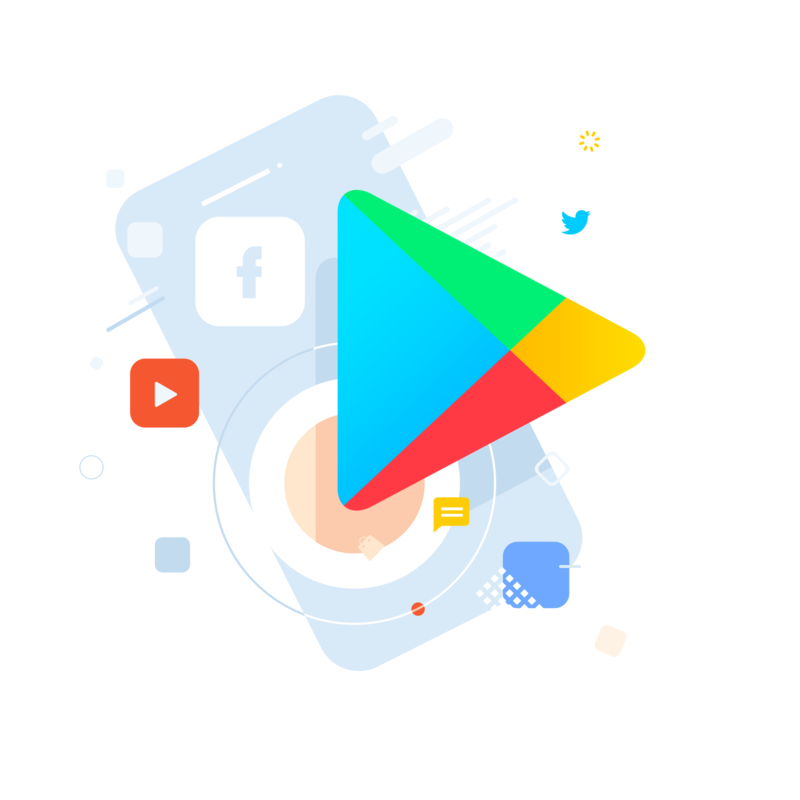
# Google playstore

# prediction



Name:S.MADHANKUMAR

[mail:madhankumar67575@gmail.com](mailto:abishekbhat31@gmail.com)

Linkedin: <https://www.linkedin.com/in/-madhankumar-s>

Github: https://github.com/madhanop

ABSTRACT

The ability to use services and products on the go has been a major leap in this century.Applications on the Google play store aim to do exactly that. Owing to worldwide accessibility and the ease of use, it has not only become the most popular application download destination but also a hotbed for competing services to attract and gain customers. This project aims to employ machine learning & visual analytics concepts to gain insights into how applications become successful and achieve high user ratings. We started the analysis by creating a data dictionary to understand the structure of the dataset and what each feature represents. Post that, we handled the missing values in some of the columns by either dropping the rows or imputing them with the mode values, depending on the percentage of nulls in each feature. We transformed and some of the columns like Installs, Size and Price to numeric type for ease of analysis.

## 

## Data set:

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

Source Code:

[**https://github.com/madhanop/capstone-project-GooglePlaystore**](https://github.com/madhanop/capstone-project-GooglePlaystore)

# ACKNOWLEDGE

I am using this opportunity to express my gratitude to everyone who supported me throughout the course of my capstone project. I am thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

Further, I have fortunate to have Mr. ANBU JOEL as my mentor. He has readily shared his immense knowledge in data analytics and guide me in a manner that the outcome resulted in enhancing my data skills.

I certify that the work done by me for conceptualizing and completing this project is original and authentic.

Date: 1/7/2022 Name: S.MADHAN KUMAR

# CERTIFICATION OF COMPLETION

I certify that the project titled " Google PlayStore" was undertaken and completed.

(1st August 2022).

Mentor: Mr. ANBU JOEL

Date: 1st August, 2022

Place:Trichy.

CONTENT

1. **INTRODUCTION……………..6**
2. **DATASET…………………..6**
3. **EDA……………….7**
4. **METHODS…………….8**
5. **RANDOM FOREST …………….9**
6. **K-NEAREST NEIGHBORS ……………….12**
7. **RESULTS ……………….13**
8. **CODE………………15**

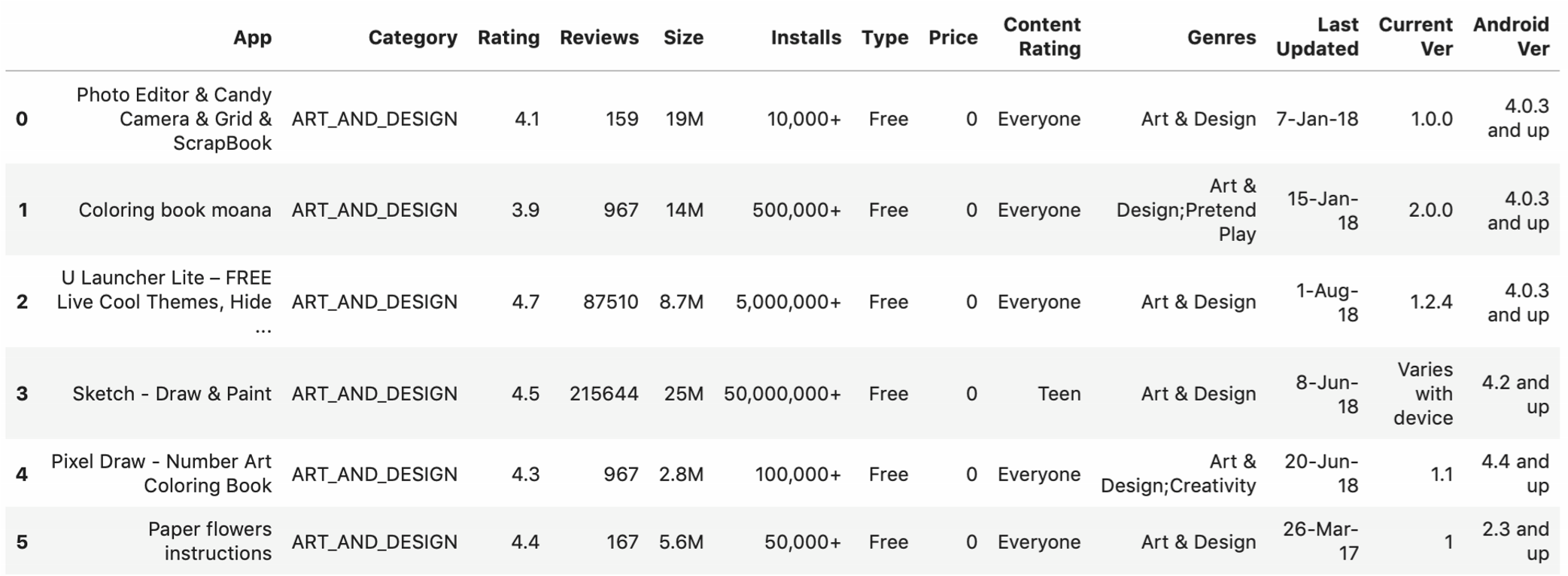
**INTRODUCTION**

While many public data sets provided Apple store data, there is not enough analysis into apps in Google Play store, partly because it’s more difficult to scrape data from Google Play store.

Nowadays, as mobile phones become more popular, people tend to spend more time on their phones and app usage increases a lot. While this is a great opportunity for many app developers, it also becomes a challenge for many developers/businesses to develop a popular app.

In this project, I will try to predict the number of installs (target variable) from some features of the app itself, displayed in google play store app page. I am trying to find out what kind of apps are more popular and tend to stay longer in people’s phones. Since the exact number of installs was not available, but an estimate was given, it is treated as a categorical variable, so the problem becomes a classification problem.

DATASET DESCRIPTION



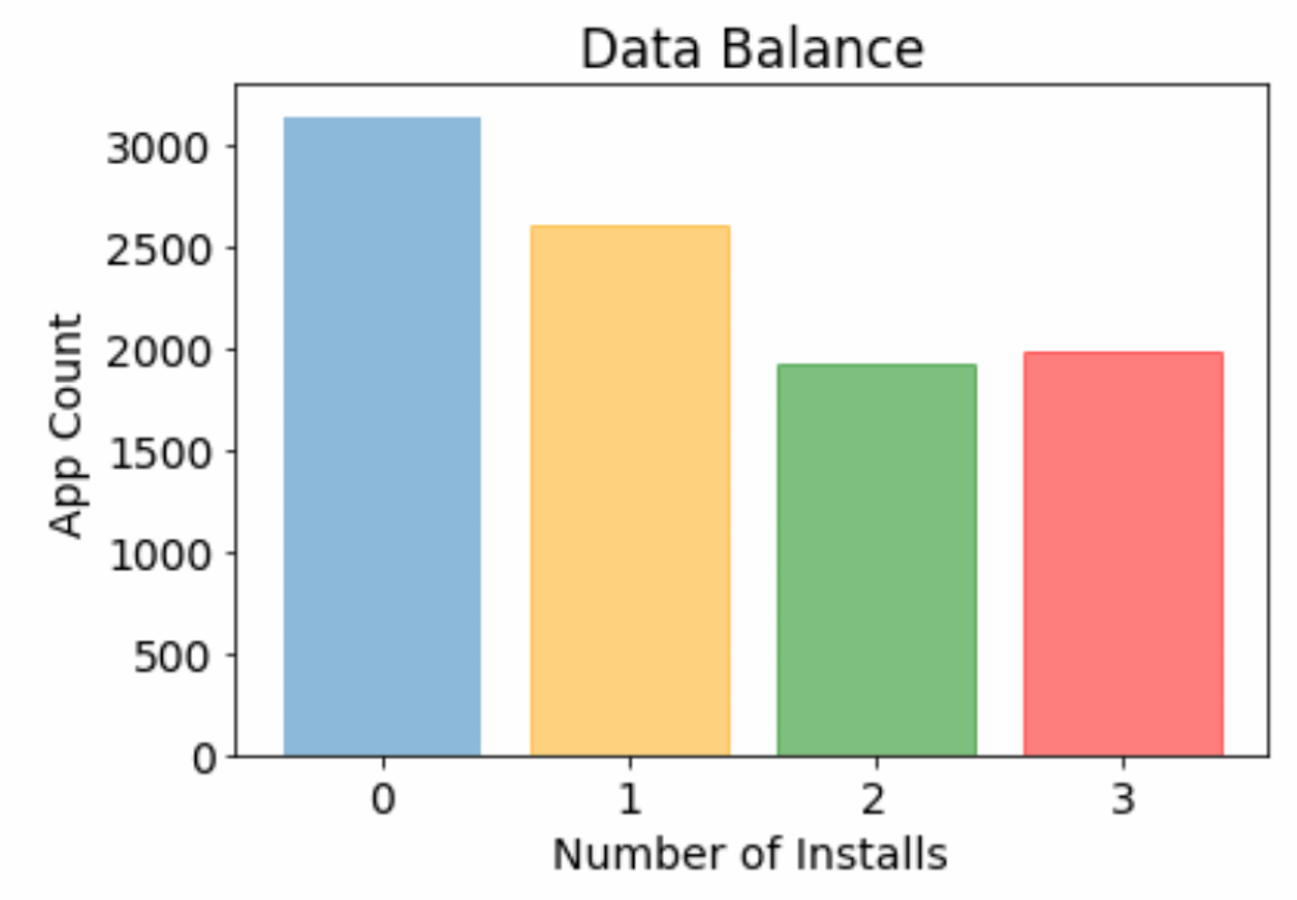
This dataset was downloaded from Kaggle and scraped from Google play store. The dataset was updated eight months ago.

1. googleaplystore.csv. This file has 10,841 entries, each representing an app listed in Google Store, and it has 13 columns. These columns are:
   * App: name of the App.
   * Category: categorical.
   * Rating: 1 to 5 (5 being the highest), continuous.
   * Number of reviews: continuous.
   * Size: file size, continuous. After preprocessing, all measured in megabyte, but some say "vary with device".
   * Number of installs: categorical target variable. After preprocessing, values include: <10k, <500k, <5 million, >5 million.
   * Type: categorial, Paid or Free.
   * Price: continuous, measured in US dollar.
   * Content Rating: categorical variable. Everyone, Everyone 10+, Teen, Mature 17+, Adult only 18+, Unrated. Unrated apps are treated like highmaturity apps until they get a rating.
   * Genres: categorical variable, similar to Category, but provides more detailed types.
   * Last Updated: last date this app was updated, at the time when this dataset was made.
   * Current Ver: current version of the app, treat as categorical. Some say "Varies with device".
   * Android Ver: Android operating system requirement, treat as categorical. Some say "Varies with device".

1. googleplaystore\_user\_reviews.csv
   * 64,296 entries, each representing one review. Columns: App, Translated\_Review, Sentiment, Sentiment\_Polarity, Sentiment\_Subjectivity.

**EDA**

Below shows the balance of the dataset. Target variable *Number of Installs* has been transformed to class 0, class 1, class 2, class 3.

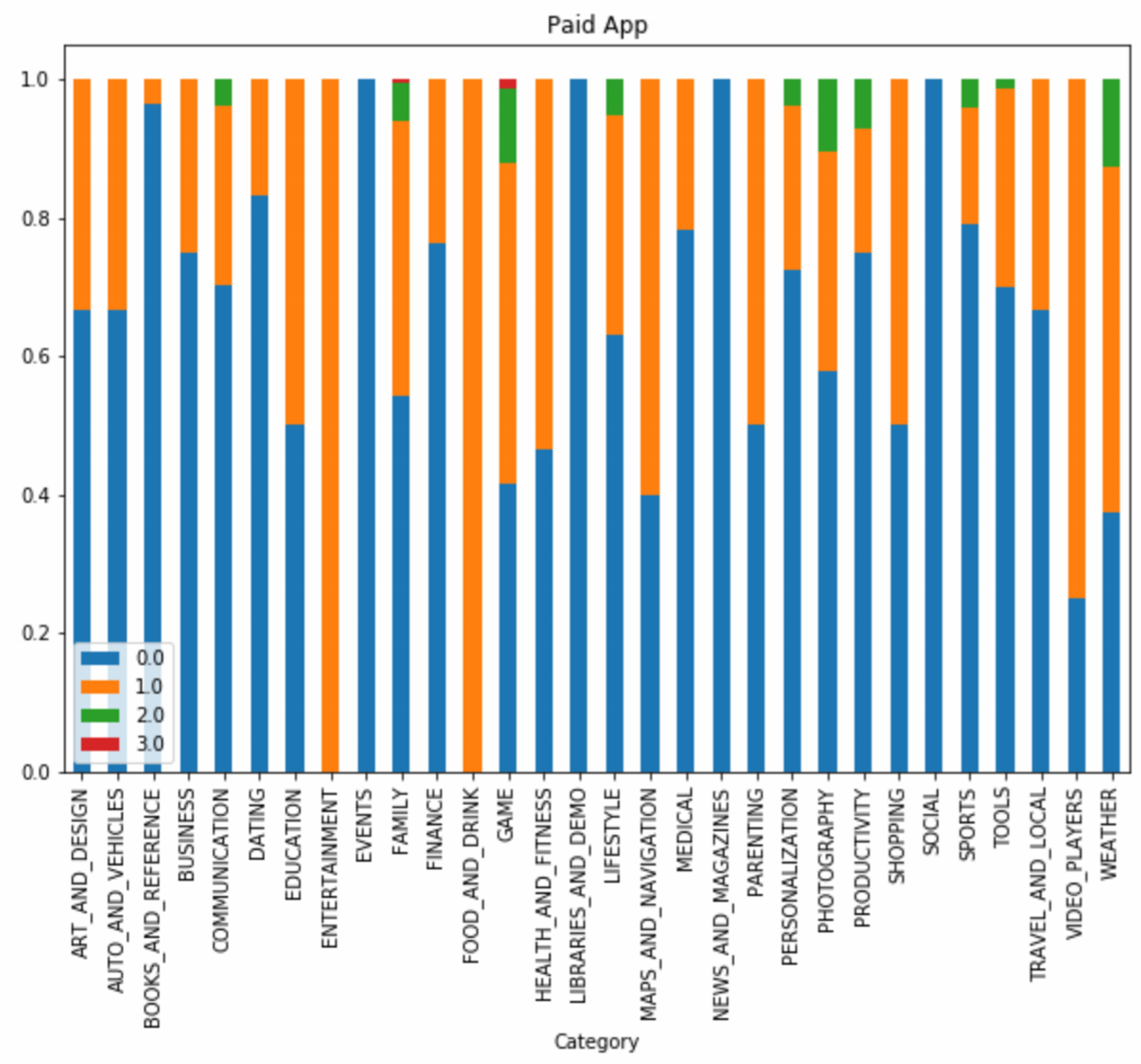


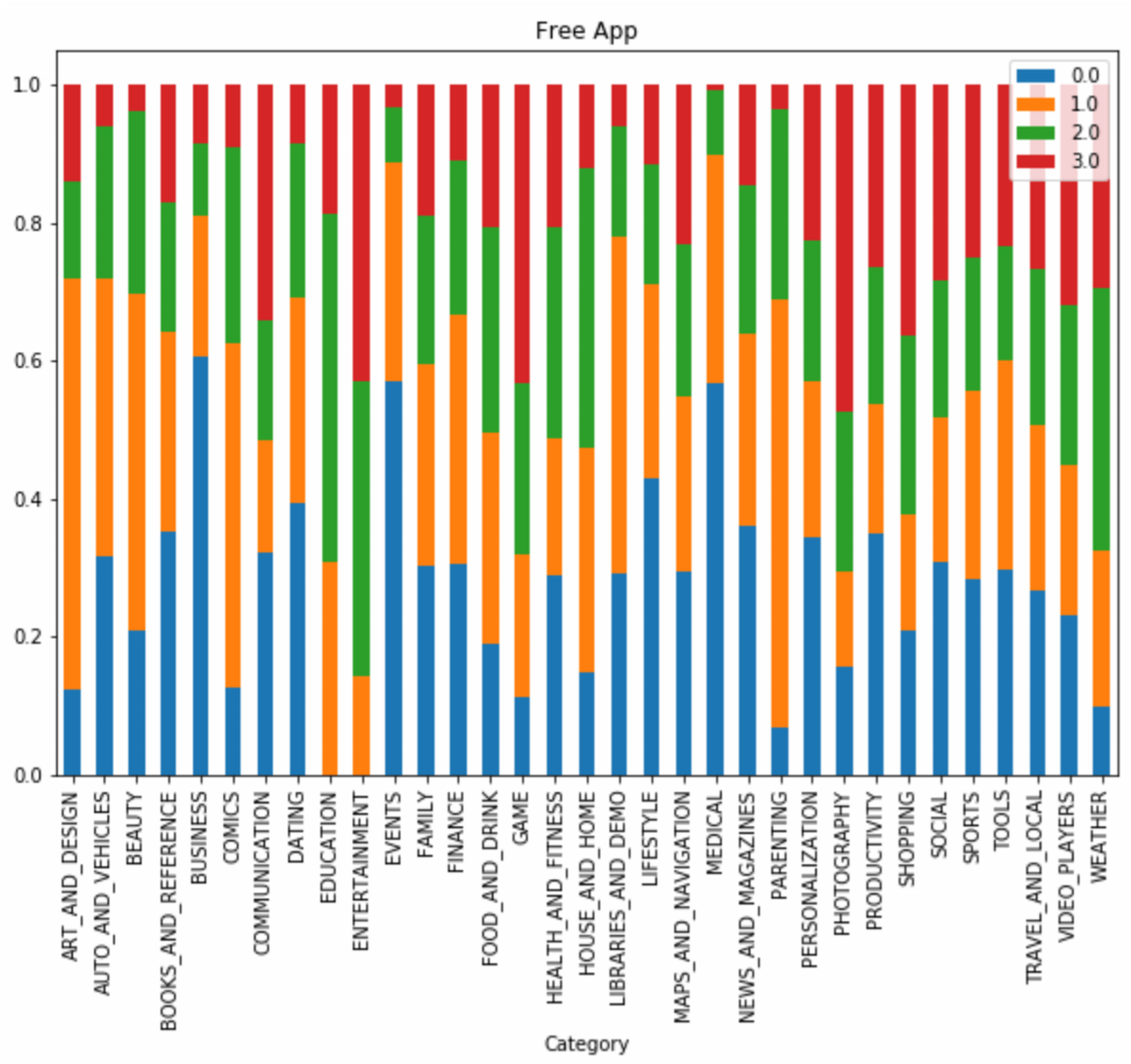
Based on my guess, free apps should be downloaded more often. From the stacked bar plots below, I do find that on average, green bars and red bars show up more often when the apps are free of charge.

Also, whether the app developer launch localized versions of their products with different file size considering the users’ Android device does make a difference. More popular apps tend to have localized device-specific versions.

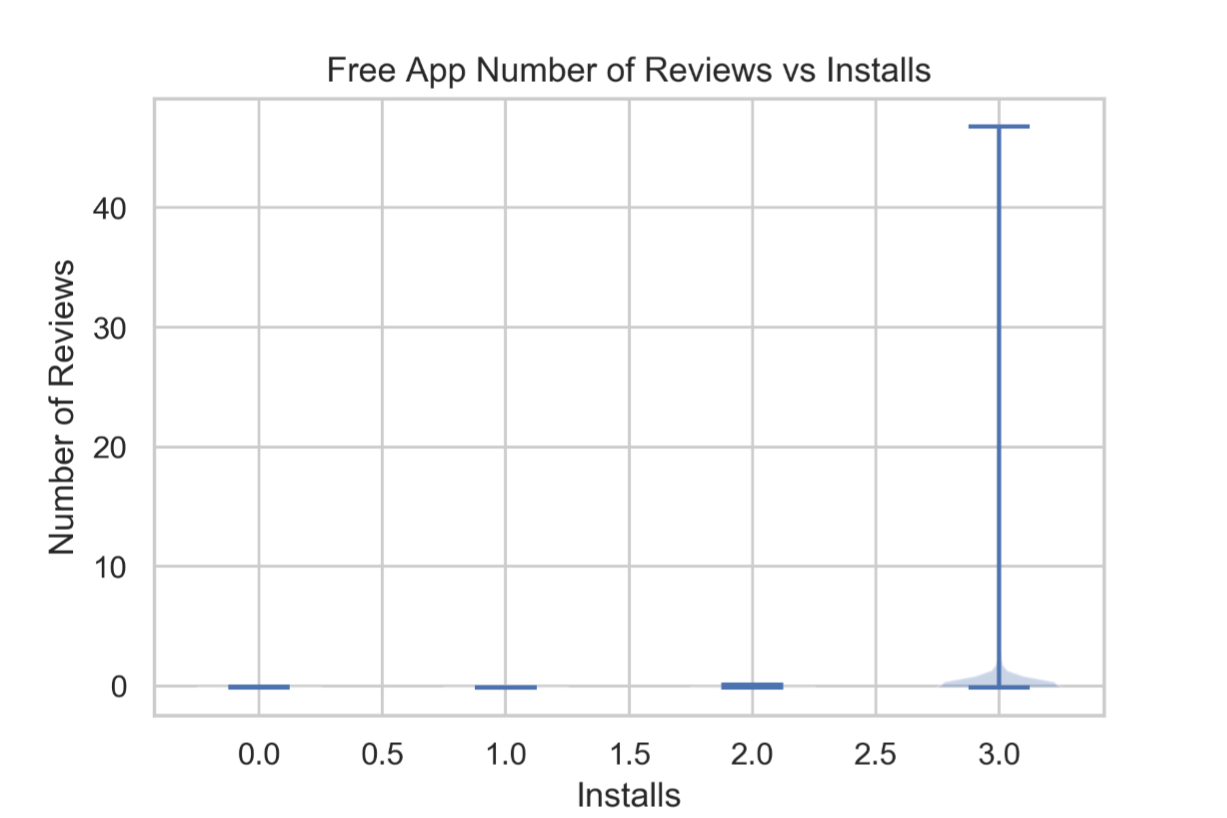
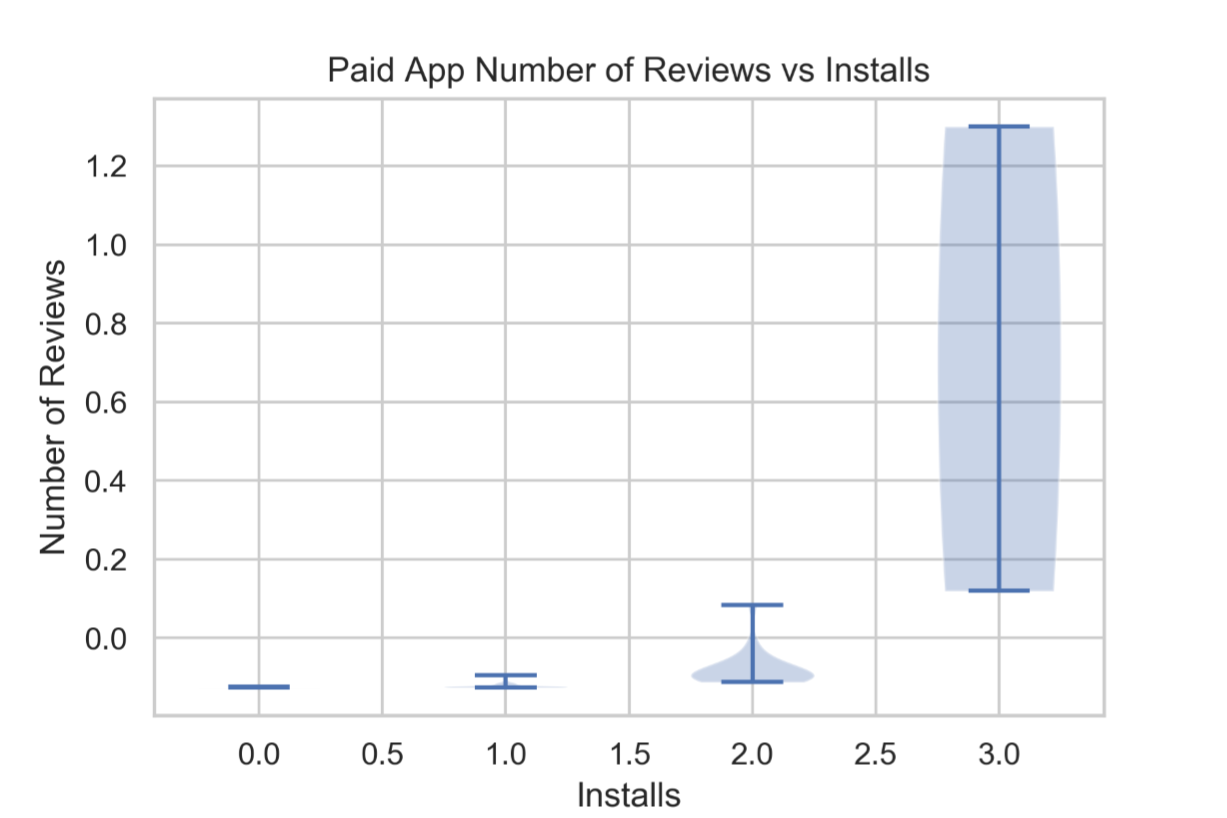
Furthermore, entertainment, food\_and\_drinks, video\_players categories seem to be more popular (larger number of installs), compared to lifestyle and medical.





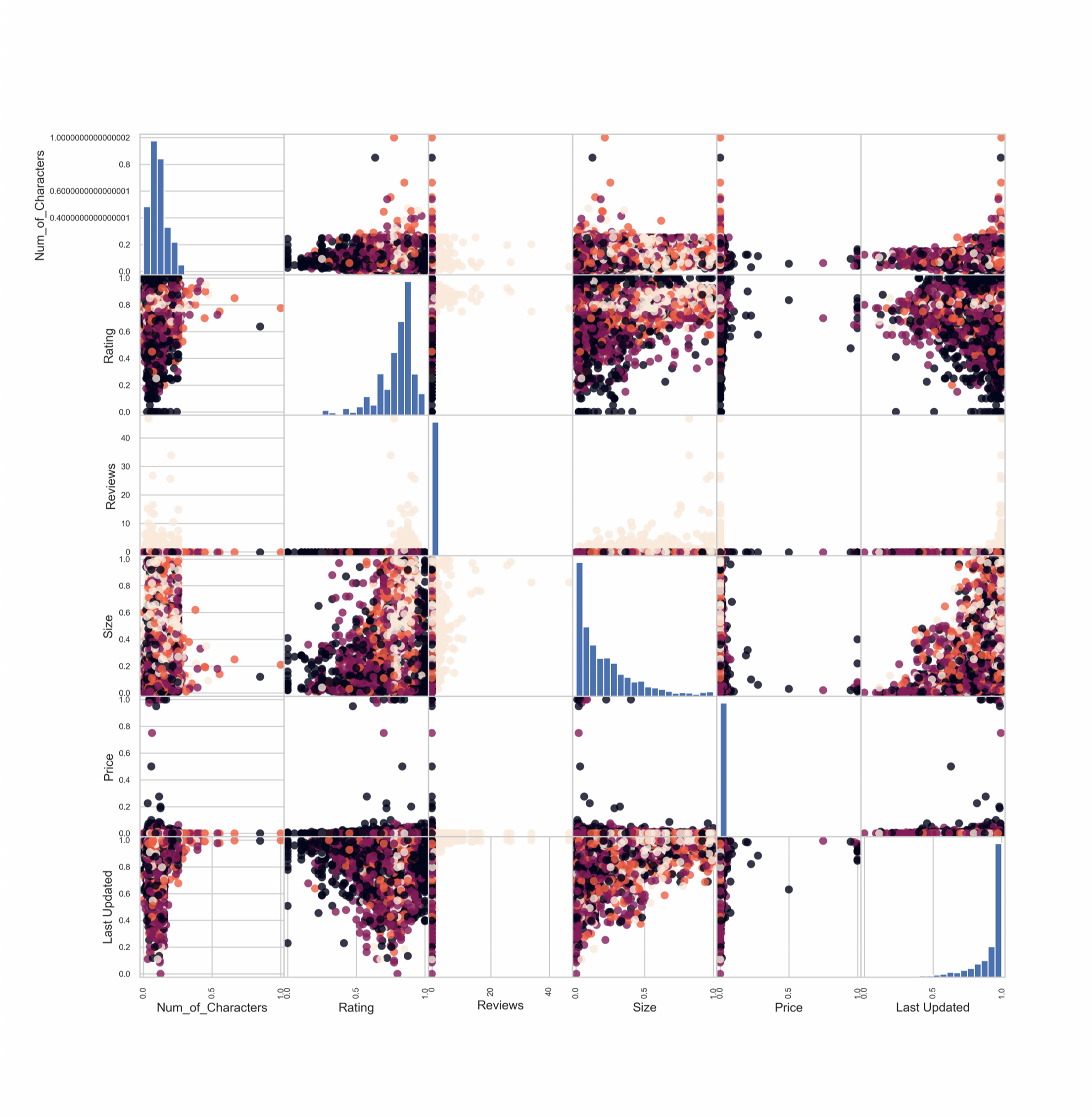


Also, it seems like the more popular apps have more reviews, and this is much more obvious with paid apps. My understanding is that when people spend money on an app, they are more likely to care more about it and leave reviews about it.



Now let’s look at the scatter matrix.

Looking at the distributions of all the continuous features, the most prominent founding is that they are all screwed. The number of characters in the app’s name is usually small, rating is usually high (if rated), most apps have few reviews, more apps have small file sizes, most apps are free or very cheap, most apps update frequently.

From the scatter plot, number of reviews is a strong indicator of whether the app has more downloads or not. Also, it seems like the higher the rating, the more popular the app is. 

**METHODS**

DATA PREPROCESSING

1. Data Cleaning
   * drop duplicates
   * use RegEx to extract data and change data types to what they should be
   * group the values in *Number of installs* to <10k, <500k, <5 million, >5 million
   * join googleplaystore\_user\_reviews and googleplaystore on app name
2. Feature engineering
   * Make a column for number of characters in app name
3. Missing value
   * Drop a few rows where ***Android Ver***, ***Current Ver***, or ***Type*** is missing  Drop columns ***Translated\_Review****,* ***Sentiment****,* ***Sentiment\_Polarity****,* ***Sentiment\_Subjectivity*** because these columns have over 91% missing
   * Run MCAR test on ***Rating*** and Size (p = 0), need to impute them
   * For ***Size***(continuous), when the value is “Varies with device”, replace it with np.nan and treat it as missing, and create a new column called *Size\_varies* which gives value 1 or 0, use iterative imputer to impute *Size* when needed
4. One-hot encoding: *Category, Genres, Current\_Ver\_truncated, Android\_Ver\_truncated*.
5. Ordinal encoding: *Number of installs, Type, Content Rating.* Type has values Free and Paid, so actually it does not matter. Content Rating is from general public to high restrictive.
6. Standard scaler: *Reviews, Size, Price, Last Updated, Num\_of\_Characters,* These values do not have an upper bound.
7. Minmax scaler: *Rating*, since it has value from 1 to 5.
8. After preprocessing, 9671 rows and 228 columns.

RANDOM FOREST

1. Parameter
   * max\_depth: 30, 35, 40, 45, 50, 55, 60
   * min\_samples\_split: 2, 3, 4, 5
   * n\_estimators: 100
   * random\_state: given by function call
2. Metric
   * Baseline accuracy.
   * Accuracy score on test set. Since this is a classification problem and the data is relatively balanced, accuracy score can be used.
   * Standard deviation of accuracy score, since it measures splitting and nondeterministic uncertainties. The lower the standard deviation, the better (stable) the model.
   * Number of standard deviations above baseline.
   * Result: 0.844 +/- 0.005, 103.8 std above baseline

1. Pipeline
   * Split data in a stratified manner into other and test because the four classes take different percentages.
   * Split other into 5 stratified folds.
   * Impute the continuous features with missing values using iterative imputer. From MCAR test I know that they are not missing completely at random.
   * Process appropriate continuous features using standard scaler or minmax scaler (see above for reasoning), fit and transform only train set, transform CV and test sets.
   * Process appropriate categorical features (see above for reasoning) using one-hot encoder or ordinal encoder.
   * Use parameter grid to choose the best parameters and model.
   * Run it 8 times to compute standard deviation of accuracy score.

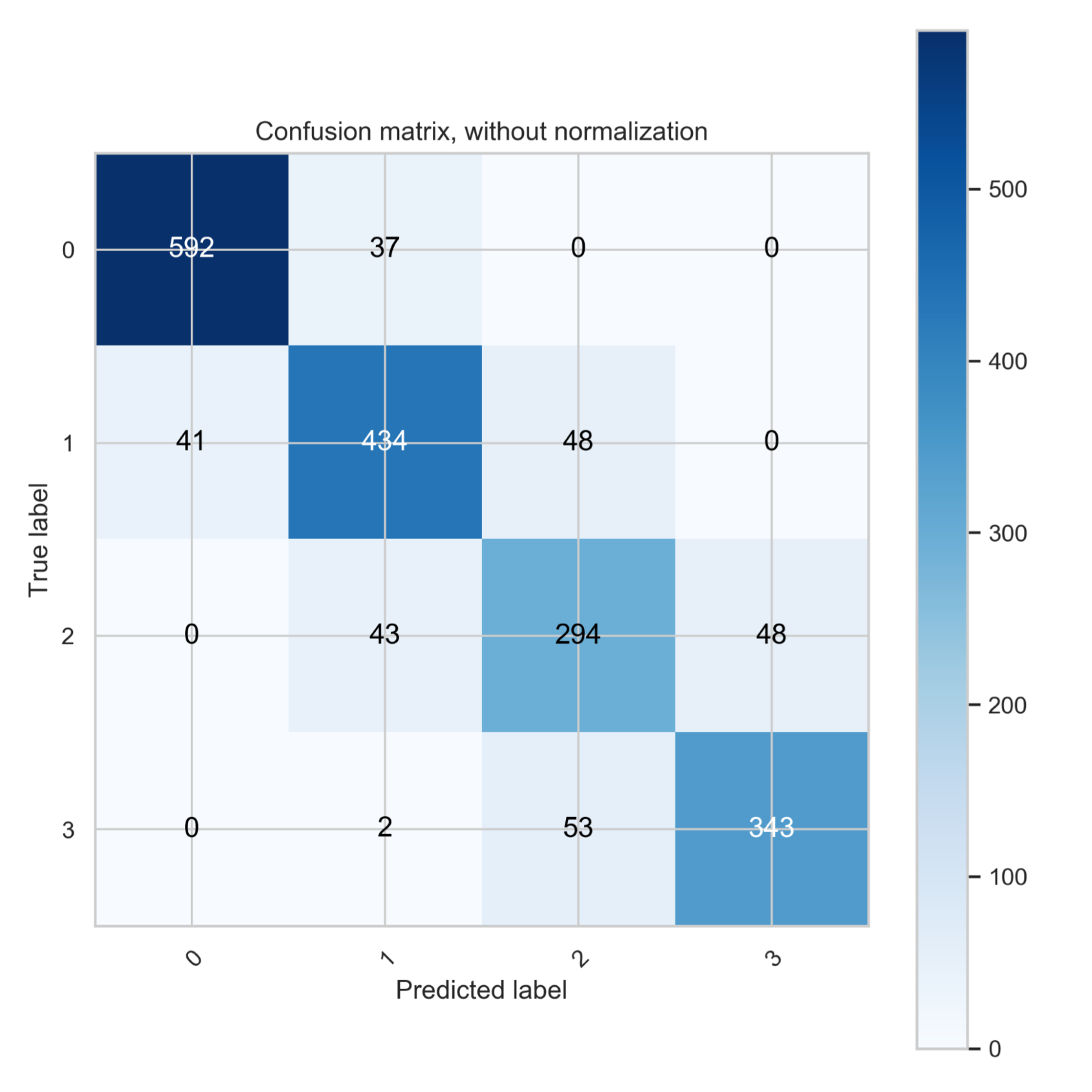
K-NEAREST NEIGHBORS

1. Parameter
   * n\_neighbors: 20, 25, 30, 35, 40, 45
2. Metric
   * Baseline accuracy, accuracy score on test set, standard deviation, number of standard deviations above baseline, same as above.
   * Result: 0.501 +/- 0.012, 14.7 std above baseline
3. Pipeline
   * Same as above.

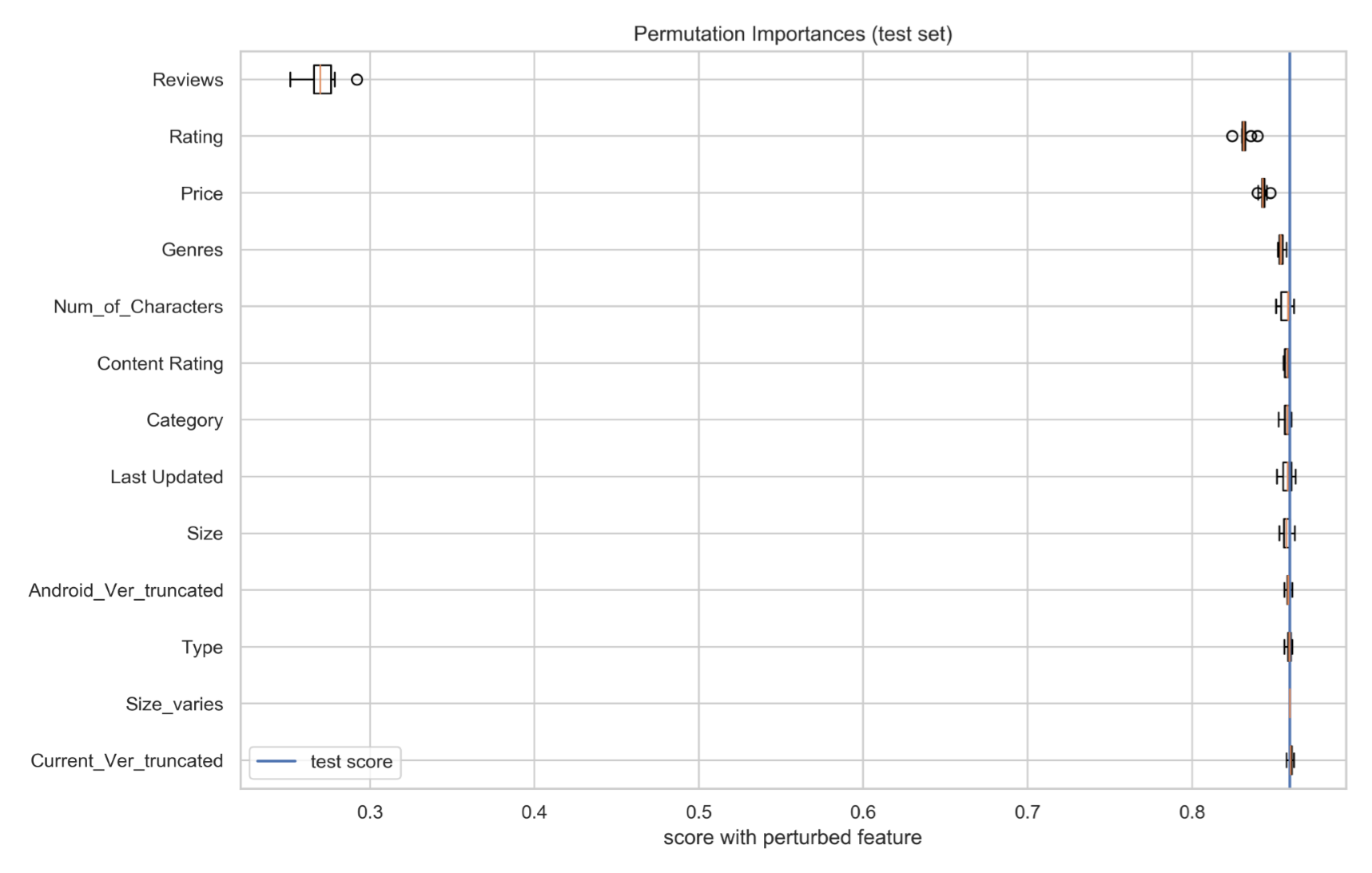
**RESULTS**

Baseline model accuracy: 51.7%, the percentage of class 0 (number of installs < 10k).

RandomForest classifier has the best performance, with average accuracy score of 57.3% and a standard deviation of 0.2%, achieving 266 standard deviations above baseline. Below is the prediction result using one RandomForest classifier model.



To break it down, the classification report shows that the f1 score for class 0 is the highest (57.3%).



From permutation test result, Number of reviews, Rating, Price, and Genres are the four most important features. When Number of reviews is permuted, the model accuracy will decrease from 86% to less than 30%.

From the results above, when an app developer wants to make a blockbuster app, he wants to shoot for more reviews, higher ratings, and make it available for free. Also, he need to choose genres that usually attract more users than others such as entertainment, video games, food & drinks.

## Code sheet

## 1. Google Play Store apps and reviews

In this Notebook, we are going to analyse the dataset (taken from Kaggle) of all the Apps in the Google Play Store The series of steps followed are :

1. Importing Packages
2. Reading Data
3. Data Preprocessing

* 3.1 Handling NULL Values
* 3.2 Handling Data Types and Values

1. Analyzing Features
2. Furthur Analysis.

* Unsupervised Methods.
* Supervised Methods. 6.Conclusions

# 1.Importing the required packages

import pandas as pd  
from scipy import stats  
import matplotlib.pyplot as plt  
import seaborn as sns  
import statsmodels.api as sm  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.ensemble import GradientBoostingRegressor  
from sklearn.ensemble import GradientBoostingClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.tree import DecisionTreeRegressor  
from sklearn.ensemble import AdaBoostRegressor  
from sklearn.ensemble import AdaBoostClassifier  
from sklearn import metrics  
from sklearn.metrics import mean\_squared\_error  
from sklearn.preprocessing import OrdinalEncoder  
from sklearn.preprocessing import scale  
from sklearn import cluster  
from sklearn.preprocessing import LabelEncoder  
from sklearn.cluster import AgglomerativeClustering  
import numpy as np  
from scipy.cluster.hierarchy import dendrogram, linkage  
from sklearn.neighbors import KNeighborsClassifier  
from flask import Flask, render\_template, request

# 2.Reading Data

Playstore=pd.read\_csv(r"C:\Users\Home\Desktop\project\googleplaystore.csv\googleplaystore.csv")  
Playstore.head()

App Category Rating \  
0 Photo Editor & Candy Camera & Grid & ScrapBook ART\_AND\_DESIGN 4.1   
1 Coloring book moana ART\_AND\_DESIGN 3.9   
2 U Launcher Lite – FREE Live Cool Themes, Hide ... ART\_AND\_DESIGN 4.7   
3 Sketch - Draw & Paint ART\_AND\_DESIGN 4.5   
4 Pixel Draw - Number Art Coloring Book ART\_AND\_DESIGN 4.3   
  
 Reviews Size Installs Type Price Content Rating \  
0 159 19M 10,000+ Free 0 Everyone   
1 967 14M 500,000+ Free 0 Everyone   
2 87510 8.7M 5,000,000+ Free 0 Everyone   
3 215644 25M 50,000,000+ Free 0 Teen   
4 967 2.8M 100,000+ Free 0 Everyone   
  
 Genres Last Updated Current Ver \  
0 Art & Design January 7, 2018 1.0.0   
1 Art & Design;Pretend Play January 15, 2018 2.0.0   
2 Art & Design August 1, 2018 1.2.4   
3 Art & Design June 8, 2018 Varies with device   
4 Art & Design;Creativity June 20, 2018 1.1   
  
 Android Ver   
0 4.0.3 and up   
1 4.0.3 and up   
2 4.0.3 and up   
3 4.2 and up   
4 4.4 and up

Playstore.shape

(10841, 13)

Playstore.columns

Index(['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type',  
 'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver',  
 'Android Ver'],  
 dtype='object')

# 3.Data Preprocessing

3.1 Handling NULL Values:

This is a very crucial step in every analysis and model, which on doing, improves the accuracy of insights and predictions.

Playstore.isnull().sum()

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

There are many NULL values in Rating, and few in Type,Content Rating and Versions.

Playstore.dropna(inplace=True) #Dropping Rows with Null values

Playstore.shape

(9360, 13)

Playstore.drop\_duplicates(inplace=True)

Playstore.shape

(8886, 13)

After removing the rows with Null values and the duplicate entries, We have got 8886 apps to analyze for their ratings and performance.

3.2 Handling Data Types of each Feature:

The data types of each feature must be changed to a proper format that can be used for analysis.

Playstore.dtypes # Displaying Data types of each feature.

App object  
Category object  
Rating float64  
Reviews object  
Size object  
Installs object  
Type object  
Price object  
Content Rating object  
Genres object  
Last Updated object  
Current Ver object  
Android Ver object  
dtype: object

The feature Reviews must be of numerical type. So we should change it.

Playstore.Reviews=Playstore.Reviews.astype('int64')#Changing to int type.

Other Features like Size, Installs, Price and Android Vers also must be of numeric type. The values they are holding must be changed to a proper format so that we can use them for analysis and plots. Example : '10000+' to 10000

* Changing the Feature : Installs

newInstalls = []  
  
for row in Playstore.Installs:  
   
 row = row[:-1]  
 newRow = row.replace(",", "")  
 newInstalls.append(float(newRow))  
   
  
Playstore.Installs = newInstalls  
  
Playstore.Installs.head()

0 10000.0  
1 500000.0  
2 5000000.0  
3 50000000.0  
4 100000.0  
Name: Installs, dtype: float64

* Changing the feature : Size

newSize = []  
  
for row in Playstore.Size:  
 newrow = row[:-1]  
 try:  
 newSize.append(float(newrow))  
 except:  
 newSize.append(0) #When it says - Size Varies.  
   
Playstore.Size = newSize  
  
Playstore.Size.head()

0 19.0  
1 14.0  
2 8.7  
3 25.0  
4 2.8  
Name: Size, dtype: float64

* Changing the feature, Price

newPrice = []  
  
for row in Playstore.Price:  
 if row!= "0":  
 newrow = float(row[1:])  
 else:  
 newrow = 0   
   
 newPrice.append(newrow)  
   
Playstore.Price = newPrice  
  
Playstore.Price.head()

0 0.0  
1 0.0  
2 0.0  
3 0.0  
4 0.0  
Name: Price, dtype: float64

* Changing the feature, Android Ver

newVer = []  
  
for row in Playstore['Android Ver']:  
 try:  
 newrow = float(row[:2])  
 except:  
 newrow = 0 # When the value is - Varies with device  
   
 newVer.append(newrow)  
   
Playstore['Android Ver'] = newVer  
  
Playstore['Android Ver'].value\_counts()

4.0 5602  
0.0 1178  
2.0 1160  
5.0 500  
3.0 246  
1.0 104  
6.0 46  
7.0 45  
8.0 5  
Name: Android Ver, dtype: int64

# 4. Analyzing Features :

4.1 Categories

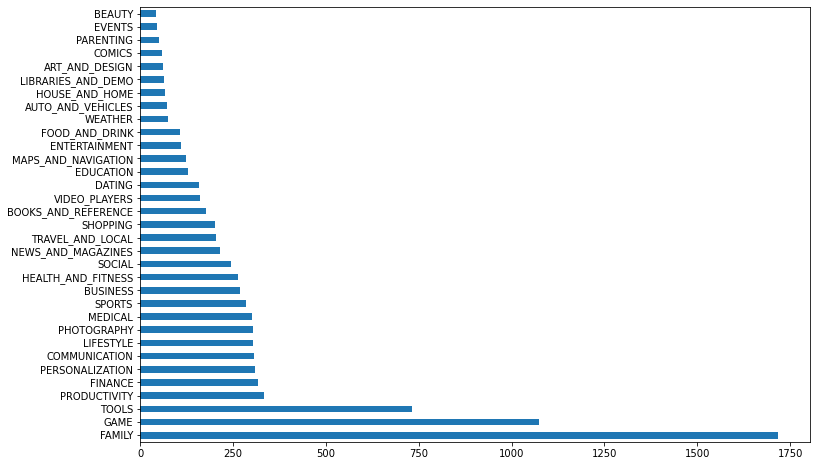
Displaying all the categories and their counts.

Playstore.Category.value\_counts()

FAMILY 1717  
GAME 1074  
TOOLS 733  
PRODUCTIVITY 334  
FINANCE 317  
PERSONALIZATION 308  
COMMUNICATION 307  
LIFESTYLE 305  
PHOTOGRAPHY 304  
MEDICAL 302  
SPORTS 286  
BUSINESS 270  
HEALTH\_AND\_FITNESS 262  
SOCIAL 244  
NEWS\_AND\_MAGAZINES 214  
TRAVEL\_AND\_LOCAL 205  
SHOPPING 202  
BOOKS\_AND\_REFERENCE 177  
VIDEO\_PLAYERS 160  
DATING 159  
EDUCATION 129  
MAPS\_AND\_NAVIGATION 124  
ENTERTAINMENT 111  
FOOD\_AND\_DRINK 106  
WEATHER 75  
AUTO\_AND\_VEHICLES 73  
HOUSE\_AND\_HOME 68  
LIBRARIES\_AND\_DEMO 64  
ART\_AND\_DESIGN 61  
COMICS 58  
PARENTING 50  
EVENTS 45  
BEAUTY 42  
Name: Category, dtype: int64

Playstore.Category.value\_counts().plot(kind='barh',figsize= (12,8))

<AxesSubplot:>



Insight : Maximum Number of Apps belong to the Family and Game Category.

4.2 Rating

Playstore.Rating.describe()

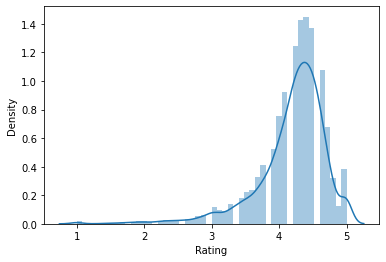
count 8886.000000  
mean 4.187959  
std 0.522428  
min 1.000000  
25% 4.000000  
50% 4.300000  
75% 4.500000  
max 5.000000  
Name: Rating, dtype: float64

Distribution Plot of 'Rating'

sns.distplot(Playstore.Rating)

C:\Users\Home\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
 warnings.warn(msg, FutureWarning)

<AxesSubplot:xlabel='Rating', ylabel='Density'>



Insight : Most of the apps, clearly hold a rating above 4.0 ! And surprisingly a lot seem to have 5.0 rating.

print("No. of Apps with full ratings: ",Playstore.Rating[Playstore['Rating'] == 5 ].count())

No. of Apps with full ratings: 271

There are 271 Apps in the store which hold 5.0 Ratings. Do all of these actually deserve it? Or are these spammed ratings? Lets analyze furthur.

4.3 Consider the Reviews:

Distribution Plot of the feature 'Reviews'

plt.figure(figsize=(10,5))  
sns.distplot(Playstore.Reviews)

C:\Users\Home\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
 warnings.warn(msg, FutureWarning)

<AxesSubplot:xlabel='Reviews', ylabel='Density'>



Let's look into those apps which have a good amount of Reviews.

Playstore[Playstore.Reviews>40000000]

App Category Rating \  
335 Messenger – Text and Video Chat for Free COMMUNICATION 4.0   
336 WhatsApp Messenger COMMUNICATION 4.4   
382 Messenger – Text and Video Chat for Free COMMUNICATION 4.0   
1670 Clash of Clans GAME 4.6   
1879 Clash of Clans GAME 4.6   
2544 Facebook SOCIAL 4.1   
2545 Instagram SOCIAL 4.5   
2604 Instagram SOCIAL 4.5   
3904 WhatsApp Messenger COMMUNICATION 4.4   
3909 Instagram SOCIAL 4.5   
3943 Facebook SOCIAL 4.1   
3986 Clash of Clans FAMILY 4.6   
4005 Clean Master- Space Cleaner & Antivirus TOOLS 4.7   
  
 Reviews Size Installs Type Price Content Rating Genres \  
335 56642847 0.0 1.000000e+09 Free 0.0 Everyone Communication   
336 69119316 0.0 1.000000e+09 Free 0.0 Everyone Communication   
382 56646578 0.0 1.000000e+09 Free 0.0 Everyone Communication   
1670 44891723 98.0 1.000000e+08 Free 0.0 Everyone 10+ Strategy   
1879 44893888 98.0 1.000000e+08 Free 0.0 Everyone 10+ Strategy   
2544 78158306 0.0 1.000000e+09 Free 0.0 Teen Social   
2545 66577313 0.0 1.000000e+09 Free 0.0 Teen Social   
2604 66577446 0.0 1.000000e+09 Free 0.0 Teen Social   
3904 69109672 0.0 1.000000e+09 Free 0.0 Everyone Communication   
3909 66509917 0.0 1.000000e+09 Free 0.0 Teen Social   
3943 78128208 0.0 1.000000e+09 Free 0.0 Teen Social   
3986 44881447 98.0 1.000000e+08 Free 0.0 Everyone 10+ Strategy   
4005 42916526 0.0 5.000000e+08 Free 0.0 Everyone Tools   
  
 Last Updated Current Ver Android Ver   
335 August 1, 2018 Varies with device 0.0   
336 August 3, 2018 Varies with device 0.0   
382 August 1, 2018 Varies with device 0.0   
1670 July 15, 2018 10.322.16 4.0   
1879 July 15, 2018 10.322.16 4.0   
2544 August 3, 2018 Varies with device 0.0   
2545 July 31, 2018 Varies with device 0.0   
2604 July 31, 2018 Varies with device 0.0   
3904 August 3, 2018 Varies with device 0.0   
3909 July 31, 2018 Varies with device 0.0   
3943 August 3, 2018 Varies with device 0.0   
3986 July 15, 2018 10.322.16 4.0   
4005 August 3, 2018 Varies with device 0.0

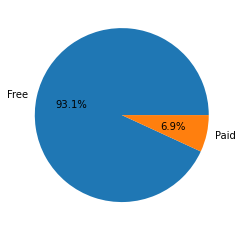
Insight : The most famous Apps like WhatsApp, Facebook and Clash of Clans are the most reviewed Apps as shown above.

Note : And I still have to figure out, how to remove the duplicate entries. My Apologies.

4.4 Type:

plt.pie(Playstore.Type.value\_counts(), labels=['Free', 'Paid'], autopct='%1.1f%%')

([<matplotlib.patches.Wedge at 0x237eca1c430>,  
 <matplotlib.patches.Wedge at 0x237eca1cb50>],  
 [Text(-1.0744351676595925, 0.2357733456018803, 'Free'),  
 Text(1.0744351566222443, -0.23577339589982083, 'Paid')],  
 [Text(-0.5860555459961413, 0.12860364305557104, '93.1%'),  
 Text(0.5860555399757695, -0.12860367049081134, '6.9%')])



Insight: 93% of the Apps are Free in the Play Store.

4.5 Price

Playstore[Playstore.Price == Playstore.Price.max()]

App Category Rating Reviews Size Installs \  
4367 I'm Rich - Trump Edition LIFESTYLE 3.6 275 7.3 10000.0   
  
 Type Price Content Rating Genres Last Updated Current Ver \  
4367 Paid 400.0 Everyone Lifestyle May 3, 2018 1.0.1   
  
 Android Ver   
4367 4.0

Insight : The most costly App in the Store is: I'm Rich - Trump Edition costing 400 Dollars!

4.6 Android Version

Playstore['Android Ver'].value\_counts()

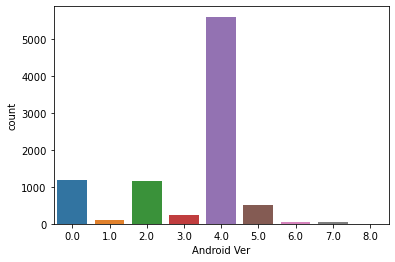
4.0 5602  
0.0 1178  
2.0 1160  
5.0 500  
3.0 246  
1.0 104  
6.0 46  
7.0 45  
8.0 5  
Name: Android Ver, dtype: int64

Count Plot of the various Versions

sns.countplot(Playstore['Android Ver'])

C:\Users\Home\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.  
 warnings.warn(

<AxesSubplot:xlabel='Android Ver', ylabel='count'>



Insight : Most of the apps support Android 4.0 and above.

# 5. Furthur Analysis

Looking at the Apps with 5.0 ratings:

Playstore\_full = Playstore[Playstore.Rating == 5]  
  
Playstore\_full.head()

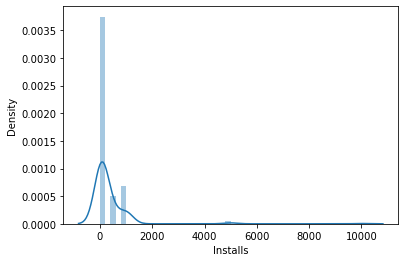
App Category Rating Reviews \  
329 Hojiboy Tojiboyev Life Hacks COMICS 5.0 15   
612 American Girls Mobile Numbers DATING 5.0 5   
615 Awake Dating DATING 5.0 2   
633 Spine- The dating app DATING 5.0 5   
636 Girls Live Talk - Free Text and Video Chat DATING 5.0 6   
  
 Size Installs Type Price Content Rating Genres Last Updated \  
329 37.0 1000.0 Free 0.0 Everyone Comics June 26, 2018   
612 4.4 1000.0 Free 0.0 Mature 17+ Dating July 17, 2018   
615 70.0 100.0 Free 0.0 Mature 17+ Dating July 24, 2018   
633 9.3 500.0 Free 0.0 Teen Dating July 14, 2018   
636 5.0 100.0 Free 0.0 Mature 17+ Dating August 1, 2018   
  
 Current Ver Android Ver   
329 2.0 4.0   
612 3.0 4.0   
615 2.2.9 4.0   
633 4.0 4.0   
636 8.2 4.0

Distribution plot of 'Installs' of Apps with 5.0 Ratings

sns.distplot(Playstore\_full.Installs)

C:\Users\Home\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
 warnings.warn(msg, FutureWarning)

<AxesSubplot:xlabel='Installs', ylabel='Density'>



Playstore\_full.Installs.value\_counts().sort\_index()

1.0 3  
5.0 8  
10.0 48  
50.0 32  
100.0 112  
500.0 27  
1000.0 37  
5000.0 3  
10000.0 1  
Name: Installs, dtype: int64

Insight : There are many Apps that have full ratings but less downloads/installs. So we can't really consider those apps as the best ones.

Consider the Apps with 5.0 Ratings and Maximum Installs :

Playstore\_full\_maxinstalls = Playstore\_full[Playstore.Installs > 1000]  
Playstore\_full\_maxinstalls[['App', 'Category', 'Installs','Rating']]

C:\Users\Home\AppData\Local\Temp\ipykernel\_14960\2678389387.py:1: UserWarning: Boolean Series key will be reindexed to match DataFrame index.  
 Playstore\_full\_maxinstalls = Playstore\_full[Playstore.Installs > 1000]

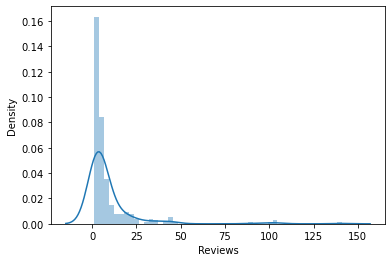
App Category Installs \  
7514 CL Keyboard - Myanmar Keyboard (No Ads) TOOLS 5000.0   
8058 Oración CX LIFESTYLE 5000.0   
8260 Superheroes, Marvel, DC, Comics, TV, Movies News COMICS 5000.0   
9511 Ek Bander Ne Kholi Dukan FAMILY 10000.0   
  
 Rating   
7514 5.0   
8058 5.0   
8260 5.0   
9511 5.0

Checking the No. of Reviews of 5.0 Rating Apps

sns.distplot(Playstore\_full.Reviews)

C:\Users\Home\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
 warnings.warn(msg, FutureWarning)

<AxesSubplot:xlabel='Reviews', ylabel='Density'>



The above distribution is clearly skewed. Apps with very few reviews easily managed to get 5.0 ratings which can be misleading. So let's filter out the ones with more than 30 reviews. These filtered ones are the apps that really stand for 5.0 rating.

Playstore\_full = Playstore\_full[Playstore.Reviews > 30]

C:\Users\Home\AppData\Local\Temp\ipykernel\_14960\2540677971.py:1: UserWarning: Boolean Series key will be reindexed to match DataFrame index.  
 Playstore\_full = Playstore\_full[Playstore.Reviews > 30]

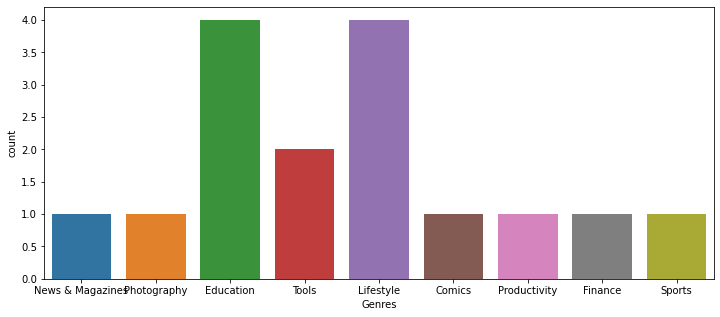
print("No. of Apps having 5.0 Rating with sufficient Reviews: ",Playstore\_full.App.count())

No. of Apps having 5.0 Rating with sufficient Reviews: 16

plt.figure(figsize=(12,5))  
sns.countplot(Playstore\_full.Genres)

C:\Users\Home\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.  
 warnings.warn(

<AxesSubplot:xlabel='Genres', ylabel='count'>



Insight : Apps related to Education, LifeStyle and Tools seem to fetch full Ratings with sufficient number of reviews.

sns.countplot(Playstore\_full.Price)

C:\Users\Home\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.  
 warnings.warn(

<AxesSubplot:xlabel='Price', ylabel='count'>



Insight : All the Apps with 5.0 ratings are Free to install.

# Unsupervised Methods

x=Playstore.drop(["Category","Installs"],axis=1)  
y=Playstore["Installs"]

x

App Rating Reviews \  
0 Photo Editor & Candy Camera & Grid & ScrapBook 4.1 159   
1 Coloring book moana 3.9 967   
2 U Launcher Lite – FREE Live Cool Themes, Hide ... 4.7 87510   
3 Sketch - Draw & Paint 4.5 215644   
4 Pixel Draw - Number Art Coloring Book 4.3 967   
... ... ... ...   
10834 FR Calculator 4.0 7   
10836 Sya9a Maroc - FR 4.5 38   
10837 Fr. Mike Schmitz Audio Teachings 5.0 4   
10839 The SCP Foundation DB fr nn5n 4.5 114   
10840 iHoroscope - 2018 Daily Horoscope & Astrology 4.5 398307   
  
 Size Type Price Content Rating Genres \  
0 19.0 Free 0.0 Everyone Art & Design   
1 14.0 Free 0.0 Everyone Art & Design;Pretend Play   
2 8.7 Free 0.0 Everyone Art & Design   
3 25.0 Free 0.0 Teen Art & Design   
4 2.8 Free 0.0 Everyone Art & Design;Creativity   
... ... ... ... ... ...   
10834 2.6 Free 0.0 Everyone Education   
10836 53.0 Free 0.0 Everyone Education   
10837 3.6 Free 0.0 Everyone Education   
10839 0.0 Free 0.0 Mature 17+ Books & Reference   
10840 19.0 Free 0.0 Everyone Lifestyle   
  
 Last Updated Current Ver Android Ver   
0 January 7, 2018 1.0.0 4.0   
1 January 15, 2018 2.0.0 4.0   
2 August 1, 2018 1.2.4 4.0   
3 June 8, 2018 Varies with device 4.0   
4 June 20, 2018 1.1 4.0   
... ... ... ...   
10834 June 18, 2017 1.0.0 4.0   
10836 July 25, 2017 1.48 4.0   
10837 July 6, 2018 1.0 4.0   
10839 January 19, 2015 Varies with device 0.0   
10840 July 25, 2018 Varies with device 0.0   
  
[8886 rows x 11 columns]

y

0 10000.0  
1 500000.0  
2 5000000.0  
3 50000000.0  
4 100000.0  
 ...   
10834 500.0  
10836 5000.0  
10837 100.0  
10839 1000.0  
10840 10000000.0  
Name: Installs, Length: 8886, dtype: float64

x\_transformed= OrdinalEncoder().fit\_transform(x)

# Preparing the data- Scaling/ Handling the data in such way they can belong in a spesific range  
# and represent the same degree of difference  
scaled\_data= scale(x\_transformed)

scaled\_data

array([[ 0.70909178, -0.1697994 , -0.99556062, ..., -0.20600172,  
 -1.29806986, 0.50615525],  
 [-0.84620985, -0.55426557, -0.70643486, ..., -0.44782642,  
 -0.33436335, 0.50615525],  
 [ 1.35137924, 0.98359911, 0.94857861, ..., -1.52504552,  
 -0.93653736, 0.50615525],  
 ...,  
 [-0.18273877, 1.56029837, -1.07530172, ..., 0.18657084,  
 -1.30263178, 0.50615525],  
 [ 1.25520559, 0.59913294, -1.01871126, ..., -0.40071771,  
 1.52462079, -2.12292294],  
 [ 1.6568471 , 0.59913294, 1.45892159, ..., 0.04524472,  
 1.52462079, -2.12292294]])

len(np.unique(y))

19

y.unique()

array([1.e+04, 5.e+05, 5.e+06, 5.e+07, 1.e+05, 5.e+04, 1.e+06, 1.e+07,  
 5.e+03, 1.e+08, 1.e+09, 1.e+03, 5.e+08, 1.e+02, 5.e+02, 1.e+01,  
 5.e+00, 5.e+01, 1.e+00])

# Hierarchical agglomerative clustering - bottom-up approach  
n\_samples, n\_features = scaled\_data.shape   
n\_digits = len(np.unique(y))   
model = cluster.AgglomerativeClustering(n\_clusters=n\_digits, linkage="average", affinity="cosine")   
model.fit(scaled\_data)

AgglomerativeClustering(affinity='cosine', linkage='average', n\_clusters=19)

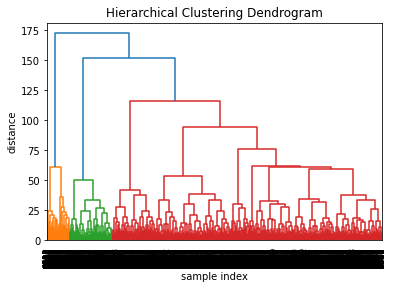
print (model.labels\_)

[10 8 10 ... 8 1 1]

print (metrics.silhouette\_score(scaled\_data,model.labels\_))  
print (metrics.completeness\_score(y, model.labels\_))  
print (metrics.homogeneity\_score(y, model.labels\_))

0.08047605967153028  
0.13797390842527352  
0.14807731311542197

# Creating Hierarchical Clustering Dendrogram  
  
model= linkage(scaled\_data, "ward")  
plt.figure()  
plt.title("Hierarchical Clustering Dendrogram")  
plt.xlabel("sample index")  
plt.ylabel("distance")  
dendrogram(model, leaf\_rotation=90., leaf\_font\_size=8.)  
plt.show()



n\_samples, n\_features = scaled\_data.shape  
n\_digits = len(np.unique(y))  
for k in range(2, 15):   
 kmeans = cluster.KMeans(n\_clusters=k)   
 kmeans.fit(scaled\_data)   
 print(k)   
 print(metrics.silhouette\_score(scaled\_data, kmeans.labels\_))  
 print(metrics.completeness\_score(y, kmeans.labels\_))   
 print(metrics.homogeneity\_score(y, kmeans.labels\_))   
   
# different results on every iteration because we are using random starting points# best score seems to be when k=13 (sometimes when k=14)

2  
0.19512666122121675  
0.1772352227974184  
0.03089224345543299  
3  
0.22693873862737815  
0.18376532635627083  
0.048231366978553895  
4  
0.20706117183915493  
0.12963878219374478  
0.052500007995839953  
5  
0.16086197078615597  
0.22820485482709713  
0.13117519237203146  
6  
0.16520819494468553  
0.2113986064852894  
0.13988409988255615  
7  
0.15419342343706186  
0.18920862392388837  
0.14131633191044074  
8  
0.15405428827976267  
0.18535488765252955  
0.14195426292681435  
9  
0.15037331709339014  
0.16327097474889832  
0.13533994844369437  
10  
0.1473226674950332  
0.1559213811414562  
0.13750415956020437  
11  
0.14824603398480019  
0.15109421622182687  
0.13828370501171433  
12  
0.14716036979146047  
0.1608314884125856  
0.15237226229854367  
13  
0.1478442561310819  
0.15297141870638786  
0.1499225949076283  
14  
0.15412580983981658  
0.1635160354839918  
0.16087575150100397

for k in range(2, 15):   
 kmeans = cluster.KMeans(n\_clusters=k)   
 kmeans.fit(scaled\_data)   
 print(k)   
 print(metrics.silhouette\_score(scaled\_data, kmeans.labels\_))  
 y\_silhouette=metrics.silhouette\_score(scaled\_data, kmeans.labels\_)  
   
 print(metrics.completeness\_score(y, kmeans.labels\_))   
 y\_completeness=metrics.completeness\_score(y, kmeans.labels\_)  
   
   
 print(metrics.homogeneity\_score(y, kmeans.labels\_))   
 y\_homogeneity=(metrics.homogeneity\_score(y, kmeans.labels\_))

2  
0.3969678059299939  
0.17759958085209232  
0.017735152307616384  
3  
0.22693873862737815  
0.18376532635627083  
0.048231366978553895  
4  
0.20690675191334595  
0.13055128214621295  
0.052949882942160555  
5  
0.16105116752228876  
0.24353546644035054  
0.1396755519605508  
6  
0.16519746465079996  
0.21178999799603918  
0.14014308668522799  
7  
0.15431275178090434  
0.19136058814333842  
0.14289068853743483  
8  
0.15420923058054745  
0.19193664016562065  
0.14692298250425867  
9  
0.15049162922810574  
0.16551864413337689  
0.13717101364611178  
10  
0.15466917418786816  
0.1715265095806062  
0.14631323059968035  
11  
0.14605325977429187  
0.17113008108444344  
0.15675302002614558  
12  
0.15385481101460324  
0.15658324691393208  
0.14680909182088422  
13  
0.14626512200188413  
0.15597113088637585  
0.15166684009983625  
14  
0.14241619230125174  
0.15194470755306572  
0.15348242429019202

print ("silhouette scores are:\n{}".format(y\_silhouette))

silhouette scores are:  
0.14241619230125174

print ("completeness scores are:\n{}".format(y\_completeness))

completeness scores are:  
0.15194470755306572

print ("homogeneity scores are:\n{}".format(y\_homogeneity))

homogeneity scores are:  
0.15348242429019202

# Supervised Methods

x=Playstore[["Rating","Reviews","Size","Price","Android Ver"]]  
y=Playstore["Installs"]

x

Rating Reviews Size Price Android Ver  
0 4.1 159 19.0 0.0 4.0  
1 3.9 967 14.0 0.0 4.0  
2 4.7 87510 8.7 0.0 4.0  
3 4.5 215644 25.0 0.0 4.0  
4 4.3 967 2.8 0.0 4.0  
... ... ... ... ... ...  
10834 4.0 7 2.6 0.0 4.0  
10836 4.5 38 53.0 0.0 4.0  
10837 5.0 4 3.6 0.0 4.0  
10839 4.5 114 0.0 0.0 0.0  
10840 4.5 398307 19.0 0.0 0.0  
  
[8886 rows x 5 columns]

y

0 10000.0  
1 500000.0  
2 5000000.0  
3 50000000.0  
4 100000.0  
 ...   
10834 500.0  
10836 5000.0  
10837 100.0  
10839 1000.0  
10840 10000000.0  
Name: Installs, Length: 8886, dtype: float64

x=x.values#attributes  
y=y.values#target

supervised\_x\_transformed= OrdinalEncoder().fit\_transform(x)

from sklearn.model\_selection import train\_test\_split  
x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.05,random\_state=0)

print("LOGISTIC REGRESSION")   
lm = LogisticRegression()  
lm.fit(x\_train,y\_train)  
lm.predict\_proba(x\_test)

LOGISTIC REGRESSION

C:\Users\Home\anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.  
  
Increase the number of iterations (max\_iter) or scale the data as shown in:  
 https://scikit-learn.org/stable/modules/preprocessing.html  
Please also refer to the documentation for alternative solver options:  
 https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression  
 n\_iter\_i = \_check\_optimize\_result(

array([[5.25798346e-02, 5.25798362e-02, 5.25798836e-02, ...,  
 5.26902272e-02, 5.26900476e-02, 5.26900206e-02],  
 [5.10676743e-02, 5.10676870e-02, 5.10678206e-02, ...,  
 5.44265682e-02, 5.44257159e-02, 5.44261984e-02],  
 [4.96671230e-02, 4.96671464e-02, 4.96674611e-02, ...,  
 5.60413765e-02, 5.60394373e-02, 5.60402982e-02],  
 ...,  
 [3.91308700e-25, 3.91386575e-25, 3.92139866e-25, ...,  
 1.52053012e-01, 1.50199471e-01, 1.51383238e-01],  
 [4.20590436e-02, 4.20591116e-02, 4.20598446e-02, ...,  
 6.49606224e-02, 6.49537751e-02, 6.49577216e-02],  
 [5.23051376e-02, 5.23051406e-02, 5.23051696e-02, ...,  
 5.30054471e-02, 5.30052711e-02, 5.30053695e-02]])

print(lm.intercept\_)

[-1.91010824e-08 -1.86636395e-08 -1.43621202e-08 -1.53099251e-08  
 1.53148372e-09 -5.61351098e-09 2.85047647e-08 9.83904344e-09  
 4.74319938e-08 1.26459705e-08 5.79624760e-08 1.59005832e-08  
 6.12621885e-08 1.93353921e-09 2.18954076e-08 -3.65591831e-08  
 -3.72893638e-08 -5.42849829e-08 -5.77236426e-08]

print(lm.coef\_)

[[-7.81762868e-08 -1.63190400e-04 -6.79484628e-07 -2.90855771e-08  
 -6.66033671e-08]  
 [-7.62442471e-08 -1.63189118e-04 -6.78821221e-07 -2.92663884e-08  
 -6.41974276e-08]  
 [-5.62239466e-08 -1.63176729e-04 -5.42048229e-07 -2.20573192e-08  
 -4.94701880e-08]  
 [-6.15899679e-08 -1.63171318e-04 -4.65838787e-07 -2.69778476e-08  
 -5.26781414e-08]  
 [ 1.17977818e-08 -1.63020251e-04 3.67252415e-08 1.59652213e-08  
 1.01673940e-08]  
 [-2.20971619e-08 -1.62963965e-04 -2.92881822e-07 -2.13794285e-08  
 -1.60794958e-08]  
 [ 1.14776470e-07 -1.61139542e-04 9.29701868e-07 1.38441135e-07  
 1.04210019e-07]  
 [ 3.78823420e-08 -1.60047575e-04 8.35889511e-07 7.37489779e-08  
 3.59640883e-08]  
 [ 1.90027798e-07 -1.39245554e-04 2.21358583e-06 1.13942664e-07  
 1.66453313e-07]  
 [ 5.02913681e-08 -1.32260946e-04 4.54187885e-07 1.24832731e-08  
 4.30643434e-08]  
 [ 2.38573368e-07 9.55994133e-05 1.80267452e-06 2.81254445e-08  
 1.95638743e-07]  
 [ 6.74828123e-08 1.71848720e-04 2.95140781e-07 -2.69876478e-08  
 5.29492008e-08]  
 [ 2.59568859e-07 1.84747552e-04 1.67806797e-06 -2.90084075e-08  
 1.97986088e-07]  
 [ 9.74548857e-09 1.85526137e-04 -2.29847149e-07 -3.28742983e-08  
 -5.66022183e-09]  
 [ 9.85877678e-08 1.86683760e-04 3.09826987e-07 -3.21092243e-08  
 4.72671425e-08]  
 [-1.53116040e-07 1.86646104e-04 -1.21201133e-06 -3.31761365e-08  
 -1.20032963e-07]  
 [-1.55177019e-07 1.86820395e-04 -1.11257819e-06 -3.32848483e-08  
 -1.26862437e-07]  
 [-2.30316170e-07 1.86741364e-04 -1.60071617e-06 -3.32329670e-08  
 -1.69824502e-07]  
 [-2.45793216e-07 1.86791952e-04 -1.74157306e-06 -3.32666260e-08  
 -1.82291589e-07]]

predicted = lm.predict(x\_test)  
print(metrics.classification\_report(y\_test, predicted))   
print(metrics.confusion\_matrix(y\_test, predicted))

precision recall f1-score support  
  
 10.0 0.00 0.00 0.00 1  
 50.0 0.00 0.00 0.00 1  
 100.0 0.00 0.00 0.00 17  
 500.0 0.00 0.00 0.00 11  
 1000.0 0.00 0.00 0.00 41  
 5000.0 0.00 0.00 0.00 25  
 10000.0 0.00 0.00 0.00 72  
 50000.0 0.00 0.00 0.00 22  
 100000.0 0.00 0.00 0.00 45  
 500000.0 0.00 0.00 0.00 25  
 1000000.0 0.00 0.00 0.00 66  
 5000000.0 0.00 0.00 0.00 36  
 10000000.0 0.00 0.00 0.00 55  
 50000000.0 0.00 0.00 0.00 13  
 100000000.0 0.03 1.00 0.06 10  
 500000000.0 0.00 0.00 0.00 3  
1000000000.0 0.00 0.00 0.00 2  
  
 accuracy 0.02 445  
 macro avg 0.00 0.06 0.00 445  
weighted avg 0.00 0.02 0.00 445  
  
[[ 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 7 0 10 0 0 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 5 0 5 0 1 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 7 0 30 0 4 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 4 0 15 0 6 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 2 0 24 0 46 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 2 0 20 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 1 0 44 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 25 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 66 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 36 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 55 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 13 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0]]

C:\Users\Home\anaconda3\lib\site-packages\sklearn\metrics\\_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, msg\_start, len(result))  
C:\Users\Home\anaconda3\lib\site-packages\sklearn\metrics\\_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, msg\_start, len(result))  
C:\Users\Home\anaconda3\lib\site-packages\sklearn\metrics\\_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, msg\_start, len(result))

print("KNN")  
model = KNeighborsClassifier()  
model.fit(x\_train,y\_train)   
print(model)

KNN  
KNeighborsClassifier()

predicted= model.predict(x\_test)  
print (metrics.classification\_report(y\_test, predicted))  
print (metrics.confusion\_matrix(y\_test, predicted))

precision recall f1-score support  
  
 10.0 0.25 1.00 0.40 1  
 50.0 0.00 0.00 0.00 1  
 100.0 0.47 0.41 0.44 17  
 500.0 0.20 0.09 0.13 11  
 1000.0 0.40 0.41 0.41 41  
 5000.0 0.31 0.40 0.35 25  
 10000.0 0.56 0.53 0.54 72  
 50000.0 0.20 0.18 0.19 22  
 100000.0 0.40 0.53 0.46 45  
 500000.0 0.20 0.12 0.15 25  
 1000000.0 0.48 0.58 0.52 66  
 5000000.0 0.26 0.19 0.22 36  
 10000000.0 0.64 0.62 0.63 55  
 50000000.0 0.62 0.38 0.48 13  
 100000000.0 0.73 0.80 0.76 10  
 500000000.0 0.50 0.67 0.57 3  
1000000000.0 1.00 0.50 0.67 2  
  
 accuracy 0.45 445  
 macro avg 0.43 0.44 0.41 445  
weighted avg 0.44 0.45 0.44 445  
  
[[ 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]  
 [ 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]  
 [ 1 0 7 2 6 1 0 0 0 0 0 0 0 0 0 0 0]  
 [ 1 0 0 1 5 1 3 0 0 0 0 0 0 0 0 0 0]  
 [ 1 1 7 2 17 7 6 0 0 0 0 0 0 0 0 0 0]  
 [ 0 0 0 0 7 10 8 0 0 0 0 0 0 0 0 0 0]  
 [ 0 0 0 0 7 12 38 10 5 0 0 0 0 0 0 0 0]  
 [ 0 0 0 0 0 1 7 4 9 1 0 0 0 0 0 0 0]  
 [ 0 0 0 0 0 0 5 5 24 4 6 1 0 0 0 0 0]  
 [ 0 0 0 0 0 0 0 1 11 3 8 1 1 0 0 0 0]  
 [ 0 0 0 0 0 0 1 0 10 6 38 9 2 0 0 0 0]  
 [ 0 0 0 0 0 0 0 0 1 1 18 7 9 0 0 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 9 9 34 2 1 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 7 5 1 0 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 2 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 0]  
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1]]

print (metrics.accuracy\_score(y\_test, predicted))

0.449438202247191

modeldtc = DecisionTreeClassifier()  
modeldtc.fit(x\_train,y\_train)

DecisionTreeClassifier()

modeldtc.score(x\_test,y\_test)

0.5123595505617977

dist = {}  
for i in [0,1,42,101,110,575,684,786,506,454,455]:  
 rf\_reg = RandomForestClassifier(random\_state=i)  
 rf\_reg.fit(x\_train, y\_train)  
 dist[i] = rf\_reg.score(x\_test,y\_test)

dist

{0: 0.5662921348314607,  
 1: 0.5617977528089888,  
 42: 0.5685393258426966,  
 101: 0.5887640449438202,  
 110: 0.5662921348314607,  
 575: 0.5707865168539326,  
 684: 0.5617977528089888,  
 786: 0.5707865168539326,  
 506: 0.5685393258426966,  
 454: 0.5730337078651685,  
 455: 0.550561797752809}

rf\_reg = RandomForestClassifier(random\_state=454)  
rf\_reg.fit(x\_train,y\_train)

RandomForestClassifier(random\_state=454)

rf\_reg.score(x\_test,y\_test)

0.5730337078651685

def LogisticReg(x\_train,y\_train,x\_test,y\_test):  
 modelLogistic=LogisticRegression()  
 modelLogistic.fit(x\_train,y\_train)  
 return modelLogistic.score(x\_test,y\_test)  
 #return mean\_squared\_error(y\_test, modelLogistic.predict(x\_test), squared=False)  
def DecisionTreeClf(x\_train,y\_train,x\_test,y\_test):  
 modeldtc = DecisionTreeClassifier(random\_state=42)  
 modeldtc.fit(x\_train,y\_train)  
 return modeldtc.score(x\_test,y\_test)  
 #return mean\_squared\_error(y\_test, modeldtc.predict(x\_test), squared=False)  
def RandomForestClf(x\_train,y\_train,x\_test,y\_test):  
 modelrfc = RandomForestClassifier(max\_depth=2, random\_state=42)  
 modelrfc.fit(x\_train,y\_train)  
 return modelrfc.score(x\_test,y\_test)  
 #return mean\_squared\_error(y\_test, modelrfc.predict(x\_test), squared=False)  
def GradientBoostingClf(x\_train,y\_train,x\_test,y\_test):  
 modelgbc = GradientBoostingClassifier(n\_estimators=100,max\_depth=2, random\_state=42)  
 modelgbc.fit(x\_train,y\_train)  
 return modelgbc.score(x\_test,y\_test)  
 #return mean\_squared\_error(y\_test, modelgbc.predict(x\_test), squared=False)  
def AdaBoostClf(x\_train,y\_train,x\_test,y\_test):  
 modelabc = AdaBoostClassifier(random\_state=45)  
 modelabc.fit(x\_train,y\_train)  
 return modelabc.score(x\_test,y\_test)

reg = [LogisticReg(x\_train,y\_train,x\_test,y\_test),DecisionTreeClf(x\_train,y\_train,x\_test,y\_test),RandomForestClf(x\_train,y\_train,x\_test,y\_test),GradientBoostingClf(x\_train,y\_train,x\_test,y\_test),AdaBoostClf(x\_train,y\_train,x\_test,y\_test)]

C:\Users\Home\anaconda3\lib\site-packages\sklearn\linear\_model\\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.  
  
Increase the number of iterations (max\_iter) or scale the data as shown in:  
 https://scikit-learn.org/stable/modules/preprocessing.html  
Please also refer to the documentation for alternative solver options:  
 https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression  
 n\_iter\_i = \_check\_optimize\_result(

reg

[0.02247191011235955,  
 0.4966292134831461,  
 0.42247191011235957,  
 0.5617977528089888,  
 0.2898876404494382]

app = Flask(\_\_name\_\_,template\_folder=r'C:\Users\Home\Desktop\template')  
Playstore = pd.read\_csv(r"C:\Users\Home\Desktop\project\googleplaystore.csv\googleplaystore.csv")  
Playstore = Playstore.drop(['App','Last Updated','Current Ver','Installs'],axis=1)  
l = list(Playstore.columns)  
k = list(Playstore.columns)  
@app.route('/')  
def index():  
 return render\_template('index.html')  
@app.route('/getting', methods=['POST'])  
def getting():  
 name = request.form['name']  
 cat = request.form['cat']  
 rat = request.form['rat']  
 rev = request.form['rev']  
 siz = request.form['size']  
 typ = request.form['type']  
 pri = request.form['price']  
 cr = request.form['cr']  
 gr = request.form['gr']  
 ar = request.form['ar']  
 df5['App'] = name  
 df2['Rating'] = rat  
 df5['Rating']= rat  
 df5['Reviews']= rev  
 df5['Price']= pri  
 df2['Reviews'] = rev  
 df2['Price'] = pri  
 for j in range(0,len(k)):  
 if(l[0]+'\_'+cat == k[j]):  
 df2[l[0]+'\_'+cat ] = 1  
 df5[l[0]] = cat  
 for j in range(0,len(k)):  
 if(l[3]+'\_'+siz == k[j]):  
 df2[l[3]+'\_'+siz] = 1  
 df5[l[3]] = siz  
 for j in range(0,len(k)):  
 if(l[4]+'\_'+typ == k[j]):  
 df2[l[4]+'\_'+typ ] = 1  
 df5[l[4]] = typ  
 for j in range(0,len(k)):  
 if(l[6]+'\_'+cr == k[j]):  
 df2[l[6]+'\_'+cr] = 1  
 df5[l[6]] = cr  
 for j in range(0,len(k)):  
 if(l[7]+'\_'+gr == k[j]):  
 df2[l[7]+'\_'+gr ] = 1  
 df5[l[7]] = gr  
 for j in range(0,len(k)):  
 if(l[8]+'\_'+ar == k[j]):  
 df2[l[8]+'\_'+ar ] = 1  
 df5[l[8]] = ar  
 df2.fillna(0,inplace=True)  
 return render\_template('op.html',ds = df5.to\_html(),pred = model.predict(df2),name = name)  
app.run()

\* Serving Flask app "\_\_main\_\_" (lazy loading)  
 \* Environment: production  
 WARNING: This is a development server. Do not use it in a production deployment.  
 Use a production WSGI server instead.  
 \* Debug mode: off

\* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)  
127.0.0.1 - - [03/Aug/2022 08:32:00] "GET / HTTP/1.1" 200 -  
127.0.0.1 - - [03/Aug/2022 08:32:00] "GET /uploads/media/default/0001/01/b5edc1bad4dc8c20291c8394527cb2c5b43ee13c.jpeg HTTP/1.1" 404 -  
127.0.0.1 - - [03/Aug/2022 08:32:00] "GET /favicon.ico HTTP/1.1" 404 -

# 6.Conclusions

In this project, we analyzed data about the App Store and Google Play mobile apps with the goal of recommending an app profile that can be profitable for both markets.