

# Exploring the Android App Market - a Data Mining Analysis of Google Playstore Applications



## SENG 474 Fall 2018 Data Mining Project

Cole Boothman St.ID V00808231

Amy Hanvoravongchai St.ID V00822271

### Contents

1. Introduction
2. Data Collection
3. Data Preprocessing and Visualization
4. Data Mining/Analysis of Reviews with Naive Bayes
5. Conclusions
6. References

### README:

Our project notebook uses multiple python modules (sklearn, seaborn, etc.) for data processing and graphing. We have included an HTML file and a .pdf of the finished version of our project, called `Finished_Project.html` and `Finished_Project.pdf`. However, if you wish to run the notebook yourself, we have also included a pip requirements file called `requirements.txt` that include all the necessary modules needed to run the notebook. Please install these if you wish to run the code yourself.

We suggest viewing `Finished_Project.html` for ease of viewing, since the PDF file cuts some of the text and displays off between pages.

### If you wish to run the notebook yourself:

To install the required modules, in the /474project/ folder directory run `pip install -r requirements.txt`

Additionally, you'll need to download some packages for the NLTK python module. This is the natural language toolkit that we are pulling stopwords from. To do this, open up a python console and run:

```
>>> import nltk
>>> nltk.download()
showing info http://nltk.github.com/nltk_data/
```

Click download from the GUI.

After everything is downloaded for NLTK and you've installed all the additional packages from the requirements.txt file, you should be good to run the notebook.

## 1. Introduction

Within the past 10 years, the market for mobile applications has grown tremendously. As the technology for smart phones and mobile applications has increased, so has the capabilities and complexity of our mobile applications. Consumers are now able to

browse and download millions of applications available onto their phones from the Android and iPhone app stores, choosing from a large variety of applications spanning across social media, video gaming, utility, health technology and many more categories of mobile apps.

The rapid rise of the mobile application market has increased the demand for software development to be more mobile oriented, from developing new mobile applications to making new web applications and websites user friendly for mobile users. This shift in application development accounts for the fact that mobile users surpassed the amount of desktop users in 2014 [1], and that in 2017 the estimated global application revenue for mobile applications was expected to be around 77 billion dollars [2].

Given the size and growth of the mobile market, companies who exist or wish to invest in the mobile application market may wish to conduct a thorough economic analysis of existing mobile applications. We can consider questions such as 'What makes an mobile application successful?', 'What do people have to say about mobile applications that are highly rated?' or 'What categories of mobile applications have the most downloads?' as queries that might help us investigate how mobile applications become well known, acquire high ratings and profitable.

In this project, we analyze two datasets containing information on applications in the Android Google Play Store. The first dataset is the metadata of 10842 applications in the Google Play Store, containing the names, categories, downloads, number of reviews, and other data on each application. We conduct an analysis of the correlation between this metadata and popular applications in the Google Play Store, as well as the correlation for unpopular applications. The second dataset is a set of 64292 app reviews, in which we analyze commonly used words in positive and negative reviews, as well as present a model that can be used for predicting sentiment in app reviews.

## 2. Data Collection

For this project, we used the following dataset from Kaggle.com:

<https://www.kaggle.com/lava18/google-play-store-apps>

This dataset consists of two csv files, containing details and reviews of applications on the Google Play Store.

File 1: `googleplaystore.csv`

This csv file contains metadata on 10842 applications in the Google Play Store. For each application, this includes:

- App Name
- Category
- Number of Reviews
- Size
- Number of Installs
- Type (Free, Paid)
- Price
- Content Rating
- Genres
- Last Updated
- Current Version
- Android Versions supported

As an example, here is the first row in our file:

App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
Photo Editor	ART_AND_D	4.1	159	19M	10,000+	Free		0 Everyone	Art & Design	07-Jan-18	1.0.0	4.0.3 and up

We will use this dataset to examine the correlations between successful apps and their various metadata, as well as what makes an application *not* successful.

File 2: `googleplaystore_user_reviews.csv`

This csv file contains review details on 64292 apps on the Play Store. For each application, this includes:

- App Name
- Translated Review
- Sentiment
- Sentiment Polarity
- Sentiment Subjectivity

As an example, here is the first row in our file:

App	Translated_R	Sentiment	Sentiment_P	Sentiment_Subjectivity
10 Best Food	I like eat deli	Positive	1	0.53333333

We will use this dataset to examine positive and negative reviews, and examine a model that makes predictions for new reviews based on this dataset.

### 3. Data Preprocessing and Visualization

Let's examine the data and do any preprocessing and cleanup that might be needed.

In [28]:

```
import pandas as pd
import numpy as np

data = pd.read_csv('googleplaystore.csv')

print('The total number of applications in this dataset is: {}'.format(data.shape[0]))
data.head()
```

The total number of applications in this dataset is: 10841

Out[28]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Version
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	4.1 and above
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	4.1 and above
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	4.1 and above
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 above
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 above

There's a couple things that we can improve on our dataset for processing.

- Some columns have NaN as values. For example, Categories and Ratings contain this. Let's filter these results.
- Some of the install values have 'Free' and 'Paid' as it's column value, we'll remove these.
- The number of installs include the "+" character. We will remove these so we can use the number of installs as an float.
- The size category has the suffix 'M' for megabytes, and 'k' for some apps that are small. ie. 600kb. We'll remove the k and M letters and divide app sizes in kb by 1000 to convert to Megabytes.
- Remove dollar sign character from prices
- Convert values to string or float values where necessary.

In [29]:

```
# DATA CLEANING

# Drop any duplicate Apps
data.drop_duplicates(subset='App', inplace=True)
data = data[data['Android Ver'] != np.nan]
data = data[data['Android Ver'] != 'NaN']

# Some values in the 'Installs' column are 'Free'/'Paid'
# This is an error -> remove these columns.
data = data[data['Installs'] != 'Free']
```

```

data = data[data['Installs'] != 'Paid']

# Remove the '+' and '-' Characters from the installs column.
data['Installs'] = data['Installs'].str.replace(',', '')
data['Installs'] = data['Installs'].str.replace('+', '')
data['Installs'] = data['Installs'].astype(int)

# Remove the 'M' for megabytes in size.
# There are also some apps with kb for size. So we will divide by 1000 to convert to MB.
data['Size'] = data['Size'].apply(lambda x: str(x).replace('Varies with device', 'NaN') if 'Varies with device' in str(x) else x)
data['Size'] = data['Size'].str.replace('M', '')
data['Size'] = data['Size'].str.replace(',', '')
data['Size'] = data['Size'].apply(lambda x: float(str(x).replace('k', '')) / 1000 if 'k' in str(x) else x)

# Remove $ from Price
data['Price'] = data['Price'].str.replace('$', '')

# Change values to float/int
data['Installs'] = data['Installs'].astype(float)
data['Size'] = data['Size'].astype(float)
data['Price'] = data['Price'].astype(float)
data['Reviews'] = data['Reviews'].astype(int)

```

## Visualization and Data Analytics

We are going to do some visualization and exploration of our metadata set. We'll graph and take a look at ratings, as well as compare price and size of applications, and analyze the categories and attribute distributions.

Lastly we'll graph some pairplots using Python's seaborn library to visualize and compare some of the datasets's attributes, and see if we can infer any patterns that seem to appear in the data.

The attributes we are using for our pairplots are:

- Rating
- Number of Reviews
- Number of Installs
- Size
- Price

Each value will be plotted with the type of application (Free and Paid)

## Category Distribution

In [30]:

```

# Display the plots inline with the notebook
%matplotlib inline

import seaborn as sns
import numpy as np
import matplotlib.ticker as ticker

from matplotlib import pyplot as plt

```

Here are the individual counts for each App Category:

In [31]:

```
data['Category'].dropna().value_counts()
```

Out[31]:

FAMILY	1832
GAME	959
TOOLS	827
BUSINESS	420
MEDICAL	395
PERSONALIZATION	376
PRODUCTIVITY	374

LIFESTYLE	369
FINANCE	345
SPORTS	325
COMMUNICATION	315
HEALTH_AND_FITNESS	288
PHOTOGRAPHY	281
NEWS_AND_MAGAZINES	254
SOCIAL	239
BOOKS_AND_REFERENCE	222
TRAVEL_AND_LOCAL	219
SHOPPING	202
DATING	171
VIDEO_PLAYERS	163
MAPS_AND_NAVIGATION	131
EDUCATION	119
FOOD_AND_DRINK	112
ENTERTAINMENT	102
AUTO_AND_VEHICLES	85
LIBRARIES_AND_DEMO	84
WEATHER	79
HOUSE_AND_HOME	74
EVENTS	64
ART_AND_DESIGN	64
PARENTING	60
COMICS	56
BEAUTY	53

Name: Category, dtype: int64

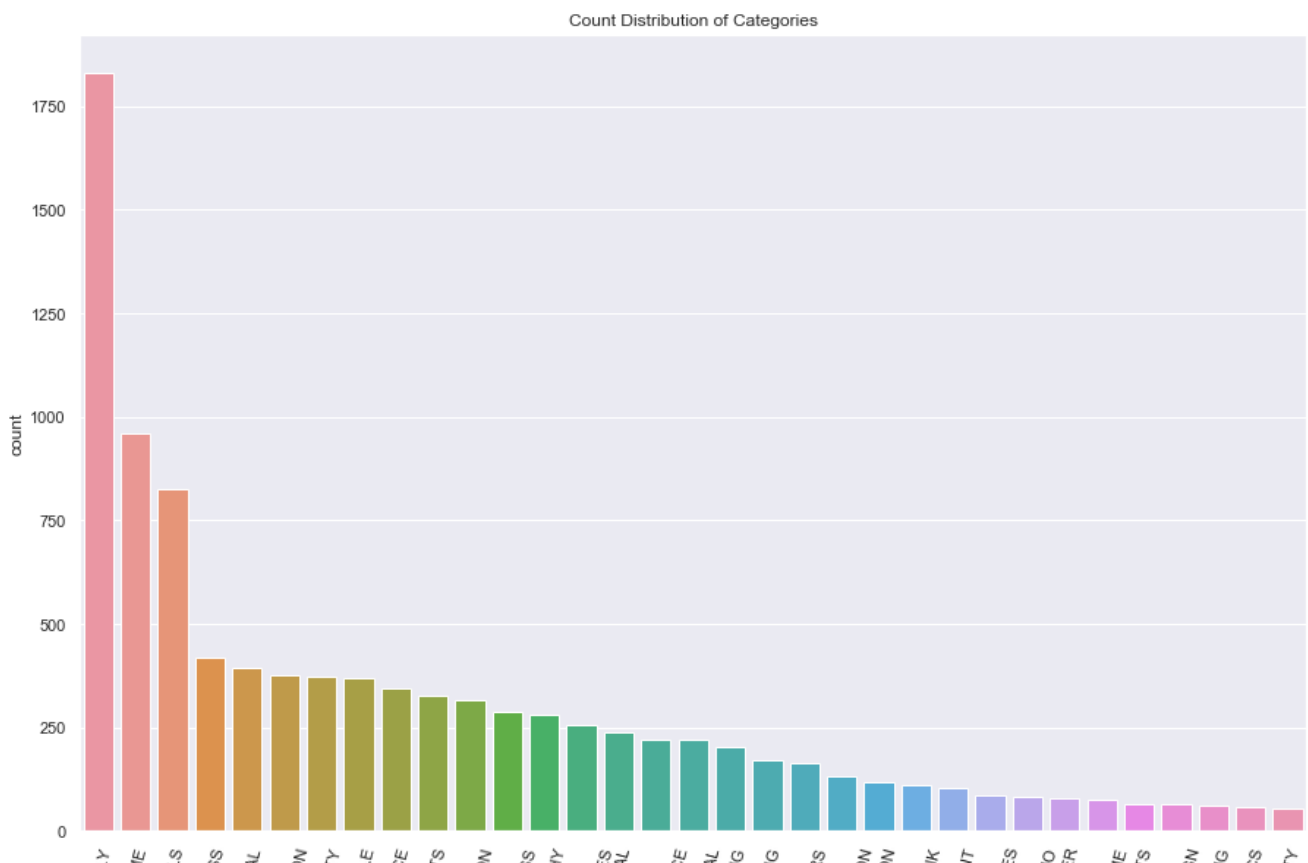
Below is a graph of the category distribution:

In [32]:

```
sns.set(rc={'figure.figsize':(15,10)})

categories = data['Category'].dropna()
cat_count = sns.countplot(x = 'Category', data=data, order = data['Category'].value_counts().index)
cat_count.set_title('Count Distribution of Categories')
x_labels = data['Category'].value_counts().keys()

# Need to iterate since get_xticklabels() prints out all the values
for item in cat_count.get_xticklabels():
    item.set_rotation(75)
```





We can see that the App Categories Family, Game and Tools hold a large share of the Play Store Market apps. Interestingly Business and Medical follow up in fourth and fifth place for the number of applications in their respective category. Perhaps the relatively big share of business and medical applications points to a trend in more enterprise applications being ported to mobile? Note that Medical is a separate category from Lifestyle, and Health and Fitness as well.

Categories have genres as well. What does the genre distribution look like?

In [33]:

```
data['Genres'].value_counts()
```

Out[33]:

Tools	826
Entertainment	561
Education	510
Business	420
Medical	395
Personalization	376
Productivity	374
Lifestyle	368
Finance	345
Sports	331
Communication	315
Action	299
Health & Fitness	288
Photography	281
News & Magazines	254
Social	239
Books & Reference	222
Travel & Local	218
Shopping	202
Simulation	193
Arcade	184
Dating	171
Casual	165
Video Players & Editors	162
Maps & Navigation	131
Puzzle	119
Food & Drink	112
Role Playing	105
Strategy	95
Racing	91
...	
Entertainment;Pretend Play	2
Puzzle;Creativity	2
Video Players & Editors;Music & Video	2
Health & Fitness;Action & Adventure	1
Health & Fitness;Education	1
Entertainment;Education	1
Lifestyle;Education	1
Puzzle;Education	1
Communication;Creativity	1
Art & Design;Pretend Play	1
Adventure;Education	1
Lifestyle;Pretend Play	1
Strategy;Creativity	1
Adventure;Brain Games	1
Trivia;Education	1
Art & Design;Action & Adventure	1
Video Players & Editors;Creativity	1
Travel & Local;Action & Adventure	1
Casual;Music & Video	1
Strategy;Education	1
Tools;Education	1
Parenting;Brain Games	1

```
Comics;Creativity 1
Racing;Pretend Play 1
Board;Pretend Play 1
Music & Audio;Music & Video 1
Role Playing;Brain Games 1
Role Playing;Education 1
Books & Reference;Creativity 1
Arcade;Pretend Play 1
Name: Genres, Length: 118, dtype: int64
```

Upon immediate inspection, it looks like some of the 'Genres' have the same label as some categories. In addition, there are a lot of genres in which only one or a few apps fall into that genre.

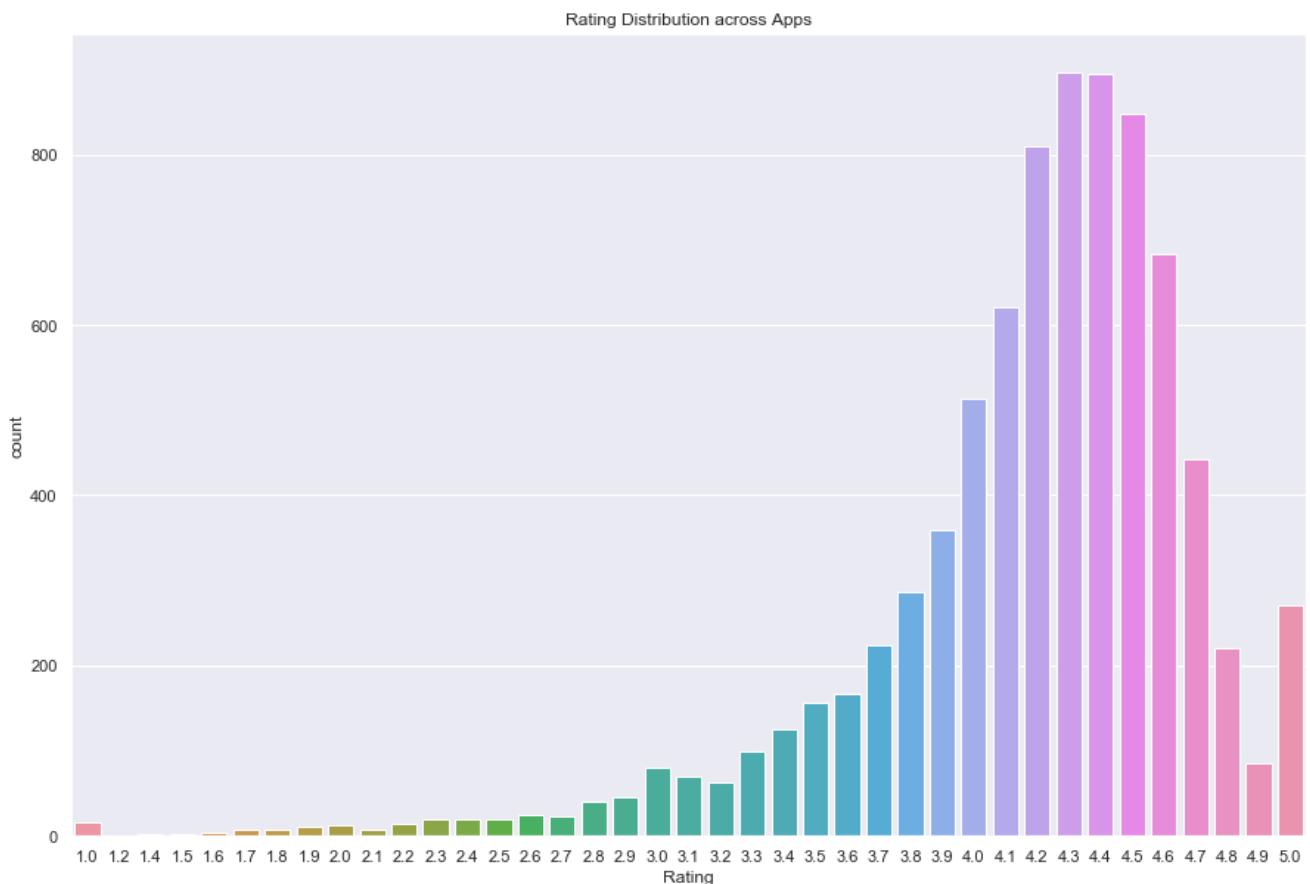
Now let's take a look at the rating distribution for all applications in our dataset.

In [34]:

```
app_rating_dist = sns.countplot(x = 'Rating', data=data)
app_rating_dist.set_title('Rating Distribution across Apps')
print('\nAverage Rating across all applications: {}'.format(data['Rating'].mean()))
print('\nMedian Rating across all applications: {}'.format(data['Rating'].median()))
```

Average Rating across all applications: 4.173243045387994

Median Rating across all applications: 4.3



Somewhat surprisingly, we see that a lot of applications have a very high average rating in the app store. (When the Google Play Store shows a rating for an application, it is the average of all ratings given for that app) A large majority of applications have ratings greater than 4 stars - From above, we can see that on average, an application has a rating of about 4.1, and the median rating was 4.3.

Let's cross check these ratings against the top 10 categories, to see the rating distribution for popular categories.

In [35]:

```
sns.set(rc={'figure.figsize':(18,18)})
top_10 = data['Category'].value_counts().keys()
```

```

i=0
j=1
for k in range(0,5):
    fig = plt.figure()
    # Two figures per row
    ax1 = fig.add_subplot(2,2,1)
    ax2 = fig.add_subplot(2,2,2)
    ax1.set_title('Ratings for Category: {}'.format(top_10[i]))
    ax2.set_title('Ratings for Category: {}'.format(top_10[j]))

    cat1= data[data.Category == top_10[i]]
    cat2= data[data.Category == top_10[j]]
    sns.countplot(y = 'Rating', ax=ax1, data=cat1, palette=sns.color_palette("PuBuGn_d"))
    sns.countplot(y = 'Rating', ax=ax2, data=cat2, palette=sns.color_palette("PuBuGn_d"))
    i = i+2
    j = i+1

for m in range(0,10):
    cat = data[data.Category == top_10[m]]['Rating']
    print(' Average Rating for Category: {} is {}'.format(top_10[m], cat.mean()))
    print(' Median Rating for Category: {} is {}\n'.format(top_10[m], cat.median()))

```

Average Rating for Category: FAMILY is 4.179664179104478  
Median Rating for Category: FAMILY is 4.3

Average Rating for Category: GAME is 4.247368421052632  
Median Rating for Category: GAME is 4.3

Average Rating for Category: TOOLS is 4.039554317548746  
Median Rating for Category: TOOLS is 4.2

Average Rating for Category: BUSINESS is 4.098479087452472  
Median Rating for Category: BUSINESS is 4.2

Average Rating for Category: MEDICAL is 4.166551724137932  
Median Rating for Category: MEDICAL is 4.3

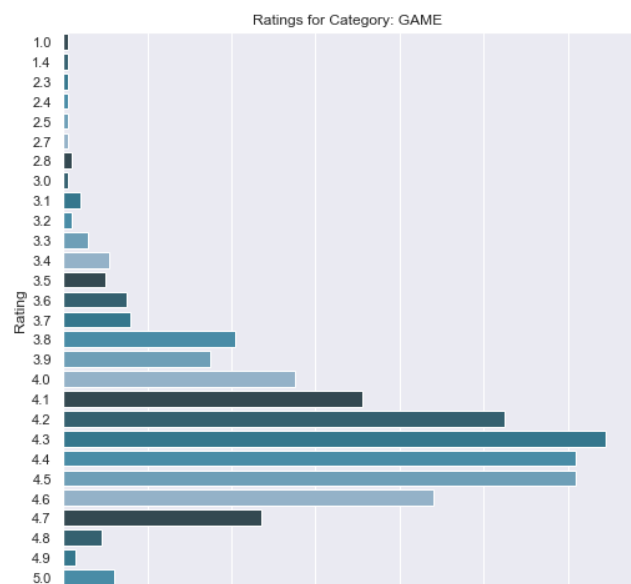
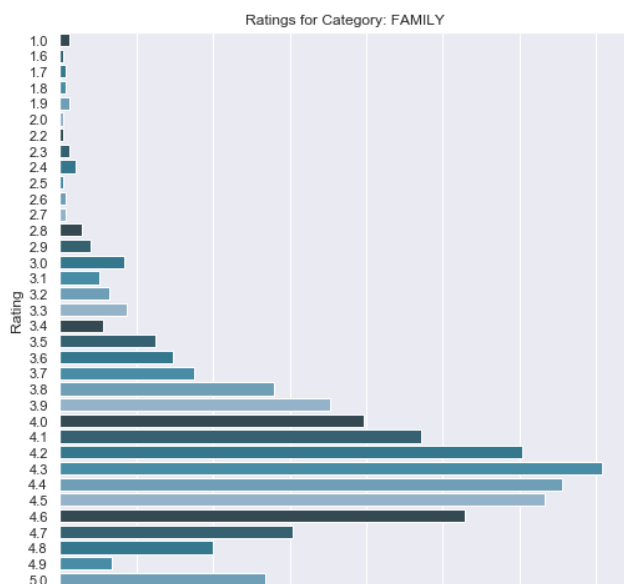
Average Rating for Category: PERSONALIZATION is 4.332214765100671  
Median Rating for Category: PERSONALIZATION is 4.4

Average Rating for Category: PRODUCTIVITY is 4.183388704318936  
Median Rating for Category: PRODUCTIVITY is 4.3

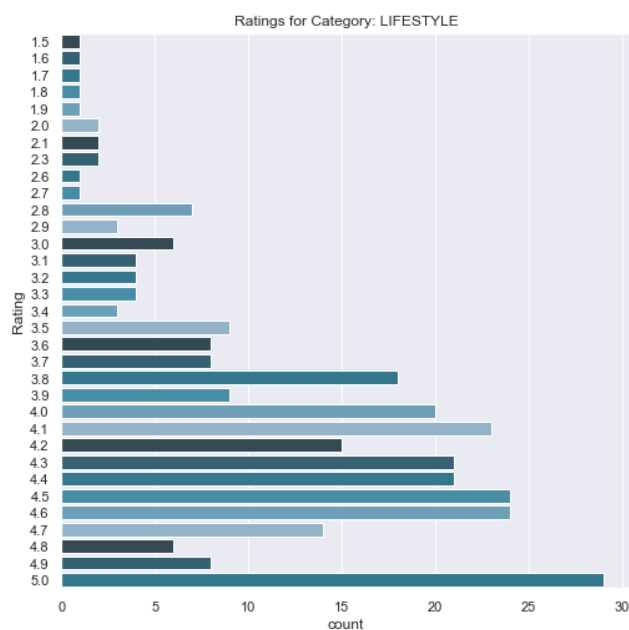
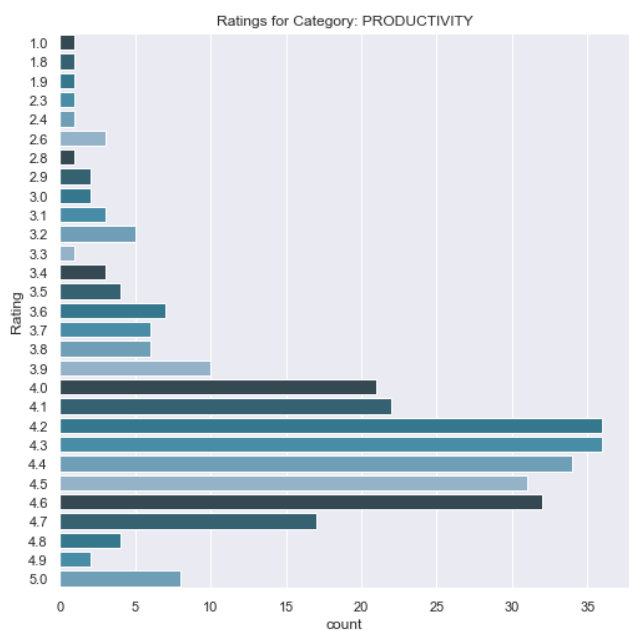
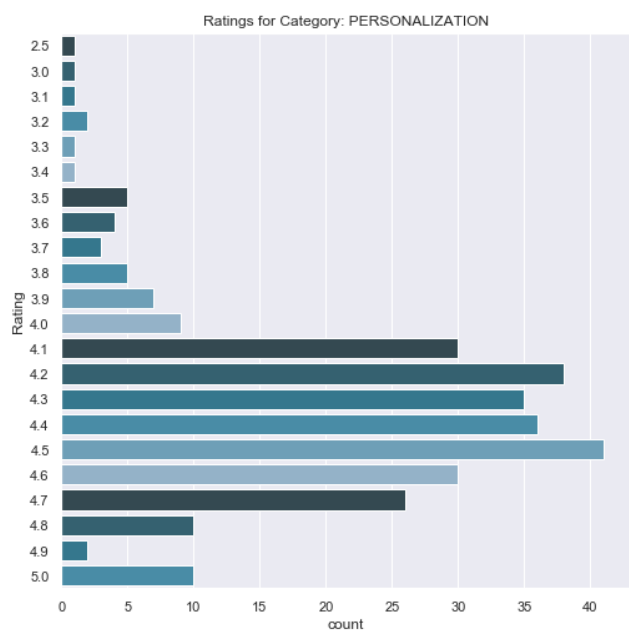
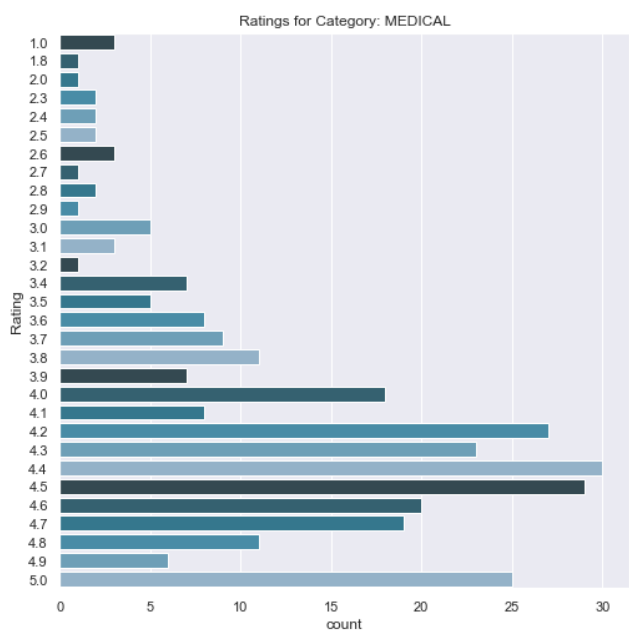
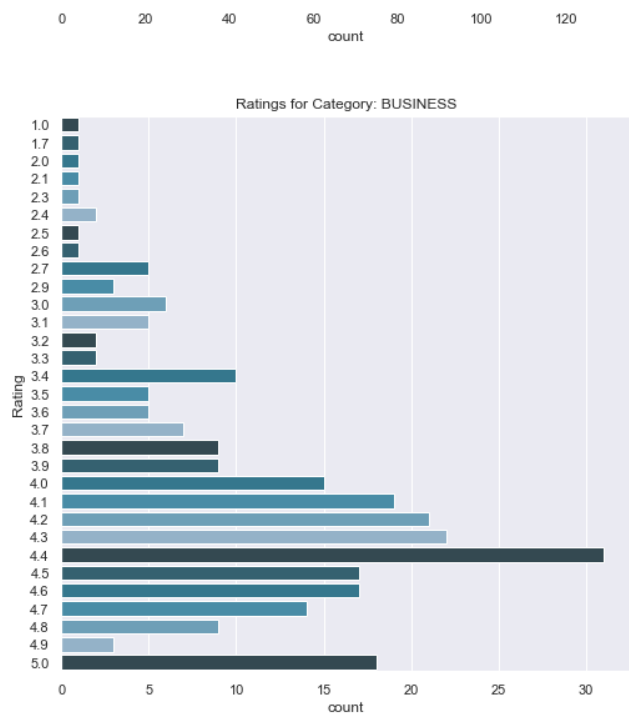
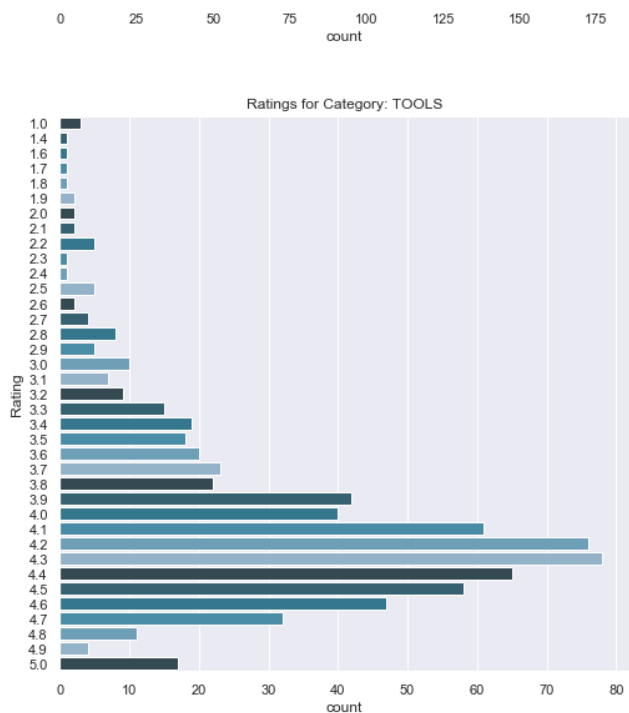
Average Rating for Category: LIFESTYLE is 4.093355481727575  
Median Rating for Category: LIFESTYLE is 4.2

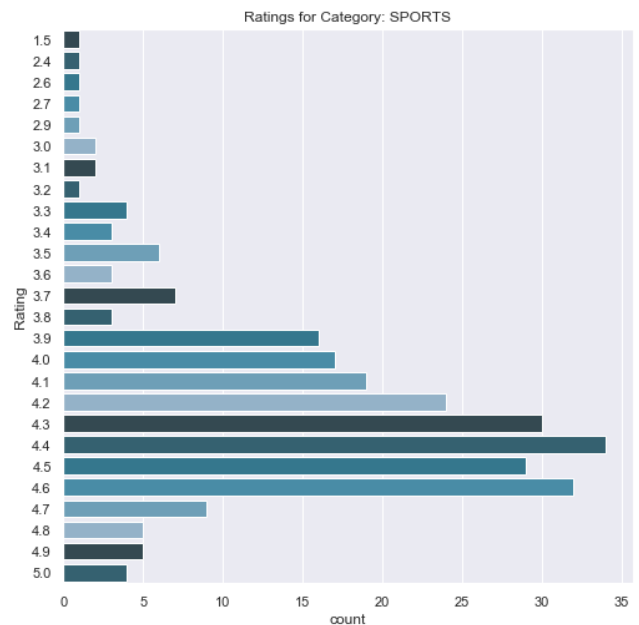
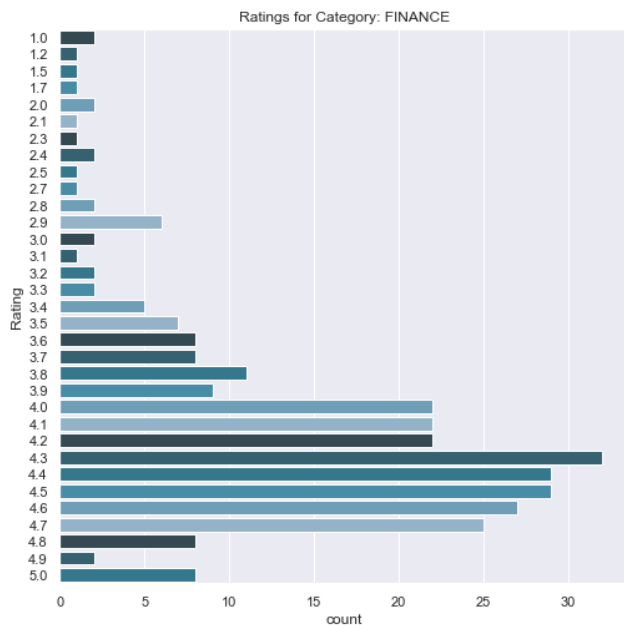
Average Rating for Category: FINANCE is 4.115562913907285  
Median Rating for Category: FINANCE is 4.3

Average Rating for Category: SPORTS is 4.216153846153846  
Median Rating for Category: SPORTS is 4.3









We can see that the average rating for each category is pretty similar (Around ~4 stars), but that the ratings for each category do vary a bit. Notice that the category "Lifestyle" has the highest number of 5 star ratings!

Now let's take a look at price and size distribution.

## How are Price and Size distributed?

In [36]:

```
sns.set(rc={'figure.figsize':(15,10)})
price_plot = sns.countplot(x = 'Price', data=data[data.Price>0])
price_plot.set_title('Price Distribution for Paid Applications')

for ind, label in enumerate(price_plot.get_xticklabels()):
    if ind % 10 == 0: # every 10th label is kept
        label.set_visible(True)
    else:
        label.set_visible(False)

plt.figure()
size_plot = sns.countplot(x = 'Size', data=data)
size_plot.set_title('Distribution of Size (in Megabytes)')
for ind, label in enumerate(size_plot.get_xticklabels()):
    if ind % 25 == 0: # every 10th label is kept
        label.set_visible(True)
    else:
        label.set_visible(False)

print('Number of free and paid applications:\n {}'.format(data['Type'].value_counts()))

print('\nAverage App Price: {}'.format(data['Price'].mean()))
print('Median App Price: {}\n'.format(data['Price'].median()))

print('Average App Size: {}'.format(data['Size'].mean()))
print('Median App Size: {}\n'.format(data['Size'].median()))
```

Number of free and paid applications:

Free 8902

Paid 756

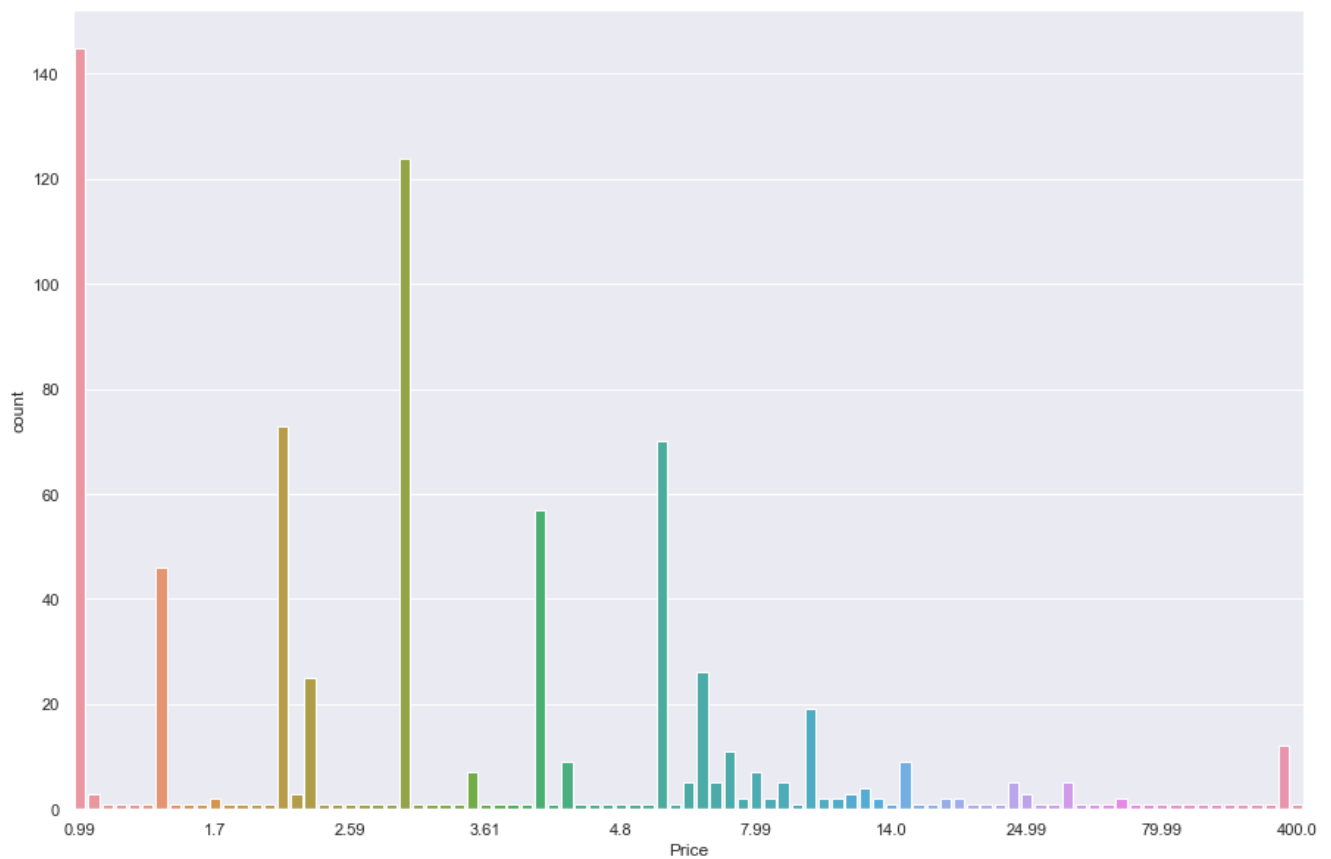
Name: Type, dtype: int64

Average App Price: 1.0992990992856404

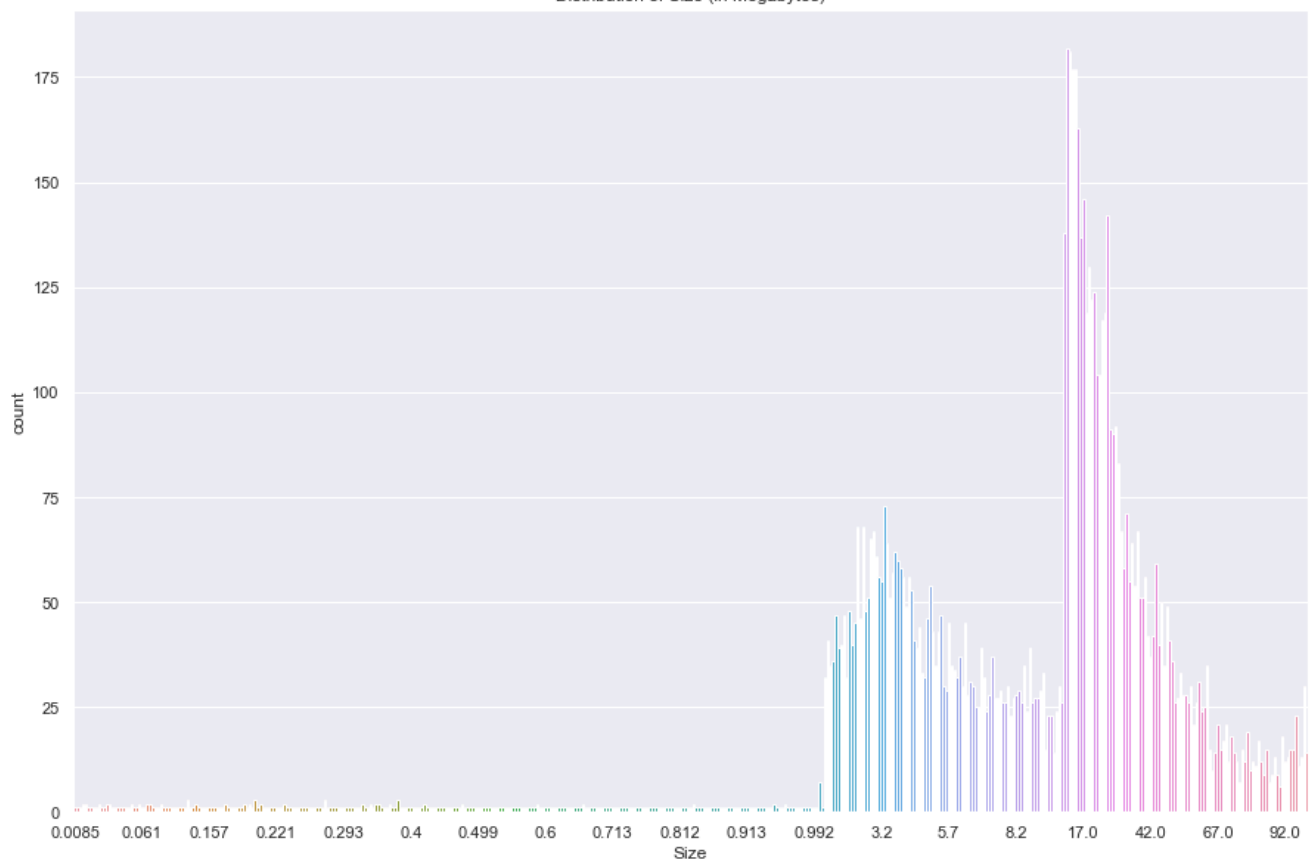
Median App Price: 0.0

Average App Size: 20.39528931451613

Median App Size: 12.0



Distribution of Size (in Megabytes)



A large majority of the applications are free - Here we are only plotting the paid application prices.

Note: Both of the Price/Size X axis are not to scale - due to so many different price/size points we decided to only keep a few of the labels.

There are a small amount of applications that cost a fair bit of money - what kind of applications are they?


**What kind of applications are expensive?**

## What kind of applications are expensive?


In [37]:

```
data[['App', 'Category', 'Price']][data.Price >= 200]
```

Out[37]:

	App	Category	Price
4197	most expensive app (H)	FAMILY	399.99
4362	 I'm rich	LIFESTYLE	399.99
4367	I'm Rich - Trump Edition	LIFESTYLE	400.00
5351	I am rich	LIFESTYLE	399.99
5354	I am Rich Plus	FAMILY	399.99
5355	I am rich VIP	LIFESTYLE	299.99
5356	I Am Rich Premium	FINANCE	399.99
5357	I am extremely Rich	LIFESTYLE	379.99
5358	I am Rich!	FINANCE	399.99
5359	I am rich(premium)	FINANCE	399.99
5362	I Am Rich Pro	FAMILY	399.99
5364	I am rich (Most expensive app)	FINANCE	399.99
5366	I Am Rich	FAMILY	389.99
5369	I am Rich	FINANCE	399.99
5373	I AM RICH PRO PLUS	FINANCE	399.99
9719	EP Cook Book	MEDICAL	200.00
9917	Eu Sou Rico	FINANCE	394.99
9934	I'm Rich/Eu sou Rico/أنا غني/我很有錢	LIFESTYLE	399.99

An example of one of these applications in the Google Store Menu:



## I'm Rich - Trump Edition

Rich Studios XXL Lifestyle

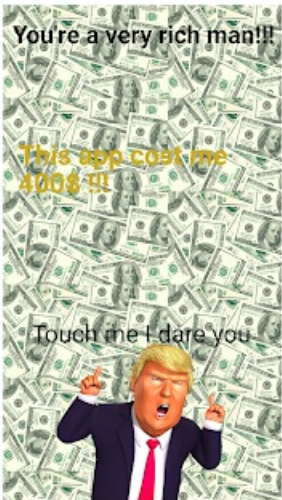


★★★★★ 291

Everyone

This app is compatible with your device.

Add to Wishlist

\$474.99 Buy

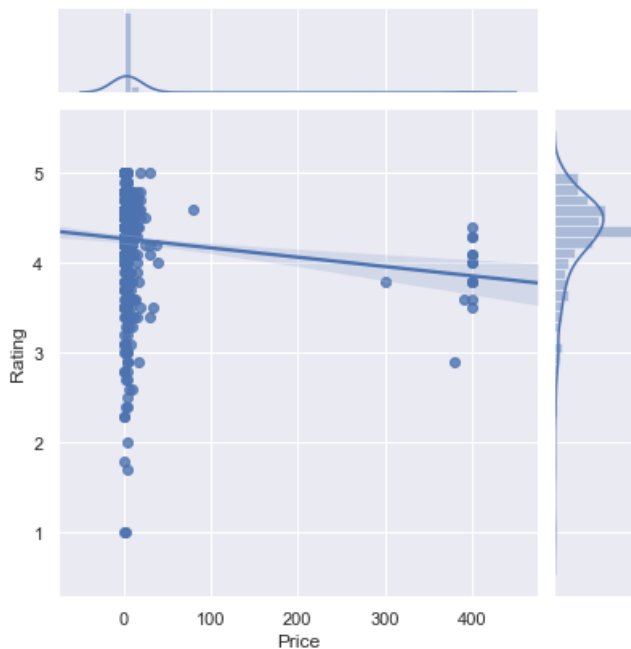
Seems like these applications are a bit silly and just for showing off.

## How does Price affect Rating?

In [39]:

```
sns.set(rc={'figure.figsize':(15,10)})  
# Here we only want to look at paid applications.  
rating_plot = sns.jointplot('Price', 'Rating', data[data.Price > 0], kind="reg")  
print("Ratings for Paid Applications")
```

Ratings for Paid Applications



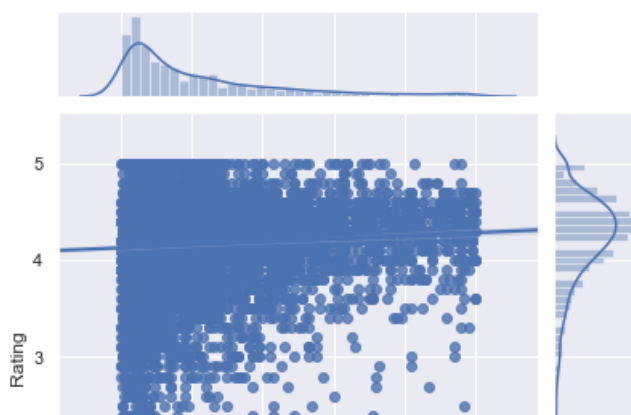
We can see that most paid applications have a high rating, and not surprisingly, some of the most expensive applications also have high ratings. (Who would pay that much money for a junk application and rate it bad?)

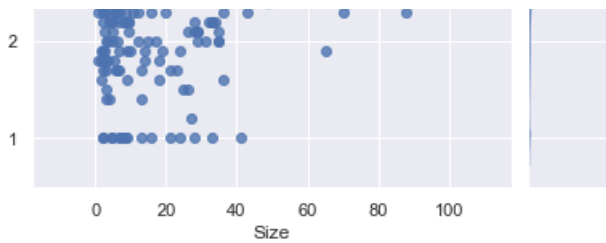
## How does size affect rating?

In [40]:

```
sns.set(rc={'figure.figsize':(15,10)})  
# Here we only want to look at paid applications.  
print('Rating according to Size of Application')  
size_rating_plot = sns.jointplot('Size', 'Rating', data, kind="reg")
```

Rating according to Size of Application





It looks like as the size of the application increases, the number of low ratings decreases.

## Pairplots for Free/Paid applications using Rating, Reviews, Size, Installs and Price

Lastly, let's plot the distribution of Free apps versus Paid apps using these attributes.

In [13]:

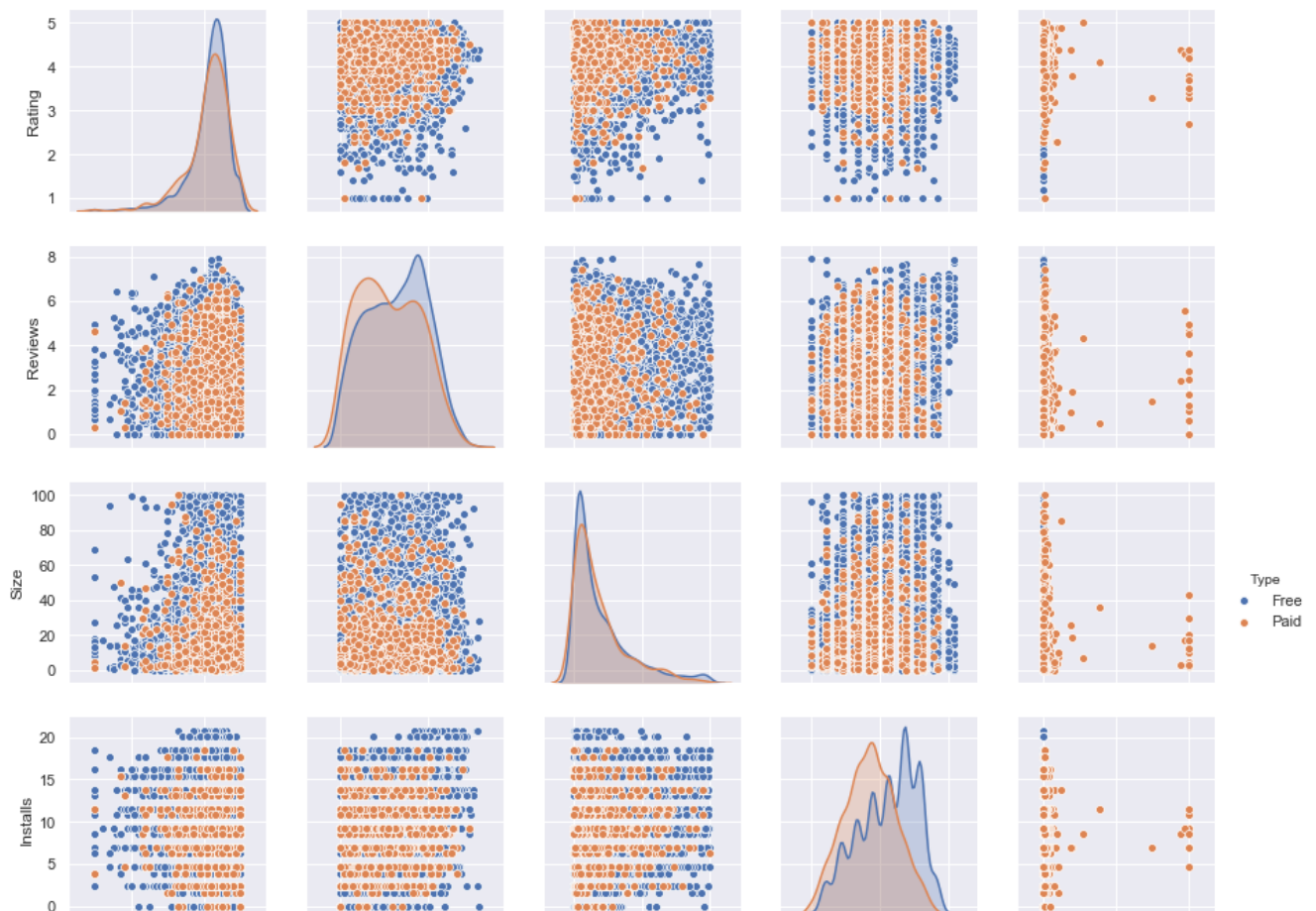
```
# Please note: Code here is referenced from the link below.
# https://www.kaggle.com/lava18/all-that-you-need-to-know-about-the-android-market

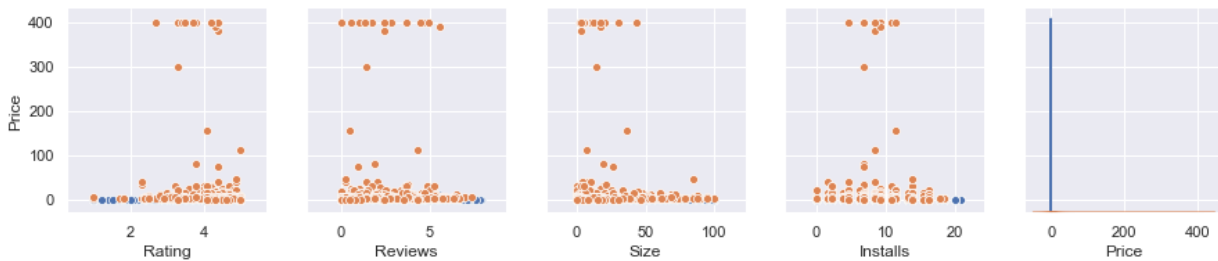
ratings = data['Rating'].dropna()
size = data['Size'].dropna()
installs = data['Installs'][data.Installs!=0].dropna()
reviews = data['Reviews'][data.Reviews!=0].dropna()
t = data['Type'].dropna()
price = data['Price']

pairplot_data = list(zip(ratings, np.log10(reviews), size, np.log(installs), t, price))
pairplot_data = pd.DataFrame(pairplot_data, columns=['Rating', 'Reviews', 'Size', 'Installs', 'Type', 'Price'])
sns.pairplot(pairplot_data, hue='Type')
```

Out[13]:

<seaborn.axisgrid.PairGrid at 0x121176d30>





Now that we've done some data exploration and visualization for some important attributes that could help determine what makes an app successful, let's take a look at our reviews dataset.

## 4. Data Mining/Analysis of Reviews with Naive Bayes for Text Classification

Using the Naive Bayes text classification algorithm, we conducted an analysis of user reviews based on the sentiment rating given to each review.

The reviews data file contains the first 'most relevant' 100 reviews for each app. Each review text or comment has been pre-processed and attributed with 3 new features: Sentiment, Sentiment Polarity, and Sentiment Subjectivity.

The user reviews data for application of the Google Playstore has a total of 37427 entries.

One important thing to note is that each review is not native english - they are translated reviews that use the Google Translate application.

First, we are going to load our data into a Pandas Dataframe. The data is in CSV format without a header line or any quotes. We can open the file with the open function and read the data lines using the reader function in the CSV module.

In [14]:

```
import pandas as pd
import random
import re

# Create the pandas DF object
data = pd.read_csv('googleplaystore_user_reviews.csv', encoding = "ISO-8859-1")
data.head()
```

Out[14]:

	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

We can see that there are some NaN values that we could clean up. Let's clean the data a little bit.

In [15]:

```
# Remove NaN values from reviews
data = data[pd.notnull(data['Translated_Review'])]
data.sample(5)
```

Out[15]:

App Translated\_Review Sentiment Sentiment\_Polarity Sentiment\_Subjectivity

	App	Translated Review	Sentiment	Sentiment_Polarity	Sentiment_Subjectivity
54646	Golfshot Plus: Golf GPS	Please make compatible Samsung watches.	Neutral	0.000000	0.000000
3137	ASOS	Intuitive visual app. Everything could possibl...	Neutral	0.000000	0.500000
43556	Family Dollar	Yeah I like smart coupons saving	Positive	0.214286	0.642857
43621	Family GPS tracker KidControl + GPS by SMS Loc...	I used love app....have used years made Google...	Positive	0.218750	0.425000
597	2GIS: directory & navigator	Always helps	Neutral	0.000000	0.000000

As we can see, some of the reviews are not native english and are a bit gramatically incorrect.

We're going to do a brief visualization of our data. Let's take a look at the sentiment distribution.

The figure below shows the counts for the sentiment values of our data. As we can see, there is a large amount of positive reviews. This correlates and helps explain that above in the metadata for Play Store applications, we had an average review of about 4.1 stars.

In [16]:

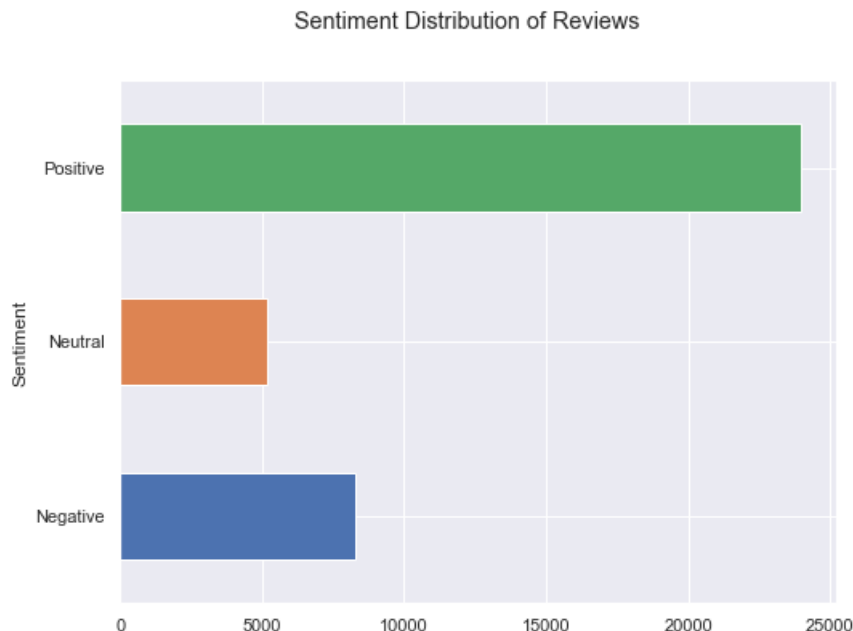
```
import matplotlib.pyplot as plt

print("\nSentiment Counts: \n")
print(data['Sentiment'].value_counts())

fig = plt.figure(figsize=(8,6))
fig.suptitle('Sentiment Distribution of Reviews')
data.groupby('Sentiment').Translated_Review.count().plot.barh(ylim=0)
plt.show()
```

Sentiment Counts:

```
Positive    23998
Negative    8271
Neutral     5158
Name: Sentiment, dtype: int64
```



For our classifier, since we have three different sentiment values, we are going to change these to values to integers. We'll also extract the review text and remove any punctuation and other things so we are training our model on the word values specifically. Corresponding sentiment values as integers:

- 'Positive' as 0
- 'Neutral' as 1
- 'Negative' as 2



In [17]:

```
def get_data(data):
    # Change dataframe to lists
    reviews = data['Translated_Review'].dropna().tolist()
    sentiment = data['Sentiment'].dropna().tolist()

    # Convert sentiment list string values to integer
    for index, item in enumerate(sentiment):
        if item == 'Positive':
            sentiment[index] = 0
        elif item == 'Neutral':
            sentiment[index] = 1
        elif item == 'Negative':
            sentiment[index] = 2

    return reviews, sentiment
```

The training model performs the following text preprocessing:

- removes capitalization,
- removes non-alphabetical symbols (a-z only),
- removes stop words (utilizing nltk.corpus),
- sets a minimum count (text frequency) for occurrences of a word to higher than three

With these preprocessing filters for the reviews, a high accuracy for the text classification can better be achieved.

In [18]:

```
from nltk.corpus import stopwords
model = {}

def train_model(data, sentiment):
    stop_words = set(stopwords.words('english'))
    # Initialize model & count holders for words
    count_of_words = {}
    sum_words_positive = 0
    sum_words_negative = 0
    sum_words_neutral = 0
    positive_count = 0
    negative_count = 0
    neutral_count = 0

    # Tokenize sentence and remove captilization
    reviews = [[y.lower() for y in x.split()] for x in data]

    for index, review in enumerate(reviews):
        # Remove stop words
        reviews[index] = [word for word in review if word not in stop_words]

    for index, sentence in enumerate(reviews):
        for word in sentence:
            # Word cleanup (remove punctuation etc.)
            word = re.sub(r'^a-z', '', word)

            # If we haven't seen the word yet, add it and initialize counts for
            # each sentiment to 0.
            if word not in model:
                model[word] = [0, 0, 0]

            # Update word count for sentiment and word in general
            model[word][sentiment[index]] += 1
            if word not in count_of_words:
                count_of_words[word] = 1
            else:
                count_of_words[word] += 1

    # Add the number of words in sentence to the sentiment vocab count.
    if sentiment[index] == 0:
        sum_words_positive += len(sentence)
        positive_count += 1
    elif sentiment[index] == 1:
        sum_words_neutral += len(sentence)
        neutral_count += 1
```

```

        elif sentiment[index] == 2:
            sum_words_negative += len(sentence)
            negative_count += 1

    return model, positive_count, negative_count, neutral_count, sum_words_positive,
    sum_words_negative, sum_words_neutral

```

A partition of the data set is split for train set and test set **randomly**.

Train data is 80% of the dataset.

Test data is 20% of the dataset.

It is important that the data be partitioned into these two subsets. We have a larger value for our training data so that our model can predict the sentiment class more effectively. We want to randomize the partition so that it is a unique training and test set each time the classifier runs.

In [19]:

```

# Split the data randomly.
train_data = data.sample(frac=0.8)
test_data = data.drop(train_data.index)

train_data, train_sentiment = get_data(train_data)
train_model_dict, positive_count, negative_count, neutral_count, sum_words_positive, sum_words_negative, sum_words_neutral = train_model(train_data, train_sentiment)
test_data, test_sentiment = get_data(test_data)

```

The resulting sentiment probabilities are calculated below.

In [20]:

```

total = positive_count + negative_count + neutral_count
probability_positive = positive_count/total
probability_negative = negative_count/total
probability_neutral = neutral_count/total

print ("Sentiment Probabilities: \n" + "Positive: " + str(probability_positive) +
      "\nNeutral: " + str(probability_neutral) + "\nNegative: " + str(probability_negative))

# Data to plot
labels = 'Positive [0]', 'Neutral [1]', 'Negative [2]'
sizes = [probability_positive, probability_neutral, probability_negative]
colors = ['green', 'yellow', 'red']

# Plot
plt.pie(sizes, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)
plt.axis('equal')
plt.suptitle('Sentiment Probability Distributions')
plt.show()

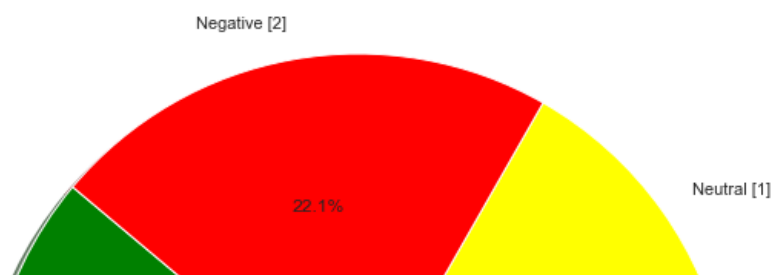
```

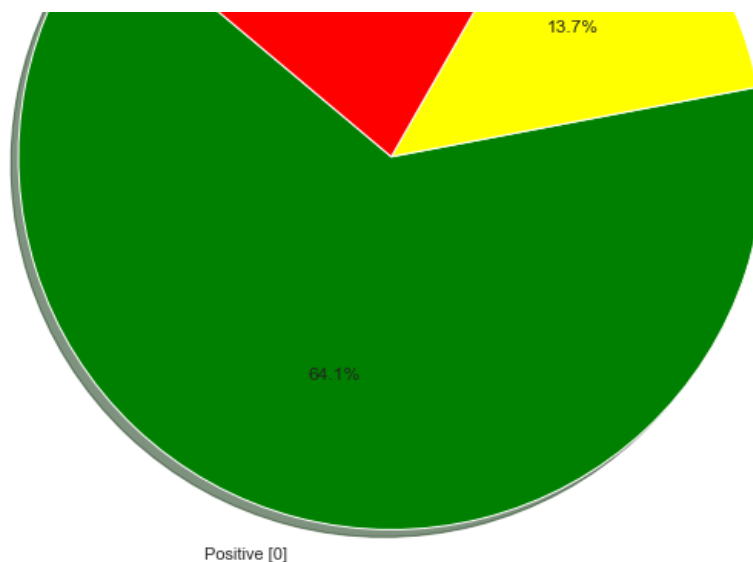
```

Sentiment Probabilities:
Positive: 0.6410393427292767
Neutral: 0.1374991650524347
Negative: 0.2214614922182887

```

Sentiment Probability Distributions





We can see that there is a large share of positive sentiments.

To avoid the zero frequency problem we do the following:

$$P(t|c) = \frac{T_{c,t} + 1}{\sum_{t \in V} (T_{c,t} + 1)} = \frac{T_{c,t} + 1}{(\sum_{t \in V} T_{c,t}) + |V|}$$

We create a list with its index representing the 0, 1, 2 sentiment values, as previously mentioned.

In [21]:

```
def calc_cond_probabilities(train_model_dict, positive_count, negative_count,
                           neutral_count, sum_words_positive, sum_words_negative, sum_words_neutral):
    cond_probabilities = {}

    for word in model:
        if word not in cond_probabilities:
            cond_probabilities[word] = [0, 0, 0]

        # Calculate conditional probabilities
        cond_probabilities[word][0] = (model[word][0] + 1) / (sum_words_positive + len(model))
        cond_probabilities[word][1] = (model[word][1] + 1) / (sum_words_neutral + len(model))
        cond_probabilities[word][2] = (model[word][2] + 1) / (sum_words_negative + len(model))

    return cond_probabilities

conditional_probabilities = calc_cond_probabilities(train_model_dict, positive_count,
negative_count, neutral_count, sum_words_positive, sum_words_negative, sum_words_neutral)
```

To avoid number overflow, we operate on the logs of probabilities on our data below:

$$\log P(c|d) = \log \alpha + \log P(c) + \sum_{t \in d} \log P(t|c)$$

In [22]:

```
import math

def classify_test_data(words, probability_positive, probability_negative, probability_neutral, conditional_probabilities):
    positive = 1
    negative = 1
    neutral = 1
    for word in words:
        if word in model:
            positive += math.log2(conditional_probabilities[word][0])
            neutral += math.log2(conditional_probabilities[word][1])
            negative += math.log2(conditional_probabilities[word][2])

    positive += math.log2(probability_positive)
```

```

positive += math.log2(probability_positive)
negative += math.log2(probability_negative)
neutral += math.log2(probability_neutral)

if (positive > negative) and (positive > neutral):
    return 0
elif (neutral > positive) and (neutral > negative):
    return 1
else:
    return 2

```

In [23]:

```

result = []
for i, element in enumerate(test_data):
    words = element.split(' ')
    result.append(classify_test_data(words, probability_positive, probability_negative, probability_neutral, conditional_probabilities))

correct = 0
incorrect = 0

for i, j in zip(result, test_sentiment):
    if i == j:
        correct += 1
    else:
        incorrect += 1

print("Accuracy: " + str((correct/len(test_sentiment))*100))

```

Accuracy: 73.13293253173012

## Results

Our classifier predicted each review's sentiment with about 73% accuracy. Why was this (relatively) low?

Originally we did not remove stopwords, and had an accuracy of about 68%. We decided to remove stopwords since they account for about 20-30% of the total word counts, and this improved the efficiency and accuracy of our model. It is worth noting that these are not all English reviews however - these are translated reviews done by Google translate. Further research or improvements for the model could be done by comparing our model against a model trained using English reviews that are not translated. We believe that this is part of the reason that the accuracy is lower than expected. Some of the translated reviews may have words that are not translated entirely correctly, or perhaps simply incorrect.

To check our model's accuracy, we ran the dataset using the SKLearn Naive Bayes MultinomialNB Module. It implements the Naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification. We want to compare the effectiveness of our model with the result of SKlearn's.

In [24]:

```

import pylab

col = ['Sentiment', 'Translated_Review']
df = pd.read_csv('googleplaystore_user_reviews.csv', encoding = "ISO-8859-1")
df = df[pd.notnull(df['Translated_Review'])]
df = df[col]
df.columns = ['Sentiment', 'Translated_Review']

```

We will remove missing values in "Translated\_Review" column, and add a column encoding the product as an integer because categorical variables are often better represented by integers than strings.

In [25]:

```

df['sentiment_id'] = df['Sentiment'].factorize()[0]
sentiment_id_df = df[['Sentiment', 'sentiment_id']].drop_duplicates().sort_values('sentiment_id')
sentiment_to_id = dict(sentiment_id_df.values)
id_to_sentiment = dict(sentiment_id_df[['sentiment_id', 'Sentiment']].values)
df.head()

```

Out[25]:

	Sentiment	Translated_Review	sentiment_id
0	Positive	I like eat delicious food. That's I'm cooking ...	0
1	Positive	This help eating healthy exercise regular basis	0
3	Positive	Works great especially going grocery store	0
4	Positive	Best idea us	0
5	Positive	Best way	0

As we can see from above, some of the translated reviews aren't entirely gramatically correct. This could have affected the accuracy of our model. Below, we run the SKLearn Naive Bayes on our dataset.

In [26]:

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB

X_train, X_test, y_train, y_test = train_test_split(df['Translated_Review'], df['Sentiment'])
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train)
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)

clf = MultinomialNB().fit(X_train_tfidf, y_train)
from sklearn import metrics

y_pred = clf.predict(count_vect.transform(X_test))
print(metrics.classification_report(y_test, y_pred,
                                     target_names=df['Sentiment'].unique()))
print("Accuracy: " + str(metrics.accuracy_score(y_test, y_pred)*100))
```

	precision	recall	f1-score	support
Positive	0.84	0.37	0.52	2064
Neutral	0.99	0.05	0.10	1332
Negative	0.71	0.99	0.82	5961
micro avg	0.72	0.72	0.72	9357
macro avg	0.84	0.47	0.48	9357
weighted avg	0.78	0.72	0.65	9357

Accuracy: 72.10644437319654

We see that the SKLearn's model predicts the sentiment value of reviews with a very similar accuracy to ours.

## 5. Conclusions

Using our dataset of metadata and reviews for mobile applications on the Google Play Store, we conducted a brief exploration and visualization of trends for current applications on the Google Play Store, as well as analyze the sentiment values for reviews of current applications on the Google Play Store.

From our dataset on metadata of some Google Play Store applications, we found that a large majority of the applications fall into the application categories "Family" and "Games". Interestingly, most applications are highly rated, with an average rating of 4.1 stars of out 5, and a median of 4.3 stars. While most applications in our dataset were free, a large majority of applications fall between \$0.99 and \$2.99, with a few outlier applications costing over \$200, with a brief investigation determining that some of these applications could fall into a junk category with no uses for the app.

For our dataset on reviews and review sentiment for Google Play Store applications, we found that a large majority of the reviews were classified as positive, which seems correlated to the high average app rating found in our metadata exploration above. Although our classifier had a lower accuracy for predicting new reviews (About ~72-73 percent), we checked our model against the SKLearn's Naive Bayes module, which predicted sentiment for new reviews with similar accuracy. We believe that this accuracy may be correlated to the fact that a majority of the reviews were not in english, and translated to english using the Google Translator, which resulted in some sentences with grammar errors. Further research and study could be conducted using a dataset of entirely native english reviews.

We hope you found our brief exploration and data mining of applications on the Google Play Store interesting, thank you!

## 6.0 References

[1] R. Saifi, "The 2017 Mobile App Market: Statistics, Trends, and Analysis", 2017. Available:

<https://www.business2community.com/mobile-apps/2017-mobile-app-market-statistics-trends-analysis-01750346>

[Accessed: 2018-11-21]

[2] W. Koehrsen, "Visualizing Data with Pair Plots in Python", 2018. Available:

<https://towardsdatascience.com/visualizing-data-with-pair-plots-in-python-f228cf529166>

[Accessed: 2018-11-24]