lm", se = FALSE, formula = y ~ x) +
  labs(title = "Polarity vs. Subjectivity in Positive Reviews",
```

```
       x = "Sentiment Subjectivity", y = "Sentiment Polarity")
# Visualization for Negative Reviews
ggplot(data_negative_scored, aes(x = Sentiment_Subjectivity, y = Sentiment_Polarity,
                                 color = sentiment_score)) +
  geom_point(alpha = 0.6) +
  scale_color_gradient(low = "darkred", high = "red")+
  geom_smooth(method = "lm", se = FALSE, formula = y ~ x) +
  labs(title = "Polarity vs. Subjectivity in Negative Reviews",
       x = "Sentiment Subjectivity", y = "Sentiment Polarity")
```