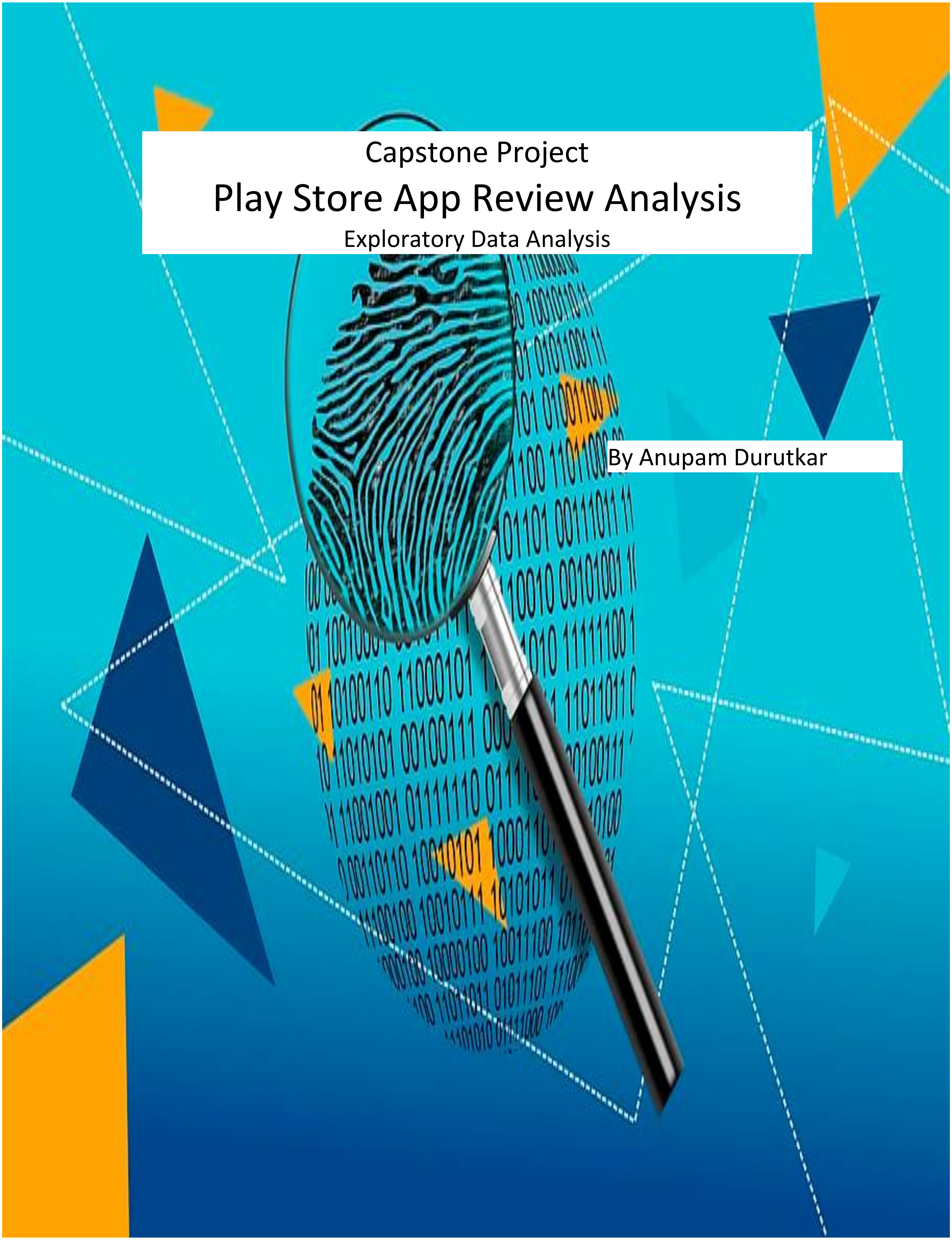
Capstone Project

Play Store App Review Analysis

Exploratory Data Analysis

By Anupam Durutkar



Description of the project goals:

The objective of this project is to deliver insights to understand customer demands better and thus help developers to popularize the product. It is of 10k Play Store apps for analyzing the Android market. This dataset contains details of different applications and reviews from different users.

Discussion of Google play store dataset will involve various steps such as:

- Loading the data into a data frame
- Cleaning the data
- Extracting statistics from the dataset
- Exploratory analysis and visualizations
- Questions that can be asked from the dataset
- Conclusion

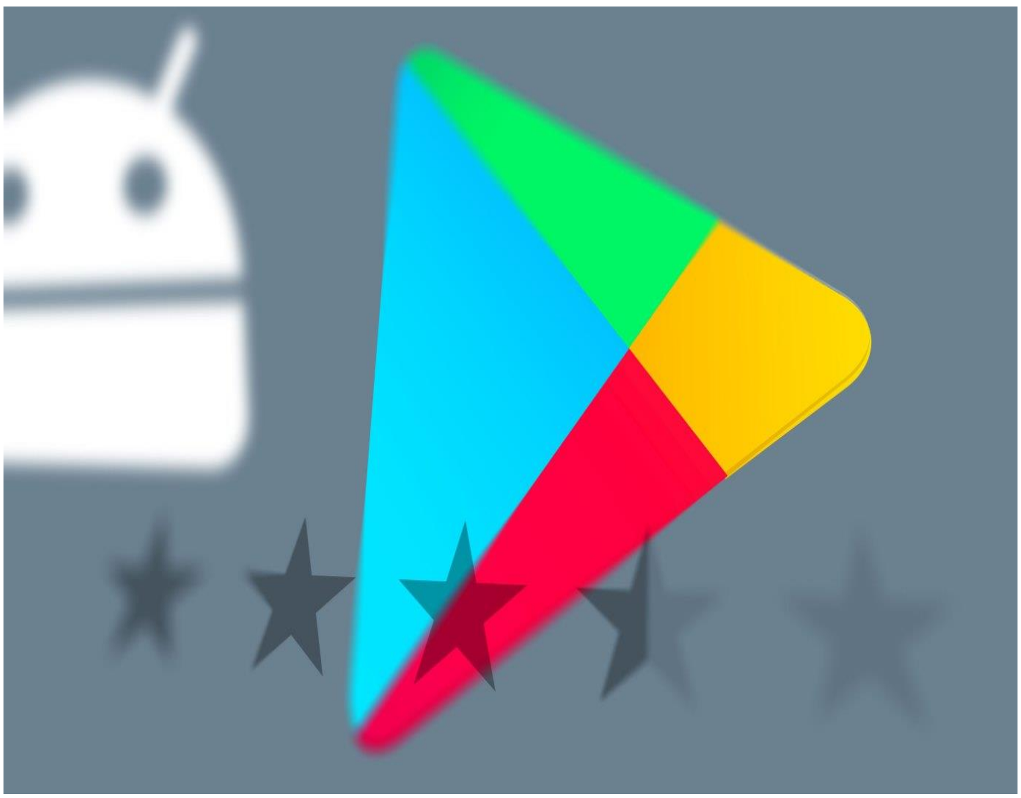


Table of Contents

1. Introduction to data

Mobile apps are everywhere. They are easy to create and can be money making. Because of these two factors, more and more apps are being developed. In this notebook, we will do a comprehensive analysis of the Android app market by comparing over ten thousand apps in Google Play across different categories. We'll look for insights in the data to devise strategies to drive growth and retention.

Following two files in the dataset which consists different of features:

1.1 datasets/play_store_data.csv:

This file contains all the details of the apps on Google Play. 13 features describe a given app.

- **App:** Name of the app
- **Category:** Category of the app. Some examples are ART_AND_DESIGN, FINANCE, COMICS, BEAUTY etc.
- **Rating:** The current average rating (out of 5) of the app on Google Play
- **Reviews:** Number of user reviews given on the app
- **Size:** Size of the app in MB (megabytes)
- **Installs:** Number of times the app was downloaded from Google Play
- **Type:** Whether the app is paid or free
- **Price:** Price of the app in US$
- **Content Rating:** A content rating (also known as maturity rating) rates the suitability of TV broadcasts, movies, comic books, or video games to its audience. To show which age group is suitable to view media and entertainment.
- **Genres:** A category of artistic, musical, or literary composition characterized by a particular style, form, or content
- **Last Updated**: Date on which the app was last updated on Google Play
- **Current Ver**: Current Version means a version of the software that is currently being supported by its publisher.
- **Android Ver:** Android versions (codenames) are used to describe the various updates for the open-source Android mobile operating system.

1.2 datasets/user_reviews.csv:

This file contains a random sample of 100 most helpful first user reviews for each app. The distribution of positive and negative reviews in each category has been pre-processed and passed through a sentiment analyzer.

- **App:** Name of the app on which the user review was provided. Matches the App column of the play_store_data.csv file
- **Translated Review:** The pre-processed user review text.
- **Sentiment:** Sentiment category of the user review - Positive, Negative or Neutral.
- **Sentiment Polarity**: Sentiment score of the user review. It lies between [-1,1]. A higher score denotes a more positive sentiment.

2. Loading and exploring the data into a data frame

2.1 Importing required packages and dataset:

In this project, data set from a given file are loaded and named '*play_store_data.csv*' & '*user_reviews.csv*'.
The following methods are used in imported data and explore data in google colab:

Mounted Google Drive

```
[ ] from google.colab import drive
    drive.mount('/content/drive', force_remount=True)

    Mounted at /content/drive
```

Importing Dataset

Read the datasets into dataframes

```
[ ] play_store_df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Project_folder/Play Store Data.csv")
    user_ratings_df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Project_folder/User Reviews.csv" ,encoding='utf8')

    ##Copying the database for comparison Original and clean version
    play_store_df_original = play_store_df.copy()
    play_store_df_original.shape

    (10841, 13)
```

```
[ ] ## Display the Play Store App data
    play_store_df.head()
```

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

```
[ ] ## Display the User Ratings data
    user_ratings_df.head()
```

	App	Translated_Review	Sentiment	Sentiment_Polarity	Sentiment_Subjectivity
0	10 Best Foods for You	I like eat delicious food. That's I'm cooking ...	Positive	1.00	0.533333
1	10 Best Foods for You	This help eating healthy exercise regular basis	Positive	0.25	0.288462
2	10 Best Foods for You	NaN	NaN	NaN	NaN
3	10 Best Foods for You	Works great especially going grocery store	Positive	0.40	0.875000
4	10 Best Foods for You	Best idea us	Positive	1.00	0.300000

2.2 Explore the structure of the datasets:

This step is about getting to know the data and understanding what must be done before the data becomes useful in a particular context. This can be done by reading the CSV file and doing an initial statistical analysis. Following method used to understand the dataset and some initial analysis:

Explore the structure of the datasets

```
[6] ## Play Store App Data
print(f"The length of the 'play_store_df' datasets is: {len(play_store_df)}")
print(f"The total number of rows and columns in play_store_df datasets is:{play_store_df.shape}")
print(f"The total number of unique Apps in play_store_df datasets is:{play_store_df['App'].nunique()}")
```

```
The length of the 'play_store_df' datasets is: 10841
The total number of rows and columns in play_store_df datasets is:(10841, 13)
The total number of unique Apps in play_store_df datasets is:9660
```

```
## User Reviews Data
print(f"The length of the user_ratings_df datasets is:",len(user_ratings_df))
print(f"The total number of rows and columns in user_ratings_df datasets is",user_ratings_df.shape)
print(f"The total number of unique Apps in user_ratings_df datasets is",user_ratings_df["App"].nunique())
```

```
The length of the user_ratings_df datasets is: 64295
The total number of rows and columns in user_ratings_df datasets is (64295, 5)
The total number of unique Apps in user_ratings_df datasets is 1074
```

```
[8] #The described method will help to see how data has been spread for numerical values.
#We can clearly see the minimum value, mean values, different percentile values, and maximum values.
play_store_df.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
App	10841	9660	ROBLOX	9	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Category	10841	34	FAMILY	1972	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Rating	9367	NaN	NaN	NaN	4.19334	0.537431	1	4	4.3	4.5	19
Reviews	10841	6002	0	596	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Size	10841	462	Varies with device	1695	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Installs	10841	22	1,000,000+	1579	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Type	10840	3	Free	10039	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Price	10841	93	0	10040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Content Rating	10840	6	Everyone	8714	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Genres	10841	120	Tools	842	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Last Updated	10841	1378	August 3, 2018	326	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Current Ver	10833	2832	Varies with device	1459	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Android Ver	10838	33	4.1 and up	2451	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
[49] user_ratings_df.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
App	64295	1074	Bowmasters	320	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Translated_Review	37427	27994	Good	247	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Sentiment	37432	3	Positive	23998	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Sentiment_Polarity	37432	NaN	NaN	NaN	0.182146	0.351301	-1	0	0.15	0.4	1
Sentiment_Subjectivity	37432	NaN	NaN	NaN	0.492704	0.259949	0	0.357143	0.514286	0.65	1

3. Cleaning the data

Before beginning our analysis, we need to make sure the data we analyze is accurate, otherwise the results of our analysis will be wrong.

3.1 Handling the Null values in the dataset:

One of the main steps in data pre-processing is handling missing data. Missing data means absence of observations in columns that can be caused while procuring the data, lack of information, incomplete results etc.

Feeding missing data could lead to wrong prediction or classification. Hence it is necessary to identify missing values and treat them

The function is created to calculate missing values by column# with a respective feature.

```
[9]  ## Let's get the ratio of missing values for each feature
     ## Creating Function to calculate missing values by column# Funct
     def missing_values_table(df):
             # Total missing values
             mis_val = df.isnull().sum()

             # Percentage of missing values
             mis_val_percent = 100 * df.isnull().sum() / len(df)

             # Make a table with the results
             mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

             # Rename the columns
             mis_val_table_ren_columns = mis_val_table.rename(
             columns = {0 : 'Missing Values', 1 : '% of Total Values'})

             # Sort the table by percentage of missing descending
             mis_val_table_ren_columns = mis_val_table_ren_columns[
                 mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
             '% of Total Values', ascending=False).round(2)

             # Print some summary information
             print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
                 "There are " + str(mis_val_table_ren_columns.shape[0]) +
                   " columns that have missing values.")

             # Return the dataframe with missing information
             return mis_val_table_ren_columns
```

```
[10] missing_values_table(play_store_df)

Your selected dataframe has 13 columns.
There are 5 columns that have missing values.
```

	Missing Values	% of Total Values
Rating	1474	13.60
Current Ver	8	0.07
Android Ver	3	0.03
Type	1	0.01
Content Rating	1	0.01

We can see that there are many NaN (missing) values in our dataset, especially in the Rating column.

There are two methods to deal with missing data:

- Dropping them
- Imputing them. Depending on the case we can allow a specific proportion of missing values, beyond which we might want to drop the variable from analysis. But this varies from case to case on the amount of information you think the variable has.

If the information contained in the variable is not that high, you can drop the variable if it has more than 50% missing values. There are projects / models where imputation of even 20 - 30% missing values provided better results. Age is missing in ~20% of cases, but you benefit by imputing them rather than ignoring the variable.

We can see that out of our 10000 rows, almost 1500 of the rows have null values in place of Ratings. Hence, dropped the Rating column.

Since the ratio of missing values for the rest of the columns is less (<0.05), we proceed to fill these with the mode values instead of dropping these rows.

```
## Drop the Null values in Rating column
play_store_df.dropna(subset =['Rating'], inplace = True)
print(f"The number of rows and columns in play_store_df before is {play_store_df_original.shape} and after dropping the NAs from 'Rating' is {play_store_df.shape}")
```

```
The number of rows and columns in play_store_df before is (10841, 13) and after dropping the NAs from 'Rating' is (9367, 13)
```

```
## Recalculate the missing value ratio
missing_values_table(play_store_df)
```

```
Your selected dataframe has 13 columns.
There are 3 columns that have missing values.
```

	Missing Values	% of Total Values
Current Ver	4	0.04
Android Ver	3	0.03
Content Rating	1	0.01

Since the ratio of missing values for the rest of the columns is less (<0.05), we proceed to fill these with the mode values instead of dropping these rows.

```
## Fill the three columns with the mode value
play_store_df['Current Ver'] = play_store_df['Current Ver'].fillna(play_store_df['Current Ver'].mode()[0])
play_store_df['Android Ver'] = play_store_df['Android Ver'].fillna(play_store_df['Android Ver'].mode()[0])
play_store_df['Content Rating'] = play_store_df['Content Rating'].fillna(play_store_df['Content Rating'].mode()[0])

missing_values_table(play_store_df)
```

```
Your selected dataframe has 13 columns.
There are 0 columns that have missing values.
  Missing Values  % of Total Values
```

3.2 Handling Duplicates:

Perform some checks and find that there are 2644 duplicated entries in 'Play Store Data' dataset from the "App" column. Hence, dropped the duplicates from the "App" column.

```
## Drop the duplicates from the "App" column
play_store_df.drop_duplicates(subset='App', inplace=True)
print(f"The number of rows and columns in play_store_df before is {play_store_df_original.shape} and after dropping the duplicates is {play_store_df.shape}")
```

```
The number of rows and columns in play_store_df before is (10841, 13) and after dropping the duplicates is (8197, 13)
```

3.3 Check for the Outliers:

On studying the dataset further, it was found that there was data with a weird anomaly. Let us find out the row in the data and purge it.

As we can see that this entry of our dataset is having a Rating of 19.0 which is way higher than the maximum rating of 5.0. Also, the value in the Reviews column has an alphabet which makes it alone entry to have so. Hence, we are removing this row to make our analysis easier.

```
[ ] play_store_df['Rating'].unique()

array([ 4.1,  3.9,  4.7,  4.5,  4.3,  4.4,  3.8,  4.2,  4.6,  3.2,  4. ,
        4.8,  4.9,  3.6,  3.7,  3.3,  3.4,  3.5,  3.1,  5. ,  2.6,  3. ,
        1.9,  2.5,  2.8,  2.7,  1. ,  2.9,  2.3,  2.2,  1.7,  2. ,  1.8,
        2.4,  1.6,  2.1,  1.4,  1.5,  1.2, 19. ])

[ ] play_store_df[play_store_df['Rating'] == 19.0]
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
10472	Life Made WI-Fi Touchscreen Photo Frame	1.9	19.0	3.0M	1,000+	Free	0	Everyone	Everyone	February 11, 2018	1.0.19	4.0 and up	4.1 and up

```
[ ] ## Drop the row that has incorrect values for our features
    play_store_df = play_store_df[~(play_store_df['Installs'] == "Free")].reset_index(drop=True)
```

3.4 Check and Convert the Following columns for EDA analysis:

As some of feature columns are non-numerical value so I have converted the categorical variables the into numerical for ease of analysis.

3.4.1 Installs

```
[ ] play_store_df['Installs'].unique()

array(['10,000+', '500,000+', '5,000,000+', '50,000,000+', '100,000+',
       '50,000+', '1,000,000+', '10,000,000+', '5,000+', '100,000,000+',
       '1,000,000,000+', '1,000+', '500,000,000+', '100+', '500+', '10+',
       '5+', '50+', '1+'], dtype=object)

[ ] ## Remove '+' and ',' from 'Installs' to make it numeric
    play_store_df['Installs'] = play_store_df['Installs'].apply(lambda x: x.replace('+', '') if '+' in str(x) else x)
    play_store_df['Installs'] = play_store_df['Installs'].apply(lambda x: x.replace(',', '') if ',' in str(x) else x)
    play_store_df['Installs'] = play_store_df['Installs'].apply(lambda x: int(x))

[ ] play_store_df['Installs'].unique()

array([     10000,     500000,    5000000,   50000000,     100000,
            50000,    1000000,   10000000,       5000,  100000000,
       1000000000,       1000,  500000000,        100,        500,
               10,          5,         50,          1])
```

3.4.2 Size

```
play_store_df['Size'].unique()

[ ] ## 'Size' column - convert Mbs to kbs
    play_store_df['Size'] = play_store_df['Size'].apply(lambda x: str(x).replace('Varies with device', 'NaN') if 'Varies with device' in str(x) else x)
    play_store_df['Size'] = play_store_df['Size'].apply(lambda x: str(x).replace('M', '') if 'M' in str(x) else x)
    play_store_df['Size'] = play_store_df['Size'].apply(lambda x: str(x).replace(',', '') if 'M' in str(x) else x)
    play_store_df['Size'] = play_store_df['Size'].apply(lambda x: float(str(x).replace('k', '')) / 1000 if 'k' in str(x) else x)
```

3.4.3 Price & Reviews

```
[ ] play_store_df['Price'].unique()

    array(['0', '$4.99', '$3.99', '$6.99', '$7.99', '$5.99', '$2.99', '$3.49',
           '$1.99', '$9.99', '$7.49', '$0.99', '$9.00', '$5.49', '$10.00',
           '$24.99', '$11.99', '$79.99', '$16.99', '$14.99', '$29.99',
           '$12.99', '$2.49', '$10.99', '$1.50', '$19.99', '$15.99', '$33.99',
           '$39.99', '$3.95', '$4.49', '$1.70', '$8.99', '$1.49', '$3.88',
           '$399.99', '$17.99', '$400.00', '$3.02', '$1.76', '$4.84', '$4.77',
           '$1.61', '$2.50', '$1.59', '$6.49', '$1.29', '$299.99', '$379.99',
           '$37.99', '$18.99', '$389.99', '$8.49', '$1.75', '$14.00', '$2.00',
           '$3.08', '$2.59', '$19.40', '$3.90', '$4.59', '$15.46', '$3.04',
           '$13.99', '$4.29', '$3.28', '$4.60', '$1.00', '$2.95', '$2.90',
           '$1.97', '$2.56', '$1.20'], dtype=object)
```

```
[ ] ## Remove "$" from "Price" columns to make it numeric
    play_store_df['Price'] = play_store_df['Price'].apply(lambda x: str(x).replace('$', '') if '$' in str(x) else str(x))
```

```
[ ] ## Convert the column types to numeric values
    play_store_df['Size'] = play_store_df['Size'].apply(lambda x: float(x))
    play_store_df['Installs'] = play_store_df['Installs'].apply(lambda x: float(x))
    play_store_df['Price'] = play_store_df['Price'].apply(lambda x: float(x))
    play_store_df['Reviews'] = play_store_df['Reviews'].apply(lambda x: int(x))
```

```
[ ] ## Display the dtypes of all the features in our dataset
    play_store_df.dtypes

    App                object
    Category           object
    Rating            float64
    Reviews             int64
    Size              float64
    Installs          float64
    Type               object
    Price             float64
    Content Rating     object
    Genres             object
    Last Updated       object
    Current Ver        object
    Android Ver        object
    dtype: object
```

4. Extracting statistics from the dataset

- Finding the Numerical and Categorical Columns
- Looking at the dataset, we think we can identify the categorical and continuous columns in it. But it might also be possible that the numerical values are represented as strings in some features. Or the categorical values in some features might be represented as some other datatypes instead of strings. Hence, it's good to check for the datatypes of all the feature

```
[89] # IDENTIFYING NUMERICAL FEATURES

numeric_data = play_store_df.select_dtypes(include=np.number) # select_dtypes selects data with numeric features
numeric_col = numeric_data.columns

# we will store the numeric features in a variable

print("Numeric Features:", numeric_data.shape)
numeric_data.describe(include='all').T
```

```
Numeric Features: (8196, 5)
```

	count	mean	std	min	25%	50%	75%	max
Rating	8196.0	4.173243e+00	5.366247e-01	1.0000	4.0	4.3	4.5	5.000000e+00
Reviews	8196.0	2.552515e+05	1.985594e+06	1.0000	126.0	3004.0	43813.0	7.815831e+07
Size	7027.0	2.175443e+01	2.272650e+01	0.0085	4.9	13.0	31.0	1.000000e+02
Installs	8196.0	9.165090e+06	5.825087e+07	1.0000	10000.0	100000.0	1000000.0	1.000000e+09
Price	8196.0	1.037884e+00	1.685788e+01	0.0000	0.0	0.0	0.0	4.000000e+02

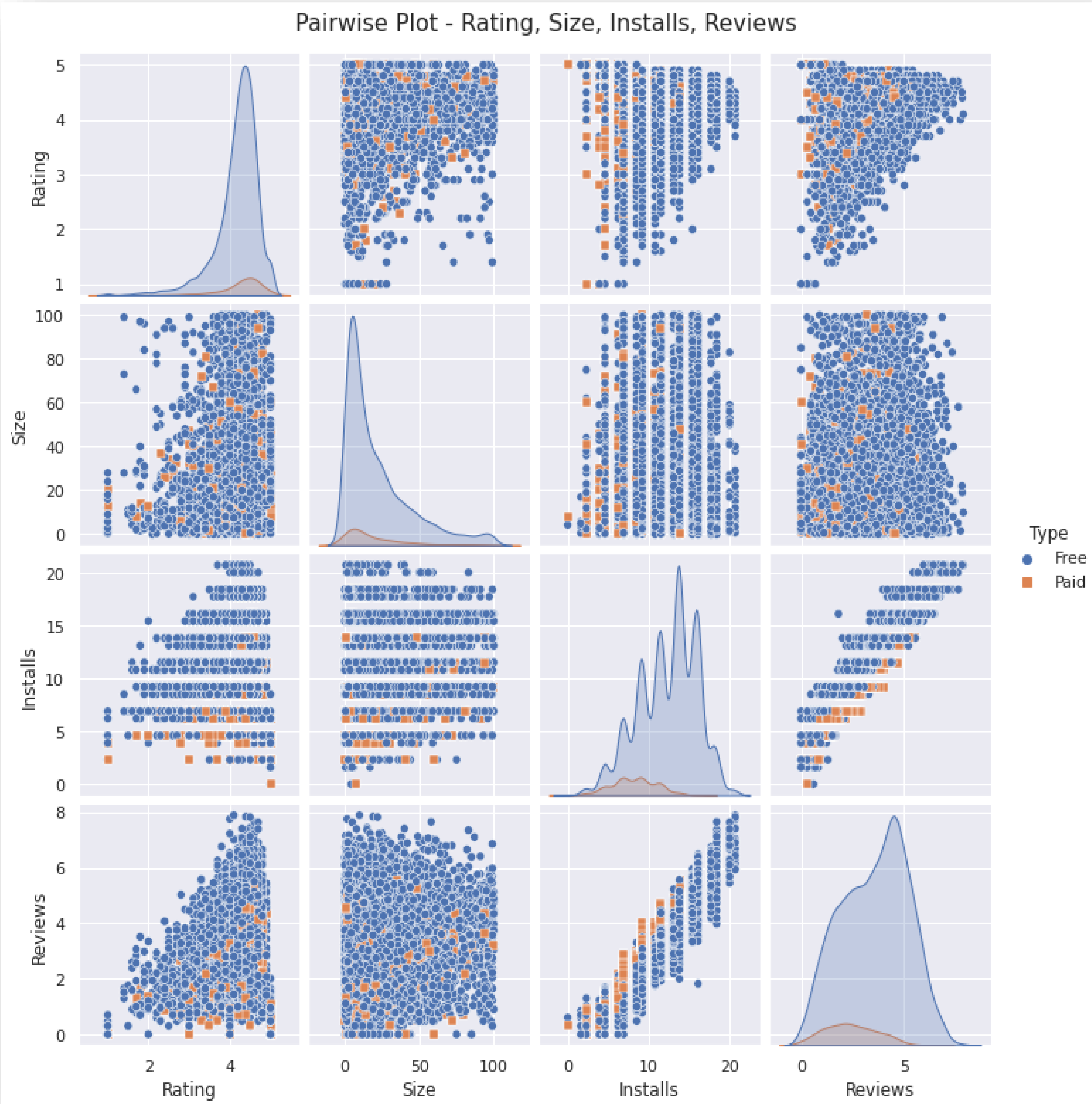
```
[88] # IDENTIFYING CATEGORICAL FEATURES

cat_data = play_store_df.select_dtypes(exclude=np.number) # select_dtypes selects data with non-numeric features
cat_col = cat_data.columns

# we will store the non-numeric features in a variable

print("Non-Numeric Features:", cat_data.shape)
cat_data.describe(include='all').T
```

```
Non-Numeric Features: (8196, 8)
```

	count	unique	top	freq
App	8196	8196	Wi-Fi Auto-connect	1
Category	8196	33	FAMILY	1608
Type	8196	2	Free	7592
Content Rating	8196	6	Everyone	6618
Genres	8196	114	Tools	717
Last Updated	8196	1300	August 3, 2018	245
Current Ver	8196	2624	Varies with device	1015
Android Ver	8196	31	4.1 and up	1813

5. Exploratory analysis and visualizations

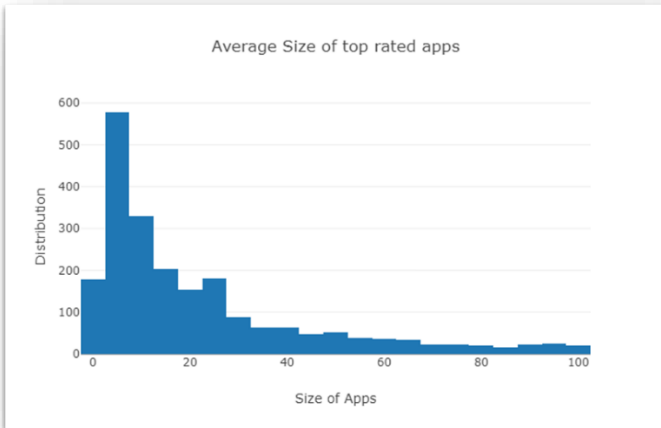
After establishing a good sense of each feature, I have proceeded with plotting a pairwise plot between all the quantitative variables to look for any evident patterns or relationships between the features.

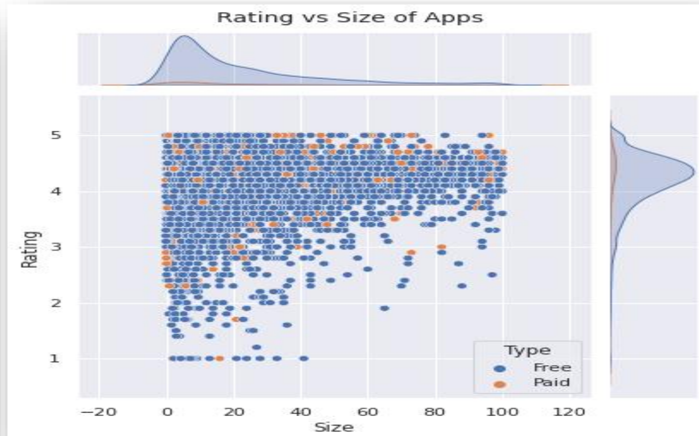


5.1 Size of the app affect the ratings and number of installs

After the examined the app size, review, and rating. For size, if the mobile app is too large, it may be difficult and/or expensive for users to download. Lengthy download times could turn users off before they even experience your mobile app. Plus, each user's device has a finite amount of disk space. For price, some users expect their apps to be free or inexpensive.

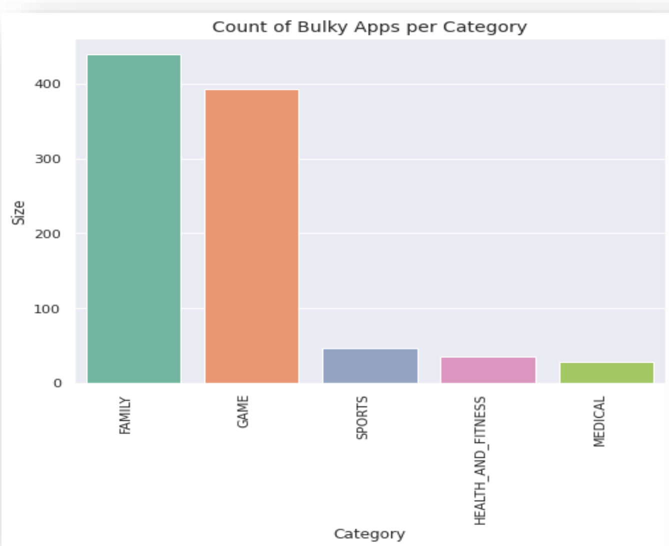
These problems compound if the developing world is part of your target market; especially due to internet speeds, earning power and exchange rates.

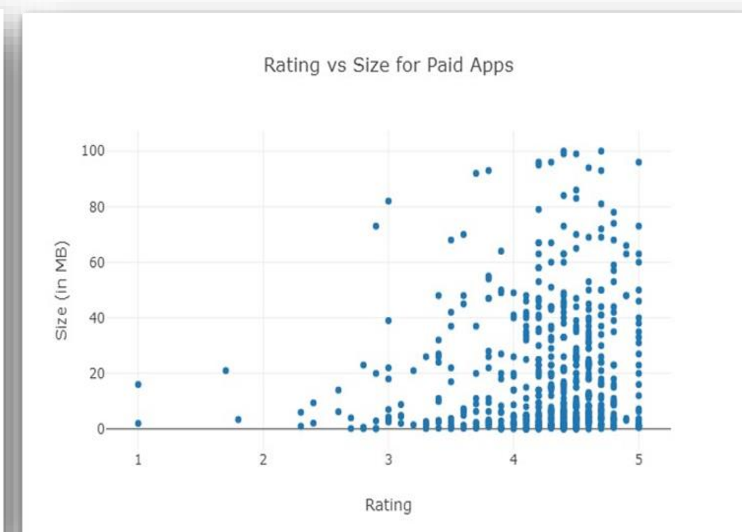




5.2. Category those are in higher in size apps and how are they rated for paid app

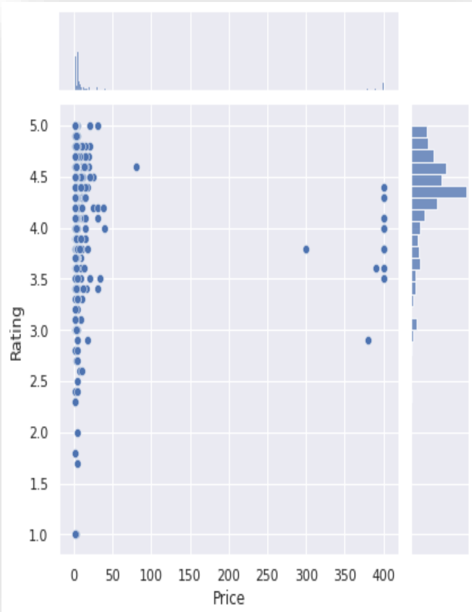
Analyzed the sizing distribution of the top-rated apps (rating greater than 4.5) and observed that most top-rated apps are optimally sized between ~2MB to ~40MB i.e., neither too light nor too higher in size. Found that the Game and Family categories have the highest number of higher in size apps. Also observed is that despite this, these Higher in size apps are highly rated indicating that they are higher in size for a purpose. Most of the paid apps that are highly rated have small sizes which imply that most paid apps are designed and developed to cater to specific functionalities and hence are not higher in size. Users prefer to pay for apps that are light-weighted. A paid app that is higher in size may not perform well in the market, but it also depends on the category of the app.

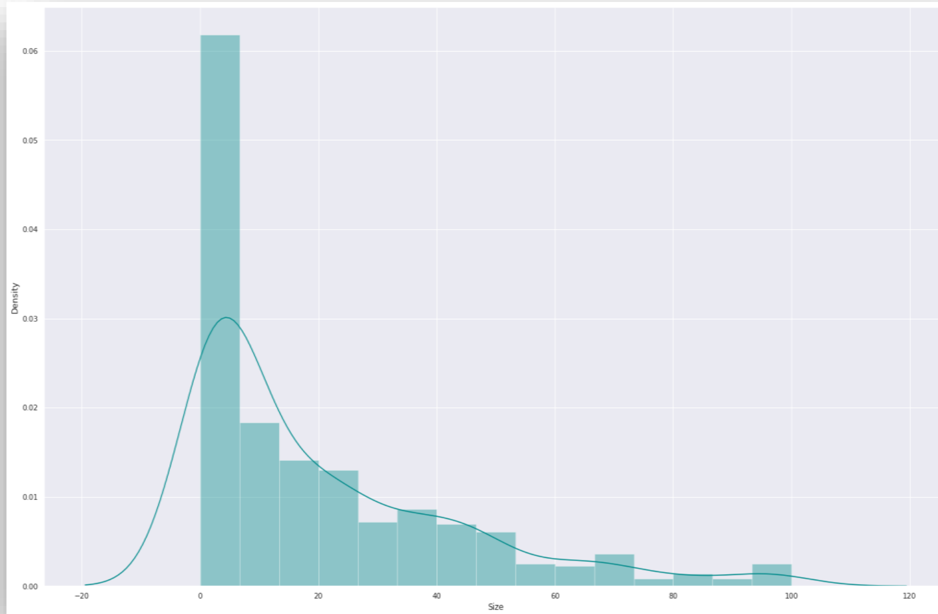




5.3. The App prices affect rating and number of installs

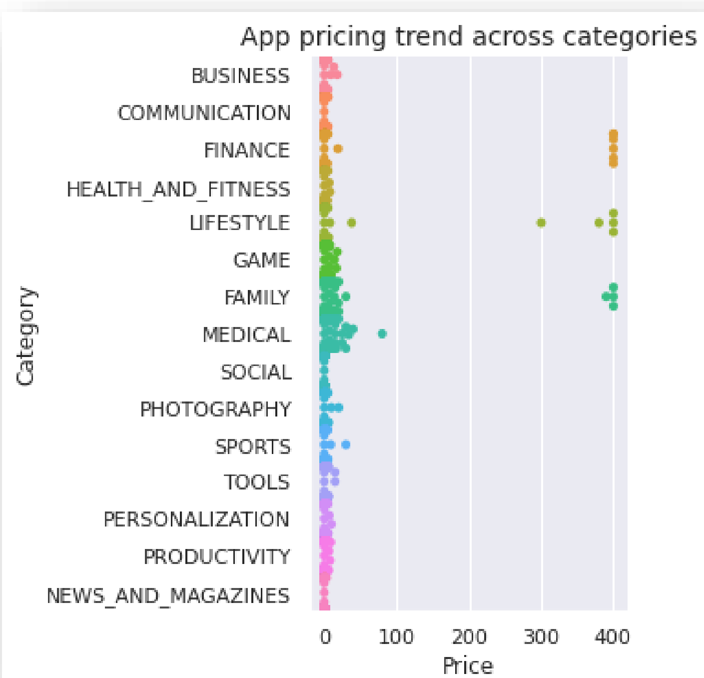
Most top-rated apps are optimally priced between ~1 to 30. There are only a very few apps priced above 20$.





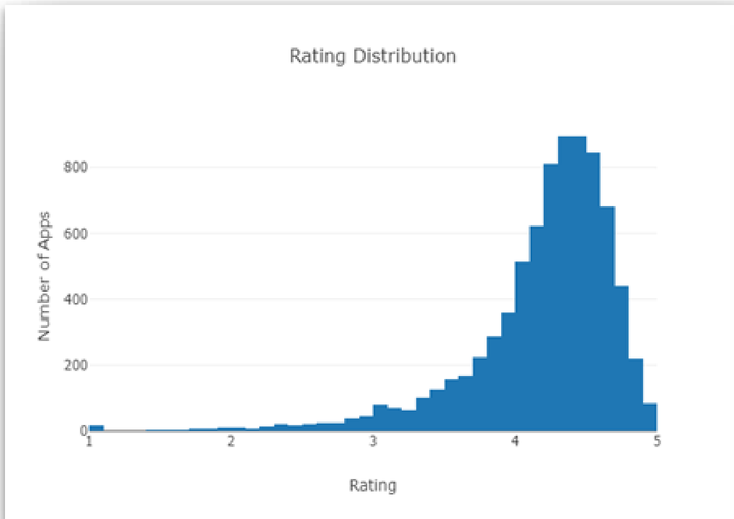
5.4. Price trend across category

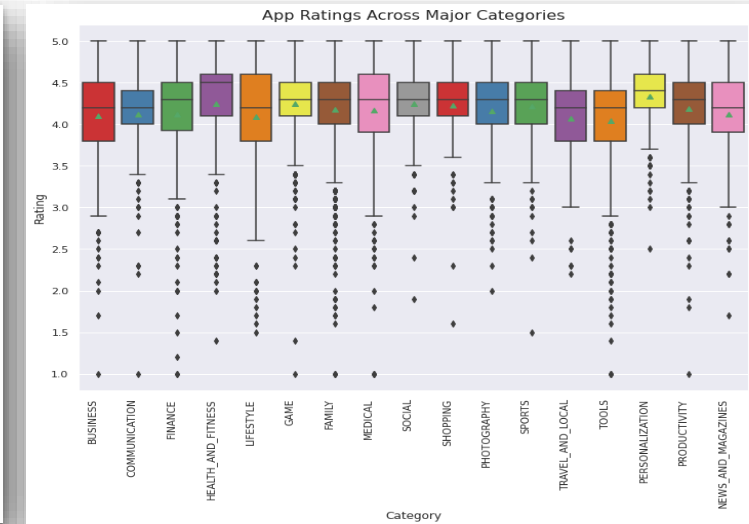
The initial intuition for a pricing strategy for apps to be popular would be to make them free. That is confirmed by the data as free apps dominate the store at 92.6%, making it over 12.5 times their paid counterparts. It follows that cheaper apps are also more popular which is supported by the fact that 90% of paid apps are less than $10 and 80% of apps are less than $5. Categorically, the most expensive apps are Family, Lifestyle and finance apps which would suggest that people are willing to pay more for medical apps, possibly due to a belief in reliability or highly specialized applications.

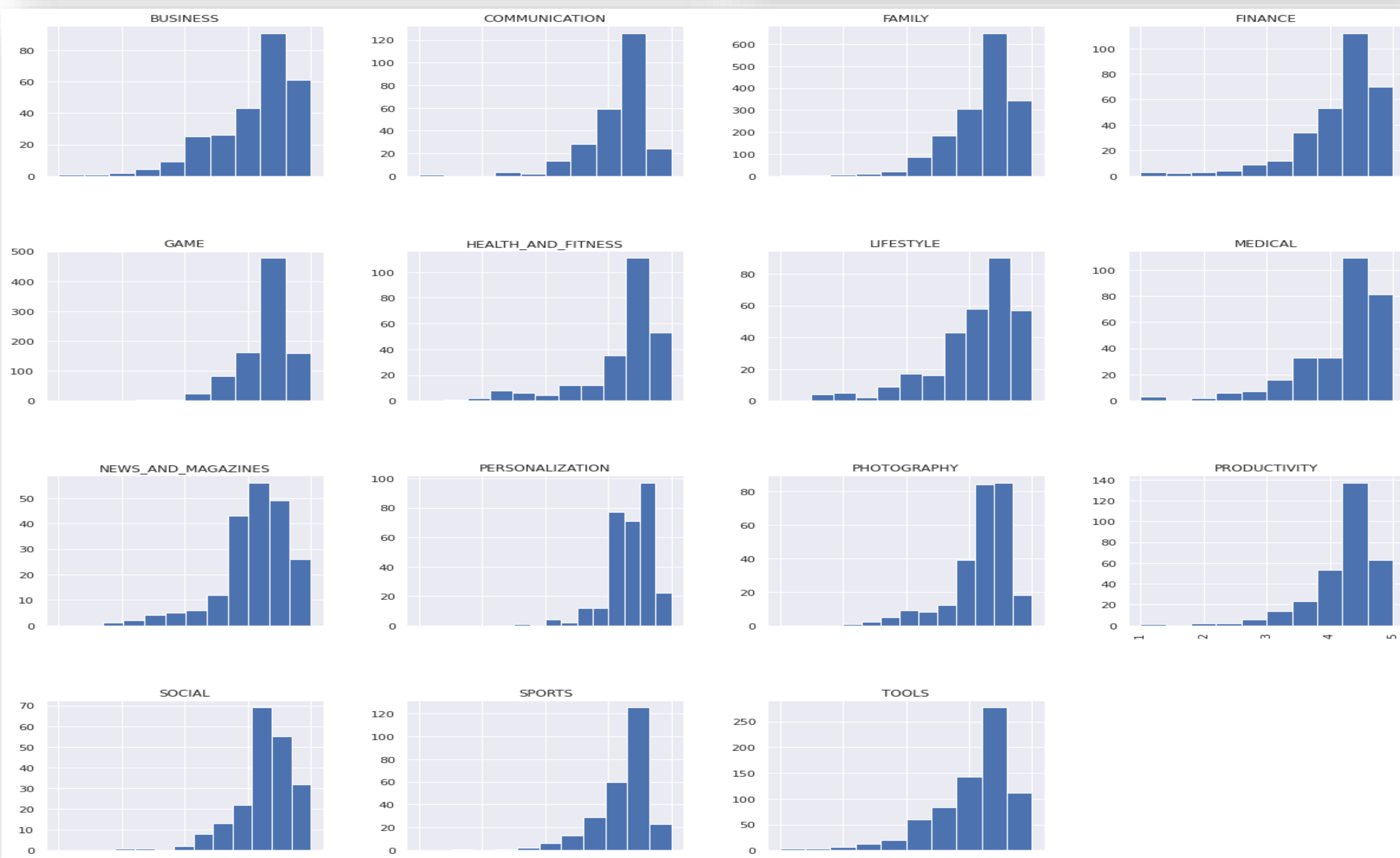


5.4. Rating Distribution

The rating distribution revealed that most apps perform reasonably well with an average rating of 4.17. We broke down the average rating by category to check if any category performs exceedingly well or badly. We conducted a One-way 'Anova Test' and confirmed that the average rating across categories is statistically different. The Health and Fitness and Books and Reference produce the best apps with 50% of apps having a rating greater than 4.5. Interestingly, half of the Dating apps have a rating lower than the average.



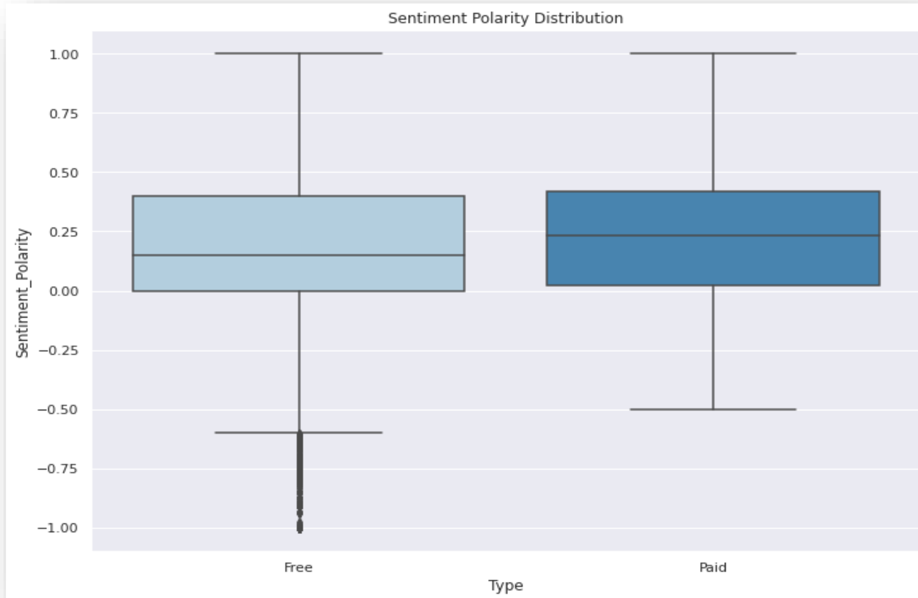


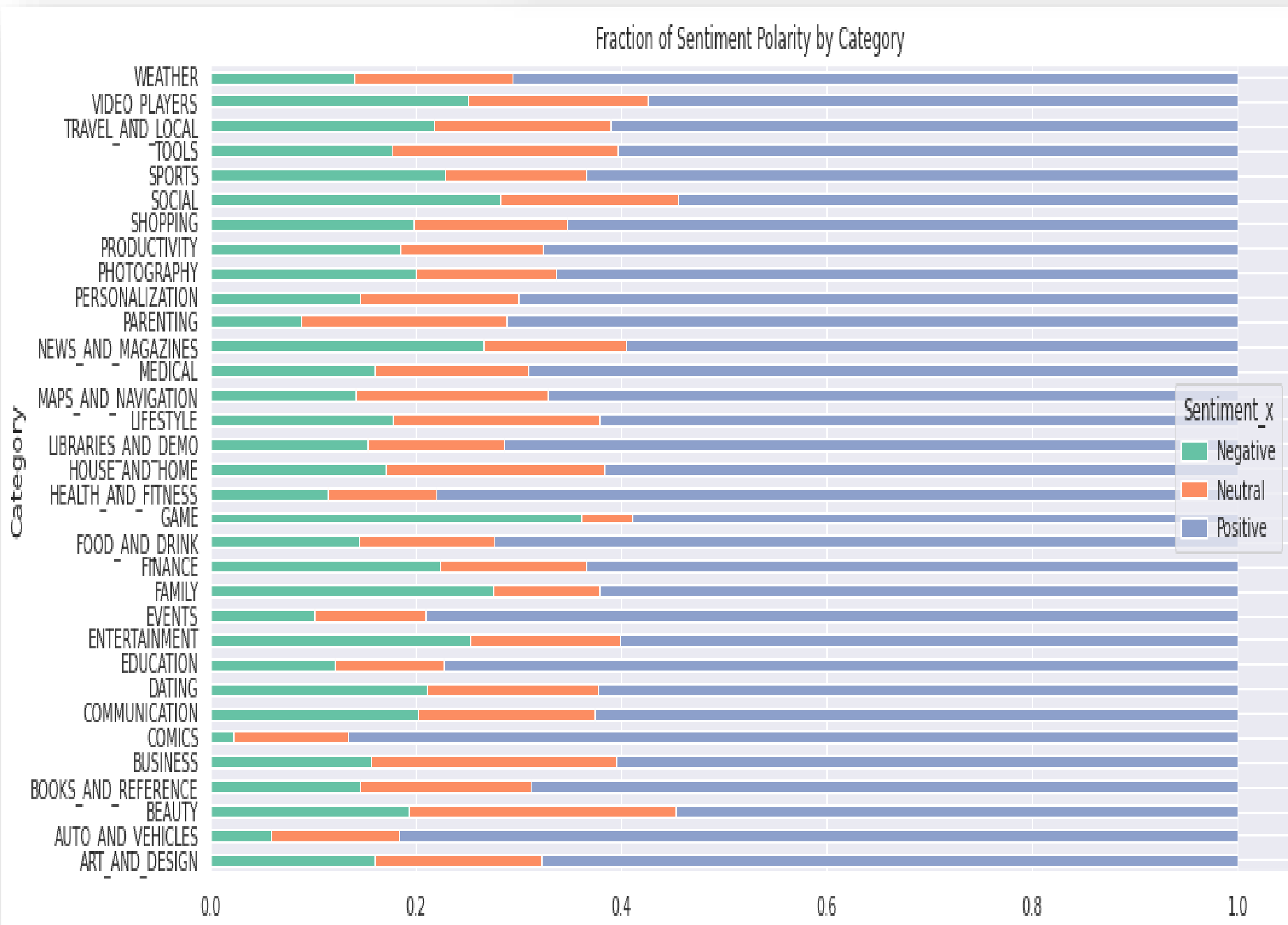


5.5. Basic Sentiment Analysis – User Reviews

We plotted the fraction of positive, negative, and neutral reviews for each category and observed that the Health and Fitness apps perform the best with more than 85% positive reviews.

On the other hand, Game and Social apps have a higher fraction of negative reviews. We compared the reviews between free and paid apps and found that people are harsher towards free apps whereas users are more tolerant when they are paying for them .





6. Conclusion

- The average rating of (active) apps on Google Play Store is 4.17.
- Users prefer to pay for apps that are light-weighted. Thus, a paid app that is higher in size may not perform well in the market.
- Most of the top-rated apps are optimally sized between ~2MB to ~40MB - neither too light nor too heavy.
- Most of the top-rated apps are optimally priced between ~1$ to ~30$ - neither too cheap nor too expensive.
- Medical and Family apps are the most expensive and even extend up to 80$.
- Users tend to download a given app more if it has been reviewed by many people.
- Health and Fitness apps receive more than 85% positive reviews.
- Game and Social apps receive mixed feedback 50% positive and 50% negative.