GOOGLE PLAY STORE

Machine Learning Project

Statistics 4241
Summer 2025

Yunjiu Li
yl5890@columbia.edu

Laurel Stewart
lks2143@columbia.edu

Table of Contents

Contributions:

Team 4 – Yunjiu Li and Laurel Stewart

Report – Laurel Stewart

Slides – Yunjiu Li

Data Acquisition and Modelling - both

Objective and Motivation

Businesses and developers have multiple paths to make money from their apps. Revenue comes from ads, in-app purchases and upcharges, charging a flat fee for the app itself, or using the app as an extension of a larger brand presence. Each of these choices requires strategic resource allocation in terms of development, marketing, pricing, or even what type of app you choose to make. Looking at the success rates of existing apps helps businesses to make informed decisions and forecasting as they create new apps for the market. Looking at Google Play Store data is important because while developing for that platform costs as much as for the Apple App Store, Android users are far less likely to spend money on apps. Moreover, because of the very many versions of hardware and Android OS in the wild, user support for Android apps is time-consuming and difficult. For these reasons, building the app that will bring in the most revenue is crucial.

Key Project Takeaways

- **Know your data and its limitations**: because our data was from scraping we did not have control over what came in, and some important information was simply not available through scraping.
- **Use the tools to sanity check**: Garbage in, garbage out. The feature importance tool used at the end of our modeling revealed a key error in our dataset.
- **Sometimes there is no answer**: Proving a negative is less satisfying than having the *Eureka!* moment of discovering the key to making lots of money in the Google Play Store.

File Locations

The project consists of data files and Python notebooks.

- Project folder - dropbox.com
- Initial data - dropbox.com. These .CSV files were combined to create a single list of URLS to scrape (url1.csv)
- Cleaned data and preprocessing - dropbox.com. The file **google_play_apps.csv** was cleaned and processed to produce **google_play_apps_encoded.csv**, the file that was used in the models.
- Code - dropbox.com. The code is broken into three notebooks:
    - data.ipynb – data scraping and initial cleaning
    - project.ipynb – EDA and feature engineering
    - model.ipynb – The models used in the project

Data Collection

Because there was not one single data set that contained a broad number of categories and nor does Google freely share this information, we got our data through multiple steps. The initial plan was to use a preexisting dataset from Kaggle.Unfortunately, we found that the sets available lacked several key characteristics of the apps that we were interested in, including the release date of the app and real installation numbers. We then realized that we would need to consolidate our data from multiple sources. First we chose 10 popular app categories and took a sample from each. Some categories and developers were excluded. For instance, to be successful in the News category, step one is “be the New York Times” or a similarly popular media company. Social networking was excluded for the same reason. Built-in and installable apps from Google itself

were also left out. App IDs were scraped by category using an online tool. Those were then consolidated into a CSV file for processing.

Tools:

Online tool for initial URLs: https://console.apify.com/

Scraper for retrieving Google Play Store information for each URL: https://pypi.org/project/google-play-scraper/

After we had our list of apps, we modified an existing Google App Store scraper to extract app-level details and build a dataframe. The dataframe had 316 rows of 27 columns.

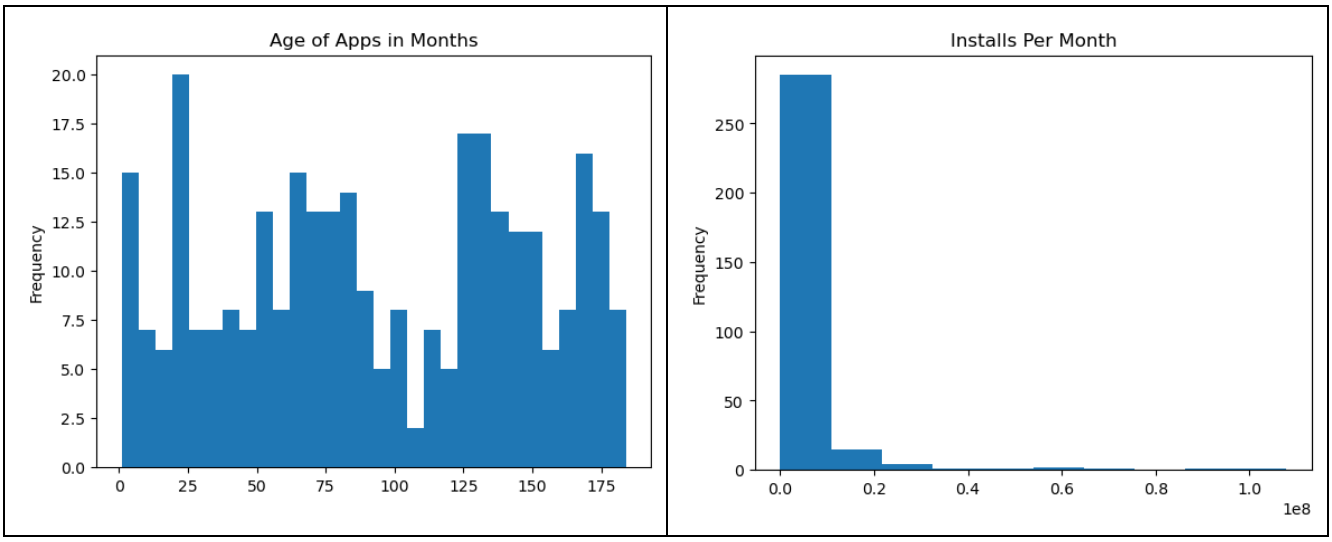
Removed in processing	Feature	Description
	title	App title/name
	description	Full app description
	descriptionHTML	HTML formatted description
	summary	Brief app summary
x	installs	Install count range
x	minInstalls	Minimum number of installs
	realInstalls	Actual install count
	score	App rating score
	ratings	Number of ratings
	reviews	Number of reviews
X	histogram	Rating distribution histogram
	price	App price
	free	Whether app is free
	currency	Price currency
	sale	Whether app is on sale
	saleTime	Sale duration/timing
	originalPrice	Original price before sale
	saleText	Sale description text
	offersIAP	Offers in-app purchases
	inAppProductPrice	In-app purchase pricing
x	developer	Developer name
x	developerId	Developer identifier
x	developerEmail	Developer contact email
x	developerWebsite	Developer website URL
x	developerAddress	Developer address
x	privacyPolicy	Privacy policy URL
	genre	App genre/category

	genreId	Genre identifier
	categories	App categories
x	icon	App icon image
x	headerImage	Header/banner image
x	screenshots	App screenshot images
x	video	Promotional video
x	videoImage	Video thumbnail image
	contentRating	Age/content rating
	contentRatingDescription	Content rating details
	adSupported	Whether app shows ads
	containsAds	Contains advertisements
	released	Release date
	lastUpdatedOn	Last update date
X	updated	Update timestamp
	version	App version number
X	comments	User comments/reviews
x	appId	Unique app identifier
x	url	Google Play Store URL

Exploratory Data Analysis and Data Pre-Processing

Before scraping, we checked the initial list of URLs for duplicates and removed one. After scraping and creating the dataframe, we checked for missing values. Three columns were missing for all rows - **saleTime, originalPrice,** and **saleText**. Those columns were removed. 267 of the rows had nulls in the **contentRatingDescription** column. Most of those were for the "Everyone" rating. For the missing values, we duplicated the **contentRatingColumn**. 23 rows were missing **score, ratings,** and **reviews.** Those were replaced with zeros. We checked the missing values in the **inAppProductPrice** column. This column was strings. We broke this into 2 numeric columns – **minInAppProductPrice** and **maxInAppProductPrice**.

Four rows were missing a released value. Those were dropped because of the high variability of that value and because of the relatively low number of missing values. Because apps were released at different dates, we calculated the app's age in months and then installs per month to avoid penalizing very new ones against apps that were years old. Ages that were rounded down to 0 were made to be 1. We were then able to look at our target - installsPerMonth



After doing a logic check on our data, we exported it back to the .CSV "google_play_apps.csv."

There were still several text-based columns in the dataset. We dropped **descriptionHTML** because it was redundant to **description**. We did the same with **contentRatingDescription, Genre,** and **containsAds.**

All currencies were in USD, so we dropped that column. There were only 2 rows which were paid apps, which were dropped. For that reason, the **price** and **free** columns were also dropped. **Ratings** and **Reviews** were converted to per-month values. None of the apps were on sale so the **sale** column was dropped.

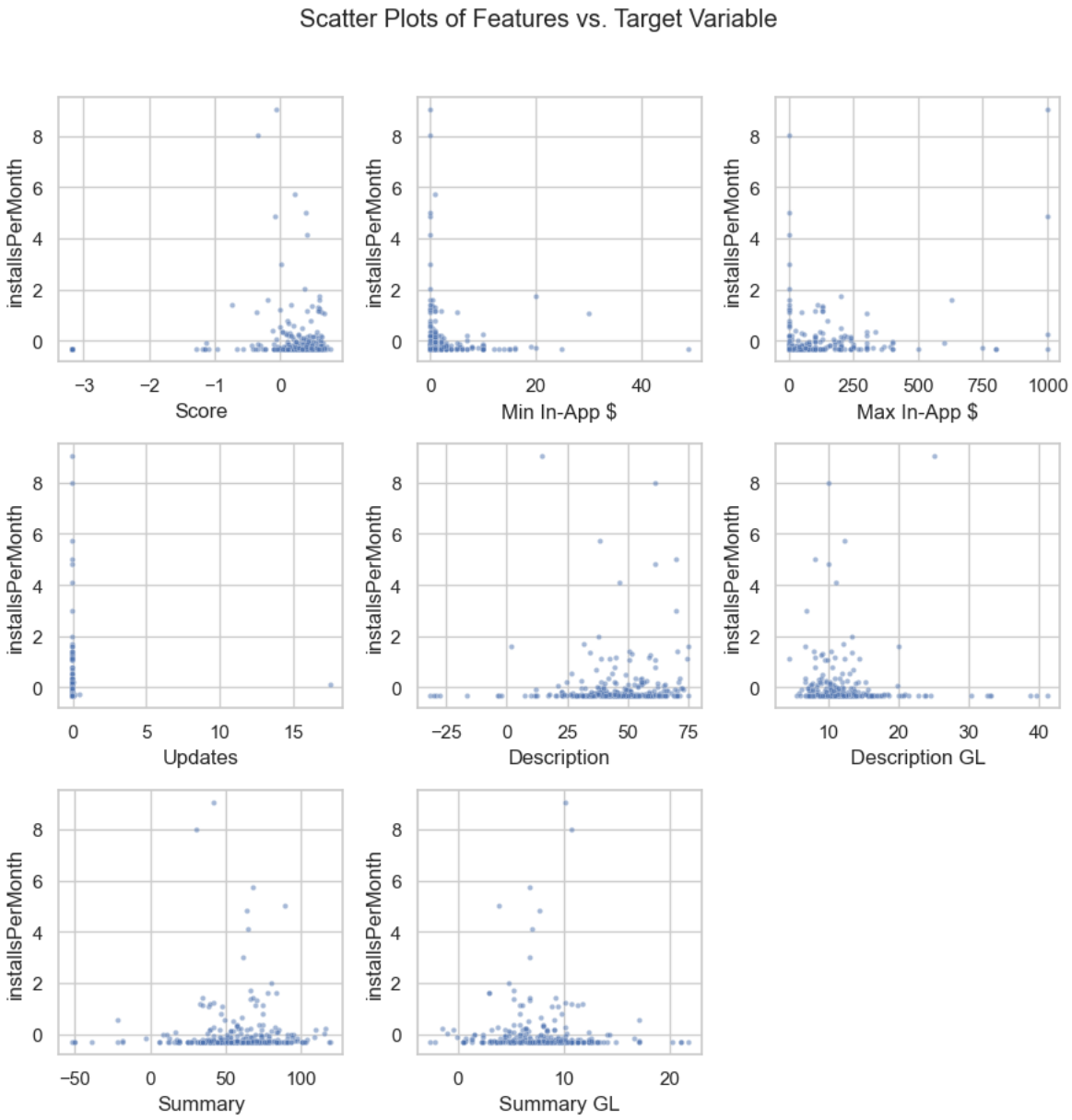
LastUpdatedOn gives an idea of how often the apps are updated regardless of age. Number of updates can be extrapolated from **version**. Using a scheme of xx.nn… Where xx is a major update and nn is a minor update, we calculated the number of updates per month, disregarding any patches, by adding the major and minor updates and dividing by the age. Having now calculated age and number of updates, we can now drop the **released, last updated on, version, version_number, version_major,** and **version_minor columns.**

Because we had text-based descriptions, we used the **textstat** package to analyze their readability. After scoring the **descriptions** and **summaries** columns for readability and reading time, we were able to drop those columns.

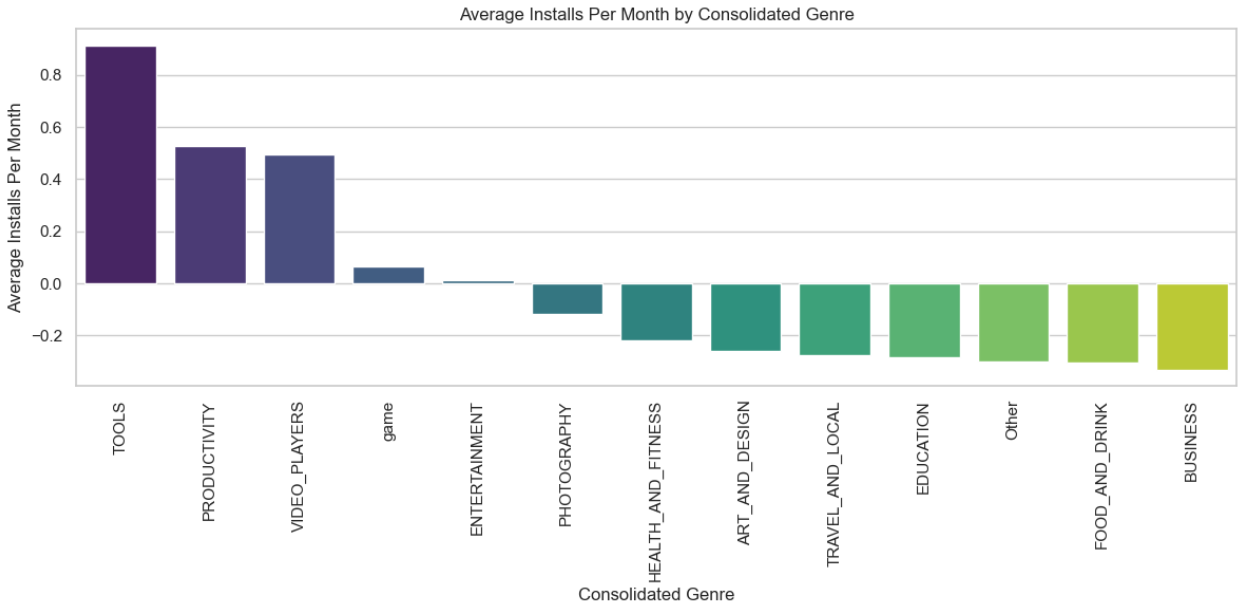
Changing Booleans to 0s and 1s. The columns for **offersIAP** (in app purchases) and **containsAds** are Boolean true/false. We changed them to 0 for false and 1 for true.

Standardizing **ratings**. Because app ratings tend to cluster between 4 and 5, it is necessary to standardize them to see any meaningful differences.

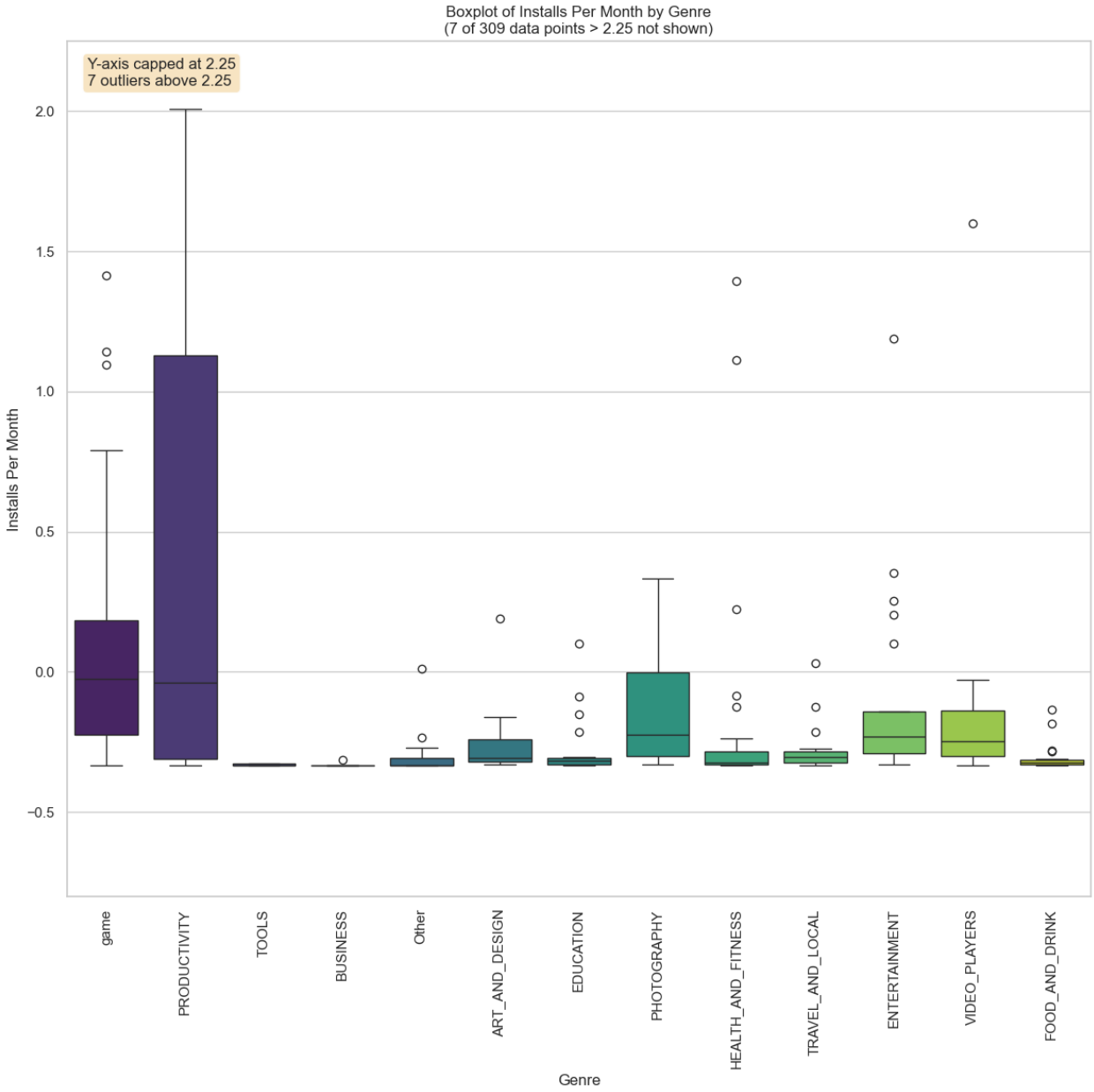
After standardizing, we can look at the relationships between the target variable and the numeric columns:



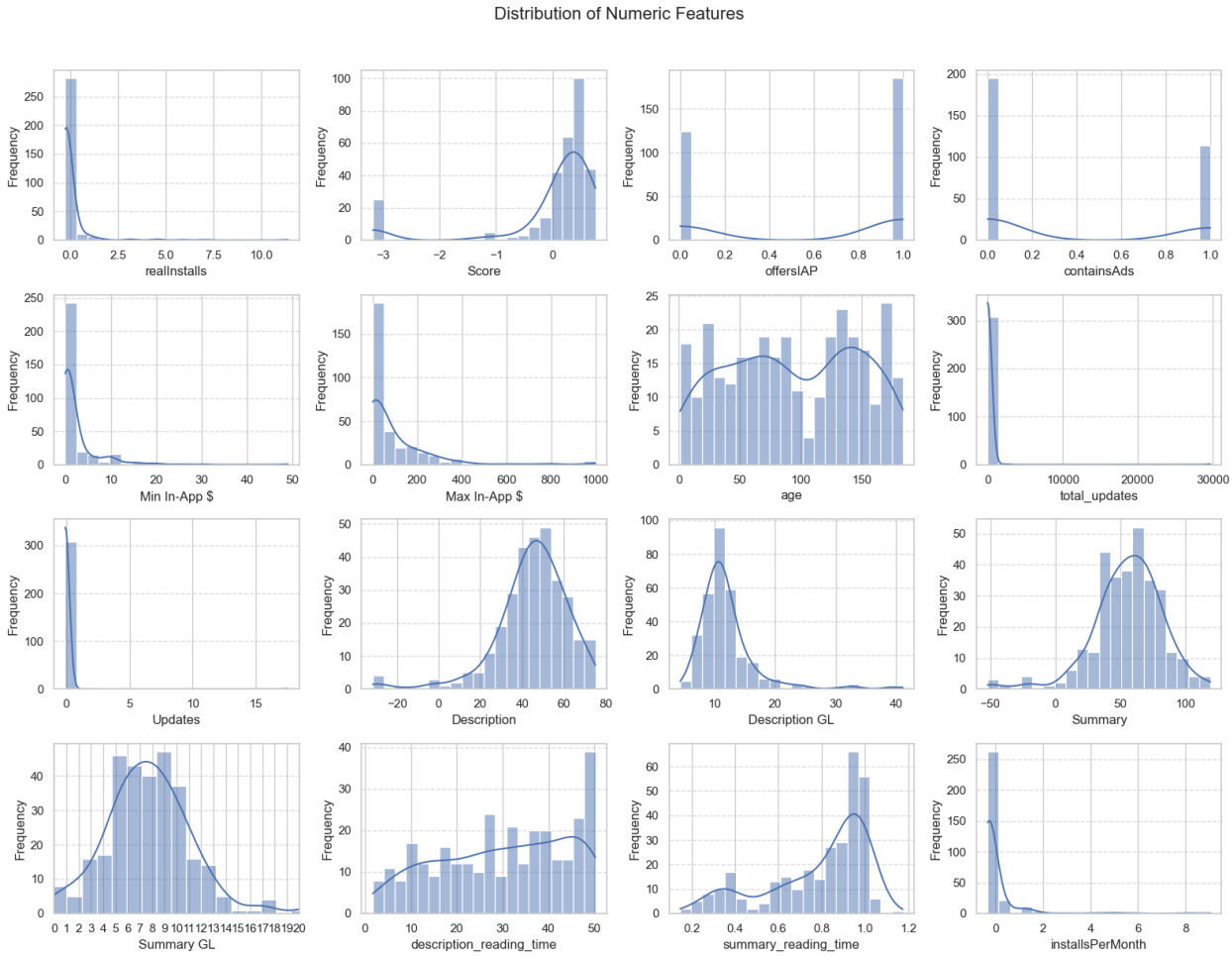
After looking at average installsPerMonth by genre (app category), we saw that the games category was broken down into more granular genre than the others. We consolidated all of those into one genre – game – and consolidated app genre with 10 or fewer into 'other.'

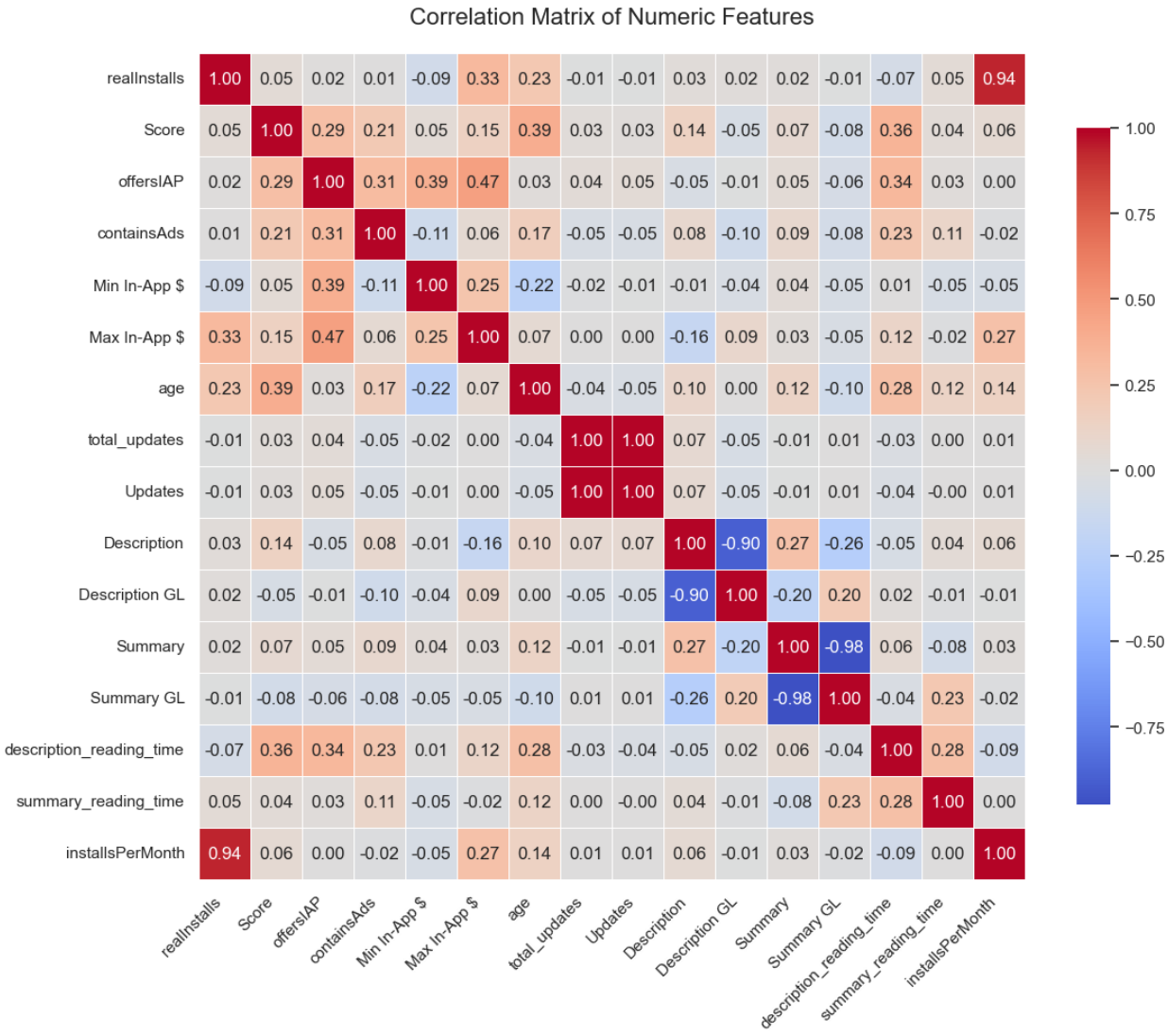


We saw a wide range of variability in our target feature by genre:

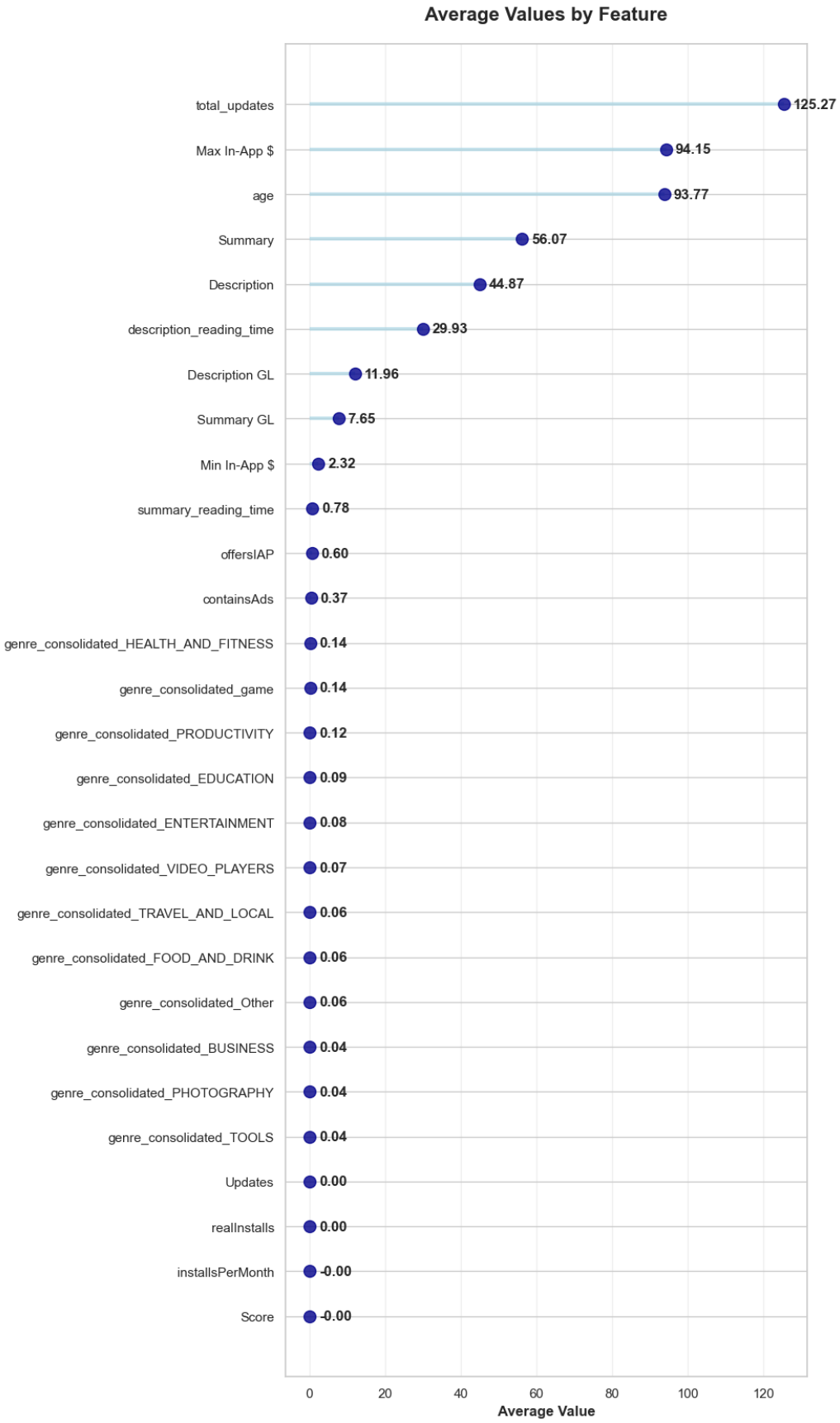


Having changed our features to numeric values, we could then look at the distribution of each as well as check for correlated features. A closer look at this time would have revealed a mistake in the columns we retained, however catching that was still to come!


Distribution of Numeric Features
Frequency
realInstalls
Score
offersIAP
containsAds
Min In-App $
Max In-App $
age
total_updates
Updates
Description
Description GL
Summary
Summary GL
description_reading_time
summary_reading_time
installsPerMonth

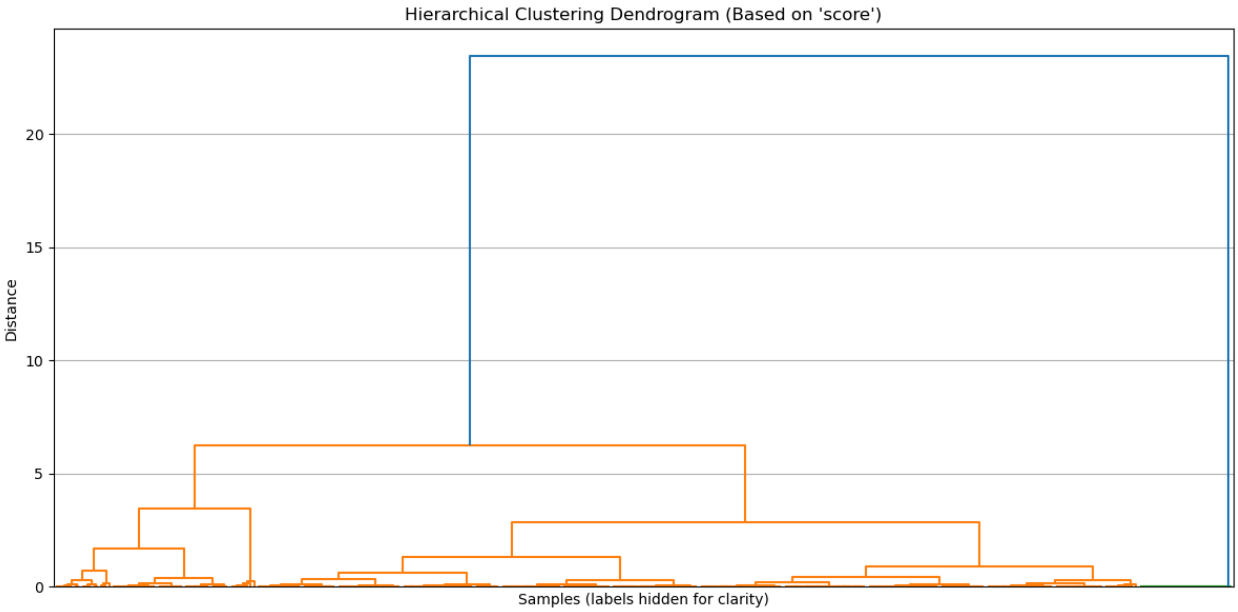


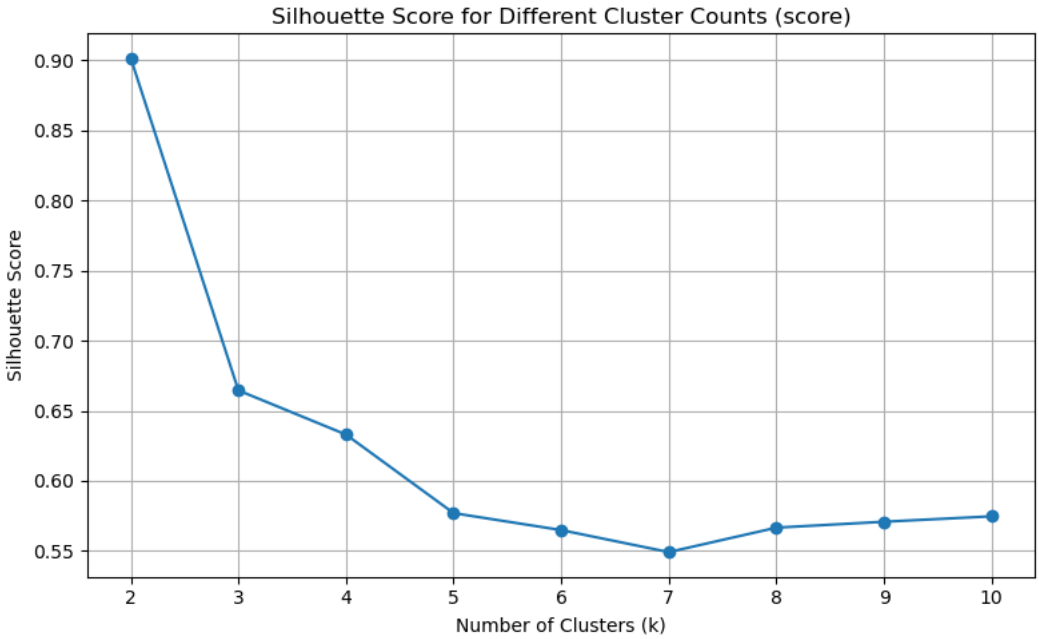
There was still one categorical feature left – genre. We used full-rank dummy encoding for those and removed our unused columns.


Average Values by Feature
total_updates
Max In-App $
age
Summary
Description
description_reading_time
Description GL
Summary GL
Min In-App $
summary_reading_time
offersIAP
containsAds
genre_consolidated_HEALTH_AND_FITNESS
genre_consolidated_game
genre_consolidated_PRODUCTIVITY
genre_consolidated_EDUCATION
genre_consolidated_ENTERTAINMENT
genre_consolidated_VIDEO_PLAYERS
genre_consolidated_TRAVEL_AND_LOCAL
genre_consolidated_FOOD_AND_DRINK
genre_consolidated_Other
genre_consolidated_BUSINESS
genre_consolidated_PHOTOGRAPHY
genre_consolidated_TOOLS
Updates
realInstalls
installsPerMonth
Score
125.27
94.15
93.77
56.07
44.87
29.93
11.96
7.65
2.32
0.78
0.60
0.37
0.14
0.14
0.12
0.09
0.08
0.07
0.06
0.06
0.06
0.04
0.04
0.04
0.00
0.00
-0.00
-0.00
0
20
40
60
80
100
120
Average Value

Feature Engineering

We tried multiple methods to see if clustering was necessary, however the results were weak:





With initial poor results in our models, we tried additional methods of feature engineering, which actually worsened the performance of the models:

```
# Feature Engineering
df['log_installsPerMonth'] = np.log1p(df['installsPerMonth'])  # Log-transformed target
df['iap_price_range'] = df['maxInAppProductPrice'] - df['minInAppProductPrice']  # Price range
df['score_to_age_ratio'] = df['score'] / (df['age'] + 1)  # Score normalized by age
df['readability_blend'] = (df['description_readability_score'] + df['summary_readability_score']) /
2  # Combined readability
df['update_density'] = df['total_updates'] / (df['age'] + 1)  # Updates per unit age
df['score_reading_interact'] = df['score'] * df['description_reading_time']  # Interaction term
```

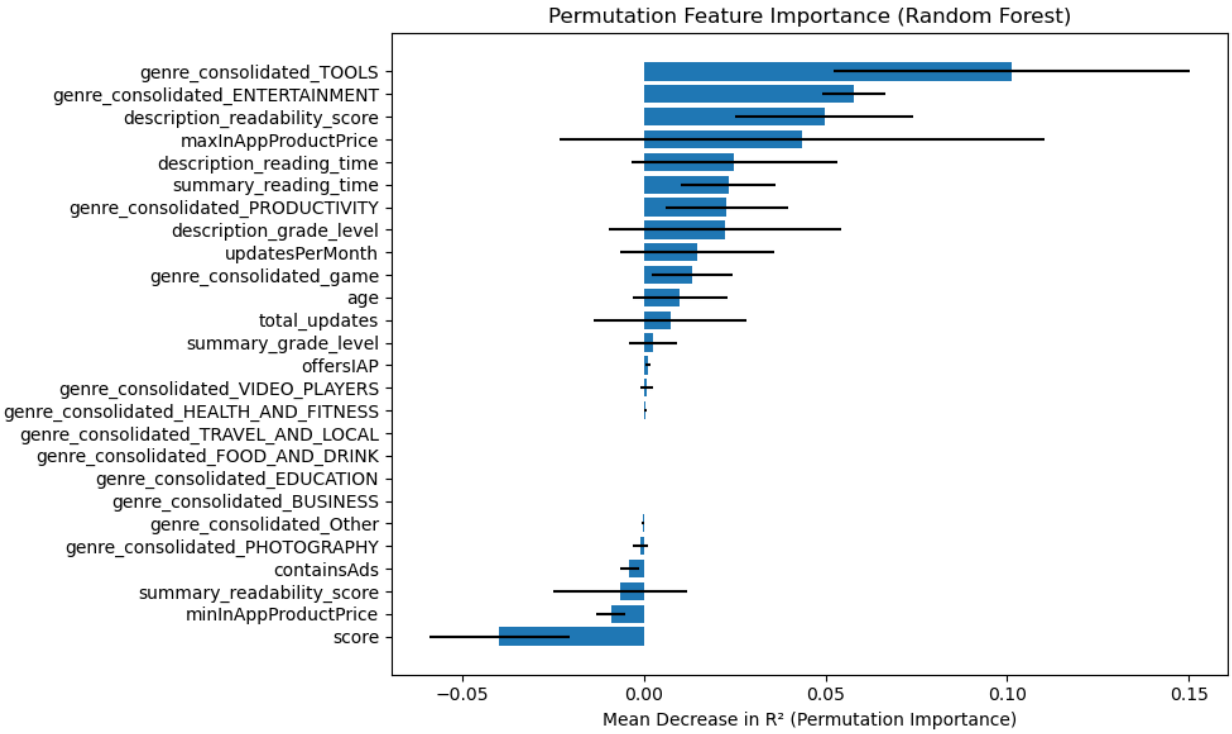
Modeling

We chose three models for the project: **simple linear regression** model, a **random forest** model, and **gradient boosting regressor**.

The initial results were really bad. Only Gradient Boosting could beat simply predicting the mean and even that was barely:

Model	R^2 Score	RMSE
Linear Regression	-0.0812	1.1733
Random Forest Regressor	-0.1983	1.2352
Gradient Boosting Regressor	**0.1112**	**1.0638**

Actually, that is not completely true. The first round of predictions was terrific. That is until we produced a chart of permutation feature importance. And in that we saw that the most important feature to the predicted value of installsPerMonth was … realInstalls. Yes, this is the column that we had not removed from our dataset. After correcting that error and remodeling we found:



Conclusions

The data of the apps we looked at on the Google Play Store was highly skewed with many outliers. The truth is most apps do not reach blockbuster or even mediocre status, regardless of quality. Some app genres are more popular than others in terms of installations, however not every app in those genres will perform well. To truly answer the question “what makes a hit app” we need more information, particularly around marketing budgets and the success of individual development teams.