COGS 108 - The Features of Successful Mobile Applications

Overview

Nowadays, due to the huge variety of moblie applications, the competition inside the market is extremely high. Every application that wants to be successful has to utilize all available data in order to improve its quality and to avoid possible decisions that could detriment its success. Thus, correct analysis of different phenomena related to applications and discovery of any hidden patterns inside relevant data are essential in helping applications achieve these goals.

Our team came up with the idea that an application's name could be an influential indicator of its success, and we hope to explore upon this idea in our project.

Names

- Aliaksandr Samushchyk
- Jiayi Zhang
- Soumya Agrawal
- Richard Duong
- Titan Ngo
- Yaman El-Jandali

Group Members IDs

- A15672156
- A14533542
- A14402679
- A15196673
- A15525832
- A15753076

Research Questions

There are several specific data science questions that we formulated. We expect that the answers to these questions will enable us to approve or disprove our hypothesis.

1) Main Question: Does the length of the title of an application influence its success?

Subquestions:

2) Are there any letters/combinations of letters that tend to be present in more/less successful applications?

3) What are the other variables that are significant?

Hypothesis

The main hypothesis is the following:

```
- The length of the title of an application does influence its success. (1)
```

We came up to this hypothesis because of several reasons. Sometimes it happens that we think about an application, we know it, but we are unable to remember its title. It makes it difficult to find and install this app or to suggest this app to another person. It could also happen that the title is so long to read that we become less interested in it. These ideas gave a rise to the hypothesis (1).

We predict to measure the success of an application using a few metrics such as the number of installations and the rating of an application.

We expect our hypothesis to be approved; in fact, as we stated previously, the length of title is likely to be significant in determining apps success. Additionally, we suppose that there might be some letters/combinations of letters that tend to be present in the title of more (or less) successful applications. We also expect that a huge amount of letters/combinations of letters will be insignificant, because it may seem that the usage of letters in titles is in some sense random.

We hope to find other variables that are significant in explaining success. We think that size and price of applications have the effect. We expect the coefficients on these variables to be negative, because it is logical that the more the price/size of an app the less installs it would have.

Methodology

We will try to answer previously stated questions in our analysis. First of all, we will introduce our dependent variable (measure of success). In futher text it will be denoted as Y. Our main goal is to estimate a model (using OLS regression) of the form:

$$Y_i = \beta^T \cdot x_i + e_i$$

Where:

- β is a vector of coefficients,
- x_i - vector of regressors,
- e_i - disturbance term;

Remember the OLS assumptions:

1) Linear form

2) Homoscedasticity in errors

3) No multicollinearity

Since there will be a lot of binary independent variables in out set of regressors(e.g. a column 'z', taking value 1 if an application's name contains 'z' inside it, and zero otherwise), we expect our matrix of regressors suffer from severe multicollinearity. In order to address this issue we will perform **Wilcoxon rank sum test** on each of these binary variables separetely to determine the most relevant and significant ones. Once we determine such variables, we will keep them in our set of regressors. With the help of this procedure our attempt to satisfy third OLS assumption will have high chances to succeed.

Then, we will run our OLS regression(**simple OLS**). Immideately after that, we will be able to perform **Breusch-Pagan test** to see whether the errors of regression satisfy the second assumption. If they do not satisfy, we will run new regression. It will be still **OLS regression**, but this time it will be **robust to heteroscedasticity** in errors. The reason why simple OLS(without robustness check) first is that if the errors of that regression satisfy the second assumption, the standard errors of estimated coefficients will be lower but still valid and correct. Thus, the p-values will also be lower and more accurate.

Background and Prior Work

The dataset that we found presents a lot of interesting insights about user preferences, and we want to explore the effects of certain user preferences on app rating and popularity. There is not a lot of background knowledge that we know about user preferences affecting app popularity other than the obvious correlations between app rating/popularity, so we would like to use the data to try and find more subtle biases that might affect it such as title/title length, etc.

References:

- 1) “A Statistical Analysis of the Apple App Store” by Colin Eberhardt Did a statistical analysis of prices of apps in the Apple App store. Did not do much other than basic statistical analysis, such as looking at the genre distribution of apps and the price differences across genres. Found a positive correlation between price and app rating. Source: https://blog.scottlogic.com/2014/03/20/app-store-analysis.html (https://blog.scottlogic.com/2014/03/20/app-store-analysis.html)
- 2) Did a statistical analysis of various factors that contribute to the success of an app in the Google Play Store. Found that most free apps are monetized by advertisements. Learned that ~80% of apps on the playstore have been downloaded less than 50k times. Found that a small amount of users who install actually take the time to write a review. Source: https://nycdatascience.com/blog/student-works/web-scraping/analysis-of-apps-in-the-google-play-store/ (https://nycdatascience.com/blog/student-works/web-scraping/analysis-of-apps-in-the-google-play-store/)

The scope of our project is a bit beyond the analysis that these projects present, but in a similar vein. While these projects analyzed the more basic factors that weigh in the success of an app, we will be focusing more on subtle user preferences that are not as obvious to correlate to success.

Dataset

- Dataset Name: Google Play Store Apps
- Link to the dataset: https://www.kaggle.com/lava18/google-play-store-apps?fbclid=IwAR0l6EIgxdnc3LWhwVVg85gZ9RokprTW6xDo47EQxwDu5Qkce24ZC2MbIBs#googleplaystore_user_reviews.csv (https://www.kaggle.com/lava18/google-play-store-apps?fbclid=IwAR0l6EIgxdnc3LWhwVVg85gZ9RokprTW6xDo47EQxwDu5Qkce24ZC2MbIBs#googleplaystore_user_reviews.csv)

Our dataset is the Google Play Store Apps data, which is web scraped from the Google Play Store and presented in a csv format on Kaggle.

This dataset provides up with a zipped folder containing two files. The first file, googleplaystore_user_reviews, has around 64.3k observations, with 5 variables/columns. These include the app name, translated review, sentiment, sentiment polarity, and sentiment subject. Most of these variables have been preprocessed already, such as the translated review and sentiment. Many observations in this file (or the reviews), are for the same app. For our project, we will not be focusing on the specific text in the reviews, so this file will not be used. Our project focuses on the second file, googleplaystore. As for the second file, there are around 10.8k observations with 13 variables. The variables include app name, category of the app, the overall user rating, number of reviews, size of the app, number of installs, whether the app is paid or free, the price of the app, the content rating, the genres, when the data was last updated, the current version of the app available on the app store, and the minimum required Android version for the app.

With this data, we can analyze different features of the app to help determine what makes an app successful and help future app developers to attain success in the Android market. Specifically, we will focus on determining factors other than the rating or number of installs to determine an app's popularity. For this purpose, we will focus on an app's title/name, an app's size, an app's price, and compare these variables with the number of reviews, an app's rating, and the number of installs.

We can use an apps title by doing analysis on the length of the title, letters used, and specific combinations used to determine if certain letters, combinations, or title lengths are features of more successful apps. Using the app's size, we can determine if smaller apps are more likely to be successful. Using the app's price, we can see if free apps appeal to consumers more and are more popular than paid apps. We can analyze the number of reviews, an app's rating, and the number of installs to determine what will be the best dependent variable for us to compare the previous features to. Features like the category, when the app was last updated, the current version of the app available on the app store, and the minimum required Android version for the app are variables that we will not analyze in our data as we are focusing our question on the title of the app rather than its "technical" features.

Setup

In [257]:
```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import string
import scipy
import statsmodels.api as sm
import statsmodels
import warnings

# Display plots directly in the notebook instead of in a new window
%matplotlib inline
```

In [258]:
```
# We decided to configure out settings exactly like we did for A2

# Configure libraries
# The seaborn library makes plots look nicer
sns.set()
sns.set_context('talk')

# Don't display too many rows/cols of DataFrames
pd.options.display.max_rows = 7
pd.options.display.max_columns = 8

# Round decimals when displaying DataFrames
pd.set_option('precision', 2)

#hides non-fatal alerts
warnings.filterwarnings("ignore")
```

Data Wrangling

We have two main files in our data set: googleplaystore.csv and googleplaystore_user_reviews.csv.

For an analysis of app names, we can focus on just the first data set. The second data set gives us reviews of each app, which is not needed for this particular project since we're focusing on app titles.

Brief Overview

Before beginning our analysis, we cleaned our data in several key stages.

1. Loaded Data
2. Previewed Size and Information
    - Determined what information/columns are most relevant
3. Removed Duplicates
4. Removed NaN
5. Pre-Processing
    - Converted strings to numeric values
    - Removed nonsensical data points (e.g. negative ratings, etc...)
    - Addressed concerns regarding outliers
6. Analyse Distribution -Additionally adjust distributions/tranform data for accurate visualizations

In [259]:
```
df = pd.read_csv("google_play_dataset/googleplaystore.csv")
```

In [260]:
```
# Preview dataset and get a grasp on its size
df
```

Out[260]:

	App	Category	Rating	Reviews	...	Genres	Last Updated	Current Ver	Android Ver
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	...	Art & Design	January 7, 2018	1.0.0	4.0.3 and up
1	Coloring book moana	ART_AND_DESIGN	3.9	967	...	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	...	Art & Design	August 1, 2018	1.2.4	4.0.3 and up
...	...	...	...	...	...	...	...	...	...
10838	Parkinson Exercices FR	MEDICAL	NaN	3	...	Medical	January 20, 2017	1.0	2.2 and up
10839	The SCP Foundation DB fr nn5n	BOOKS_AND_REFERENCE	4.5	114	...	Books & Reference	January 19, 2015	Varies with device	Varies with device
10840	iHoroscope - 2018 Daily Horoscope & Astrology	LIFESTYLE	4.5	398307	...	Lifestyle	July 25, 2018	Varies with device	Varies with device

10841 rows × 13 columns

In [261]:
```
# Show the headers of the different columns
print(list(df.columns.values))
```

```
['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type', 'Price', 'Content Rating',
'Genres', 'Last Updated', 'Current Ver', 'Android Ver']
```

In [262]:
```
# Remove any duplicate applications
# We only want to account for each app once
df.drop_duplicates(['App'], inplace = True)
```

In [263]:
```
# We decided that we only need the app names, ratings, and number of installations.
df = df[['App', 'Rating', 'Installs', 'Price', 'Size', 'Reviews', 'Category']]
```

After previewing our dataset, with 10841 observations and 13 variables, we realized that it has a lot of data we do not need.

The **'Size'** of apps, the **'Type'** (either free or paid), **'Price'**, **'Content Rating'** (as in the rating for the suggested age group), the date of when they were **'Last Updated'**, the **'Genres'** and **'Category'**, and the versions (**'Current Ver'** and **'Android Ver'**) have little to do with our analysis of specific characterisitcs of the length of title strings, so those columns were dropped.

Our project seeks to see any relationship between characteristics of apps' titles with their success. However, we do not have any direct measure of this **"*Success*"**. We decided to keep the **'Rating'** and **'Installs'** columns to determine **"*Success*"**, but we still want to look at the distributions/correlations of these variables before committing to a specific method of calculation.

In [264]:
```
# Count amount of missing values in the dataset
app_nans = df['App'].isna().sum()
rating_nans = df['Rating'].isna().sum()
installs_nans = df['Installs'].isna().sum()
print("app_nans:", app_nans,"\nrating_nans:", rating_nans,"\ninstalls_nans:", installs_nans)
```

```
app_nans: 0
rating_nans: 1463
installs_nans: 0
```

There seems to be a considerable amount of observations with missing values. However, the only column in our current dataset that has missing values is **'Ratings'**. Since we absolutely need ratings values for every observation that we have, we will drop all of the observations without them.

In [265]:
```
# Drops observations missing 'Ratings' data
df = df.dropna()

# Preview the dataframe
df
```

Out[265]:

	App	Rating	Installs	Price	Size	Reviews	Category
0	Photo Editor & Candy Camera & Grid & ScrapBook	4.1	10,000+	0	19M	159	ART_AND_DESIGN
1	Coloring book moana	3.9	500,000+	0	14M	967	ART_AND_DESIGN
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	4.7	5,000,000+	0	8.7M	87510	ART_AND_DESIGN
...	...	...	...	...	...	...	...
10837	Fr. Mike Schmitz Audio Teachings	5.0	100+	0	3.6M	4	FAMILY
10839	The SCP Foundation DB fr nn5n	4.5	1,000+	0	Varies with device	114	BOOKS_AND_REFERENCE
10840	iHoroscope - 2018 Daily Horoscope & Astrology	4.5	10,000,000+	0	19M	398307	LIFESTYLE

8197 rows × 7 columns

Data Pre-Processing & Exploratory Data Analysis

It turns out that the 'Installs' column is a column of strings instead of numbers. We want to convert these strings into numeric values so that we can plot them.

In [266]:
```
# Delete the '+' chars that are in these strings in 'Installs'
df['Installs'] = df['Installs'].str.replace('+', '')
# Delete the ',' chars that are in these strings in 'Installs'
df['Installs'] = df['Installs'].str.replace(',', '')

# Type cast these strings into integers
df['Installs'] = pd.to_numeric(df['Installs'], errors='coerce')

#change 'M' to E6 in the Reviews column so that millions can be properly converted numerically
df['Reviews'] = df['Reviews'].str.replace('M', 'E6')
df['Reviews'] = pd.to_numeric(df['Reviews'], errors='raise')

#Remove the 'M' which signifies megabytes in the Size column, make everything in units of kilobyt
es
df['Size'] = df['Size'].str.replace('M', 'E3')
df['Size'] = df['Size'].str.replace('k', '')
df['Size'] = df['Size'].str.replace(',', '')

#We will also remove apps that have varying sizes since we won't be able to model these variation
s
df['Size'] = df['Size'].replace({'Varies with device': '-1'}, regex=True)
df['Size'] = df['Size'].str.replace('+', '')
df['Size'] = pd.to_numeric(df['Size'], errors='raise')

#Remove sizes that don't make any sense and ones we flagged to be removed earlier
df = df.drop(df[df.Size <= 0].index)

#Remove the money sign from app price
df['Price'] = df['Price'].str.replace('$', '')

#Set apps with the price of "Everyone" to be removed
df['Price'] = df['Price'].str.replace('Everyone', '-1')

#Change price to be a numeric column
df['Price'] = pd.to_numeric(df['Price'], errors='raise')

#Remove prices that don't make sense
df = df.drop(df[df.Price < 0].index)
# Preview the dataframe
# Values in 'Installs' should be numeric
df
```

Out[266]:

	App	Rating	Installs	Price	Size	Reviews	Category
0	Photo Editor & Candy Camera & Grid & ScrapBook	4.1	1.00e+04	0.0	19000.0	159.0	ART_AND_DESIGN
1	Coloring book moana	3.9	5.00e+05	0.0	14000.0	967.0	ART_AND_DESIGN
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	4.7	5.00e+06	0.0	8700.0	87510.0	ART_AND_DESIGN
...	...	...	...	...	...	...	...
10836	Sya9a Maroc - FR	4.5	5.00e+03	0.0	53000.0	38.0	FAMILY
10837	Fr. Mike Schmitz Audio Teachings	5.0	1.00e+02	0.0	3600.0	4.0	FAMILY
10840	iHoroscope - 2018 Daily Horoscope & Astrology	4.5	1.00e+07	0.0	19000.0	398307.0	LIFESTYLE

7027 rows × 7 columns

Since we need some measures of title length, we will create two columns: **'Word Count'** (counts the number of words in the title) and **'Char Count'** (counts the number of characters in the title).

In [267]:
```
# Create the word count column
df['Word Count'] = df['App'].str.split().str.len()
```

In [268]:
```
# Create the char count column
df['Char Count'] = df['App'].str.len() - df['App'].str.count(' ')
```

In [269]:
```
# Preview table again to check for these columns
df
```

Out[269]:

	App	Rating	Installs	Price	...	Reviews	Category	Word Count	Char Count
0	Photo Editor & Candy Camera & Grid & ScrapBook	4.1	1.00e+04	0.0	...	159.0	ART_AND_DESIGN	9	38
1	Coloring book moana	3.9	5.00e+05	0.0	...	967.0	ART_AND_DESIGN	3	17
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	4.7	5.00e+06	0.0	...	87510.0	ART_AND_DESIGN	10	41
...	...	...	...	...	...	...	...	...	...
10836	Sya9a Maroc - FR	4.5	5.00e+03	0.0	...	38.0	FAMILY	4	13
10837	Fr. Mike Schmitz Audio Teachings	5.0	1.00e+02	0.0	...	4.0	FAMILY	5	28
10840	iHoroscope - 2018 Daily Horoscope & Astrology	4.5	1.00e+07	0.0	...	398307.0	LIFESTYLE	7	39

7027 rows × 9 columns

We also want to add columns regarding the frequency of certain letters and letter pairs.

The code below creates many columns with binary variables depending on the presence of letters and letter pairs. For example, values in column 'a' are equal to 1 if 'a' is contained in apps name and 0 otherwise, and values in column 'wq' are equal to 1 if 'wq' is contained in apps name and 0 otherwise.

Then, we count how many ones are in each of the created columns to assess if the column is useful. If there are not a lot of ones in a column (less than 0.025 *(# of rows) for example) or too many (more than 0.975* (# of rows) for example), we remove such columns. This is done because, for example, there are not a lot of titles with 'wq' inside them so 'wq' column is not useful for our analysis

In [270]:
```
alphabet = list(string.ascii_lowercase)

for k in alphabet:
    q1 = []
    for i in df['App']:
        if k in i:
            q1.append(1)
        else:
            q1.append(0)
    df[k] = q1

alphabet1 = []
for i in alphabet:
    for k in alphabet:
        alphabet1.append(i + k)

for k in alphabet1:
    q2 = []
    for i in df['App']:
        if k in i:
            q2.append(1)
        else:
            q2.append(0)
    df[k] = q2


rem_list = []           ##here 13 and 715 are indexes of first and last created columns
for i in range(13,707): ##it is possible when you will try to execute code you will need to inser
t proper values
    if sum(df[df.columns[i]]) < 0.025*df.shape[0] or sum(df[df.columns[i]]) > 0.975*df.shape[0]:
        rem_list.append(i)


s = 0
for i in rem_list:
    i = i - s
    del df[df.columns[i]]
    s += 1
```

In [271]:
```
# Preview new dataset
df
```

Out[271]:

	App	Rating	Installs	Price	...	zw	zx	zy	zz
0	Photo Editor & Candy Camera & Grid & ScrapBook	4.1	1.00e+04	0.0	...	0	0	0	0
1	Coloring book moana	3.9	5.00e+05	0.0	...	0	0	0	0
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	4.7	5.00e+06	0.0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...
10836	Sya9a Maroc - FR	4.5	5.00e+03	0.0	...	0	0	0	0
10837	Fr. Mike Schmitz Audio Teachings	5.0	1.00e+02	0.0	...	0	0	0	0
10840	iHoroscope - 2018 Daily Horoscope & Astrology	4.5	1.00e+07	0.0	...	0	0	0	0

7027 rows × 159 columns

In [272]:
```
# Preview the columns of some example letters
df[['a','b','c']]
```

Out[272]:

	a	b	c
0	1	0	1
1	1	1	0
2	1	0	1
...	...	...	...
10836	1	0	1
10837	1	0	1
10840	1	0	1

7027 rows × 3 columns

In [273]:
```
# Viewing general statistics of our data
df.describe()
```

Out[273]:

	Rating	Installs	Price	Size	...	zw	zx	zy	zz
count	7027.00	7.03e+03	7027.00	7027.00	...	7027.0	7027.0	7.03e+03	7.03e+03
mean	4.16	4.47e+06	1.17	21754.43	...	0.0	0.0	2.28e-03	7.54e-03
std	0.56	2.71e+07	18.20	22726.50	...	0.0	0.0	4.77e-02	8.65e-02
...	...	...	...	...	...	...	...	...	...
50%	4.30	1.00e+05	0.00	13000.00	...	0.0	0.0	0.00e+00	0.00e+00
75%	4.50	1.00e+06	0.00	31000.00	...	0.0	0.0	0.00e+00	0.00e+00
max	5.00	1.00e+09	400.00	100000.00	...	0.0	0.0	1.00e+00	1.00e+00

8 rows × 157 columns

Data Wrangling Conclusion

Looking at our table at the beginning and near the end, we were lucky with our dataset. We were able to keep the majority of our values since we were looking at the features that are simpler numerical values. App data seems to be fairly easy to work with and most of the data points we will be analysing are mostly numerical values. Additionally, the data uploaded was scraped from Google Play store and is more reliable since it is all publicly available. Thanks to the simplicity of our data points, by this point our data is ready for us to utilize in our analysis and visualization.

However, there were some issues that we ran into that may affect the accuracy of our analysis. The number of installations is rounded and is not a continuous value. Additionally, we suspect successful apps are more likely to snowball and become more successful thus creating a gap between apps with very little success and significant success. Distribution is additionally addressed later in the notebook and transformations we used can be seen in the visualization section. Lastly, rating may be rather unreliable when an app has too few reviews.

Even with this potential flaws in the data, however, we are confident we can still extrapolate information from our data and answer our hypothesis.

Let's Start Plotting! - EDA

Plotting the indivdual variables

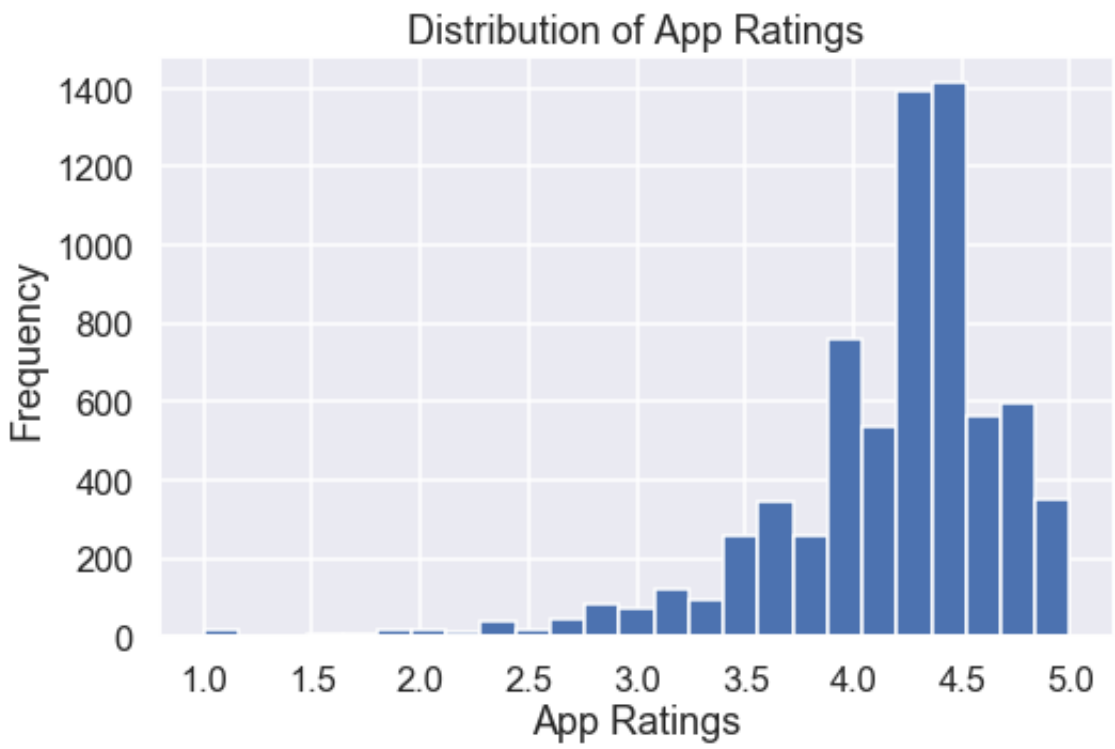
Rating

Ratings in the Google Play Store can only be values in the range between 0 and 5. We will drop ratings that are past the maximum 5 stars as an extra precaution.

In [274]:
```
# Drops invalid ratings (greater than 5)
df = df[df.Rating <= 5]
```

In [275]:
```
# Plots 'Rating' of apps (which are values from 0 to 5) to see its distribution
df['Rating'].plot.hist(bins = 25, figsize=(8,5))
plt.xlabel('App Ratings')
plt.ylabel('Frequency')
plt.title('Distribution of App Ratings')
```

Out[275]: Text(0.5,1,'Distribution of App Ratings')



In [276]:
```
# Calculate the mean of 'Rating'
mean = df['Rating'].mean()
median = df['Rating'].median()
print(" The mean is:", mean, '\n', "The median is:", median)
```

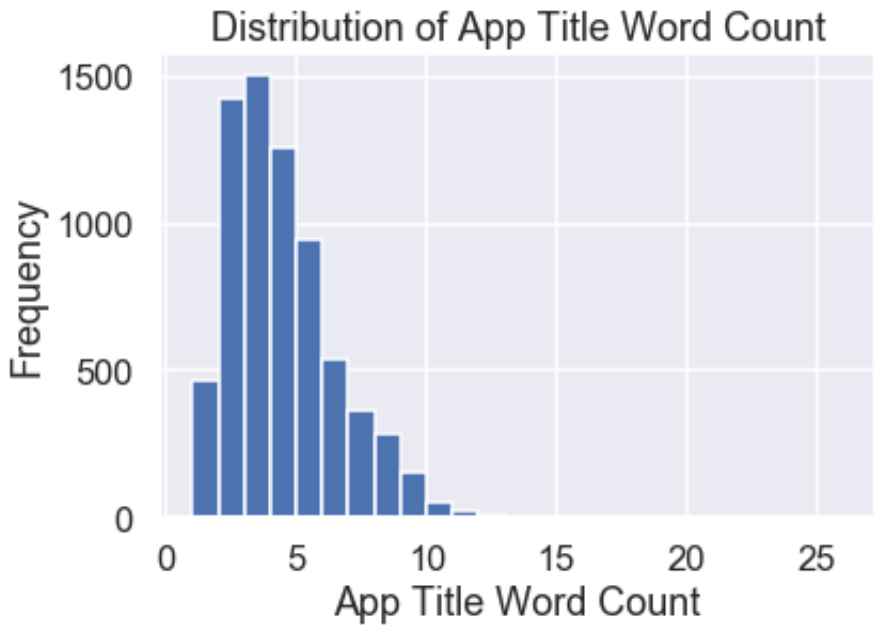
```
 The mean is: 4.160623310089655
 The median is: 4.3
```

From the above histogram, we can say that most apps have high ratings, (with a mean rating of ~4.17) probably due to peoples' rating tendencies. The graph aboves shows us that there is a greater number of apps that are rated from 2.5 - 5 than 0 - 2.5. Distribution-wise, **ratings are skewed left**.

Word Count

In [277]:
```
# Plots 'Word Count' of app titles to see its distribution
df['Word Count'].plot.hist(bins = 25)
plt.xlabel('App Title Word Count')
plt.ylabel('Frequency')
plt.title('Distribution of App Title Word Count')
```

Out[277]: Text(0.5,1,'Distribution of App Title Word Count')



The plot above has a lot of empty space probably due to the presence of outliers. Let's confirm this.

In [278]:
```
# Print the max and min of 'Word Count'
minValue = df['Word Count'].min()
maxValue = df['Word Count'].max()
print(" The min is:", minValue, '\n', "The max is:", maxValue)
```

```
 The min is: 1
 The max is: 26
```

There is a title with 26 words!

Since it is generally accepted to do so, we will be removing all outliers 2 standard deviations away form the mean. By convention, values within two standard deviations of the mean (plus or minus) contain 95% of all the points.

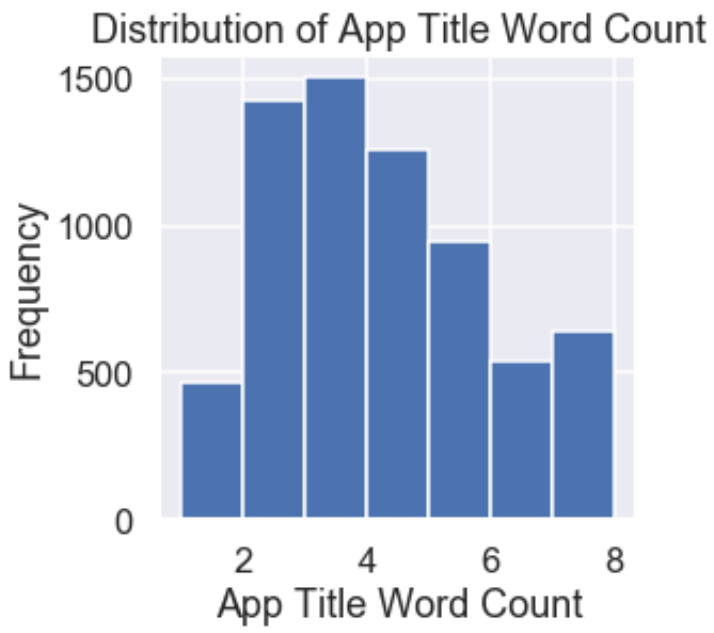
In [279]:
```
# We first store the values of the std and the mean.

wordCountSD = df['Word Count'].std()
wordCountMean = df['Word Count'].mean()

# We then remove outliers
df = df.drop(df[(df['Word Count'] < (wordCountMean - 2*wordCountSD)) |
                (df['Word Count'] > (wordCountMean + 2*wordCountSD))].index)
```

In [280]:
```
# Plots 'Word Count' of app titles to see its distribution
df['Word Count'].plot.hist(bins = 7, figsize=(4,4))
plt.xlabel('App Title Word Count')
plt.ylabel('Frequency')
plt.title('Distribution of App Title Word Count')
```

Out[280]: Text(0.5,1,'Distribution of App Title Word Count')



In [281]:
```
# Calculate the mean and median of 'Word Count'
mean = df['Word Count'].mean()
median = df['Word Count'].median()
print(" The mean is:", mean, '\n', "The median is:", median)
```

```
 The mean is: 3.7776302198612957
 The median is: 3.0
```

In [282]:
```
# Print the max and min of 'Word Count'
minValue = df['Word Count'].min()
maxValue = df['Word Count'].max()
print(" The min is:", minValue, '\n', "The max is:", maxValue)
```

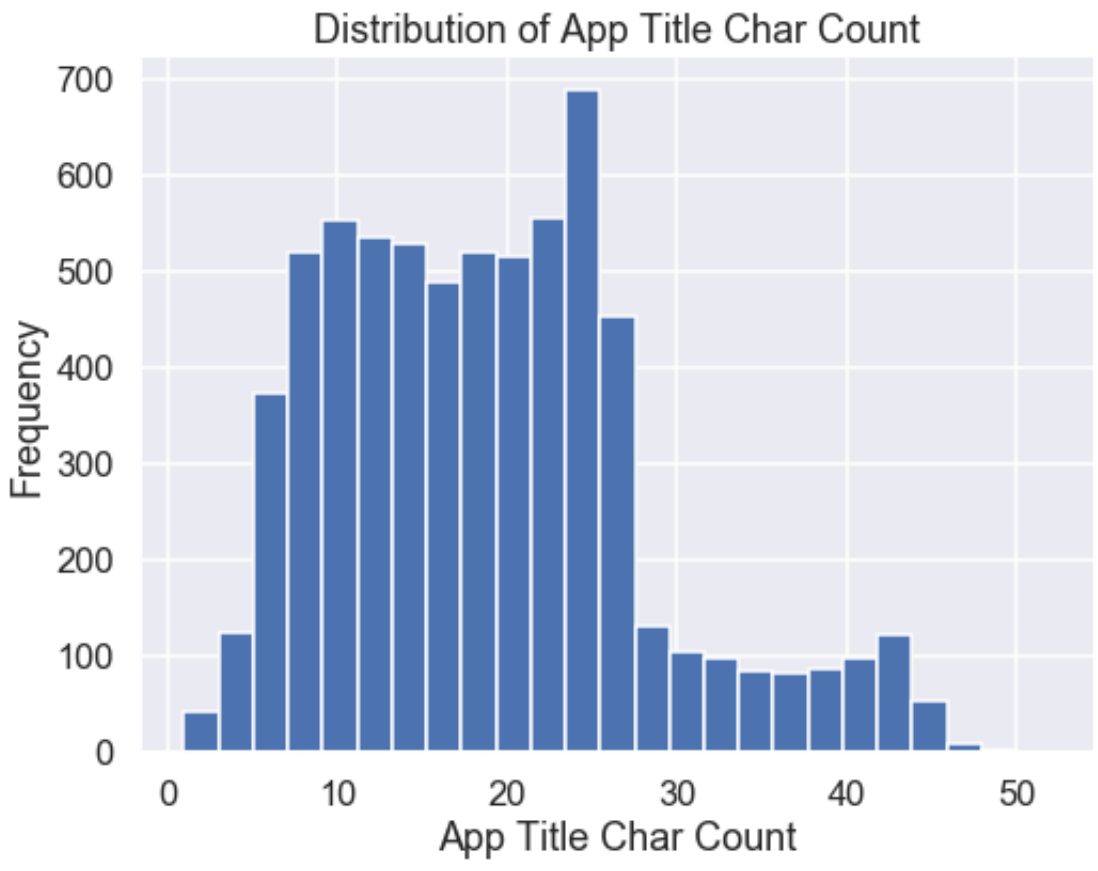
```
 The min is: 1
 The max is: 8
```

From the above histogram, we can say that most apps have relatively low word count (with a mean of ~3.79). Word count is ***slightly*** **skewed to the right**.

Char Count

In [283]:
```
# Plots 'Char Count' app titles to see its distribution
df['Char Count'].plot.hist(bins = 25, figsize=(8,6))
plt.xlabel('App Title Char Count')
plt.ylabel('Frequency')
plt.title('Distribution of App Title Char Count')
```

Out[283]: Text(0.5,1,'Distribution of App Title Char Count')



In [284]:
```
# Print the max and min of 'Char Count'
minValue = df['Char Count'].min()
maxValue = df['Char Count'].max()
print(" The min is:", minValue, '\n', "The max is:", maxValue)
```

```
 The min is: 1
 The max is: 52
```

In [285]:
```
# Calculate the mean and median of 'Char Count'
mean = df['Char Count'].mean()
median = df['Char Count'].median()
print(" The mean is:", mean, '\n', "The median is:", median)
```

```
 The mean is: 19.040283311199644
 The median is: 18.0
```

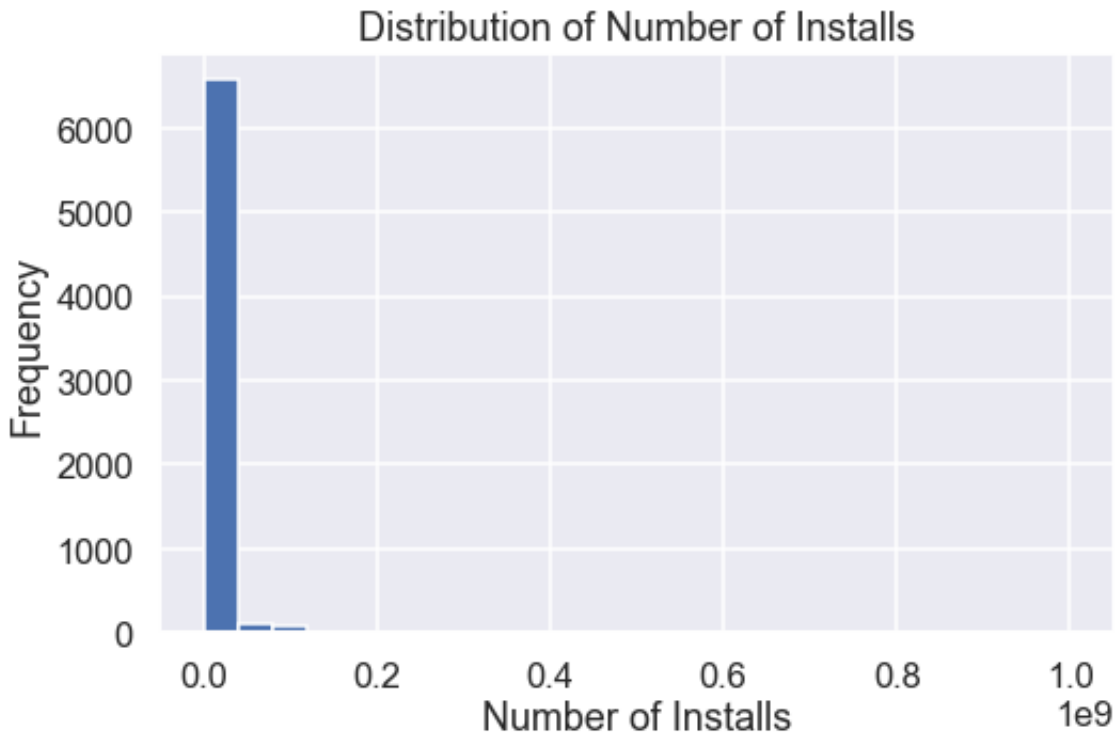
Char count has a mean of ~19.22.

By the histogram above, we can say that **char count is right skewed**.

Installs

In [286]:
```
# Plots 'Installs' to see its distribution
df['Installs'].plot.hist(bins = 25, figsize=(8,5))
plt.xlabel('Number of Installs')
plt.ylabel('Frequency')
plt.title('Distribution of Number of Installs')
```

Out[286]: Text(0.5,1,'Distribution of Number of Installs')



In [287]:
```
# Let's see how many unique values of installs we have
unique_installs = df['Installs'].unique()
unique_installs
```

Out[287]:
```
array([5.e+05, 5.e+07, 1.e+05, 5.e+04, 1.e+06, 1.e+04, 1.e+07, 5.e+03,
       5.e+06, 1.e+08, 1.e+03, 5.e+08, 1.e+02, 5.e+02, 1.e+01, 1.e+09,
       5.e+00, 5.e+01, 1.e+00])
```

In [288]:
```
# Print out the unique values of installs along with the number of apps that have each value
print("There are:", unique_installs.size, "unique installs.")
print('\n')
installs_size = df.groupby('Installs').size()
i = 0

for size in installs_size:
    print(unique_installs[i], "-->", size)
    i += 1
```

```
There are: 19 unique installs.


500000.0 --> 3
50000000.0 --> 9
100000.0 --> 66
50000.0 --> 56
1000000.0 --> 287
10000.0 --> 192
10000000.0 --> 662
5000.0 --> 401
5000000.0 --> 923
100000000.0 --> 418
1000.0 --> 943
500000000.0 --> 437
100.0 --> 1120
500.0 --> 431
10.0 --> 621
1000000000.0 --> 107
5.0 --> 91
50.0 --> 8
1.0 --> 2
```

As we can see above, the graph has many spaces in between the different intervals on the x-axis. This may be due to the fact that our data does not provide a comprehensive number of installs, but instead categegorizes them as above a certain threshold, like 10,000+ or 50,000+, and so on. We can see using .unique() that there are only 19 unique different values of installs.

We can also see that there are many apps in the beginning ranges of 0.0-0.1 1e8. As the number of installs increases to 0.5 1e8 or even 1 1e8, the number of apps with those installs begins to decrease.

In [289]:
```
# Print the max and min of 'Installs'
minValue = df['Installs'].min()
maxValue = df['Installs'].max()
print(" The min is:", minValue, '\n', "The max is:", maxValue)
```

```
 The min is: 1.0
 The max is: 1000000000.0
```

In [290]:
```
# Find out how many apps share the max value of 'Installs'
dfMax = df.loc[ df['Installs'] == df['Installs'].max() ]
dfMax
```

Out[290]:

	App	Rating	Installs	Price	...	zw	zx	zy	zz
1654	Subway Surfers	4.5	1.00e+09	0.0	...	0	0	0	0
3736	Google News	3.9	1.00e+09	0.0	...	0	0	0	0

2 rows × 159 columns

(There are 2 applications that share the same number of installs, the maximum value of installs of our dataset)

Unlike what we did with Word Count and Char Count, we decided **not** to remove outliers for installs. This is largely due to the fact that, as explained above, our dataset gives us the number of installs for apps in "plateus". By removing outliers, we will be getting rid of the applications that are extremely successful(such as Facebook, Snapchat, and Instagram). It is important to note that making this decision either way (deciding that we are getting rid of these outliers or keeping them) is inherently biased.

In [291]:
```
# Calculate the mean and median of 'Installs'
mean = df['Installs'].mean()
median = df['Installs'].median()
print(" The mean is:", mean, '\n', "The median is:", median)
```

```
 The mean is: 4468160.721263096
 The median is: 100000.0
```

Installs has a mean of ~9272736.74.

From the above histogram, it looks like installs is strongly **skewed right**.

Reviews

We know that installs is a good indicator of the success of an app, however, the install data that was scraped from the playstore is somewhat categorical, (grouped in increments). Therefore, we believe that moving toward a continuous variable such as number of reviews would tell a better story and thus find better correlations between other independent variables and the success of the app. If we could find a correlation between Reviews and Installs, we will rather use Reviews as an indicator of success rather than Installs, since it is a continuous variable.

In [292]:
```
df['Reviews'].plot(kind = 'hist', bins = 25, figsize = (7, 5))
plt.xlabel('App Reviews')
plt.ylabel('Frequency')
plt.title('Distribution of App Reviews')
```

Out[292]: Text(0.5,1,'Distribution of App Reviews')

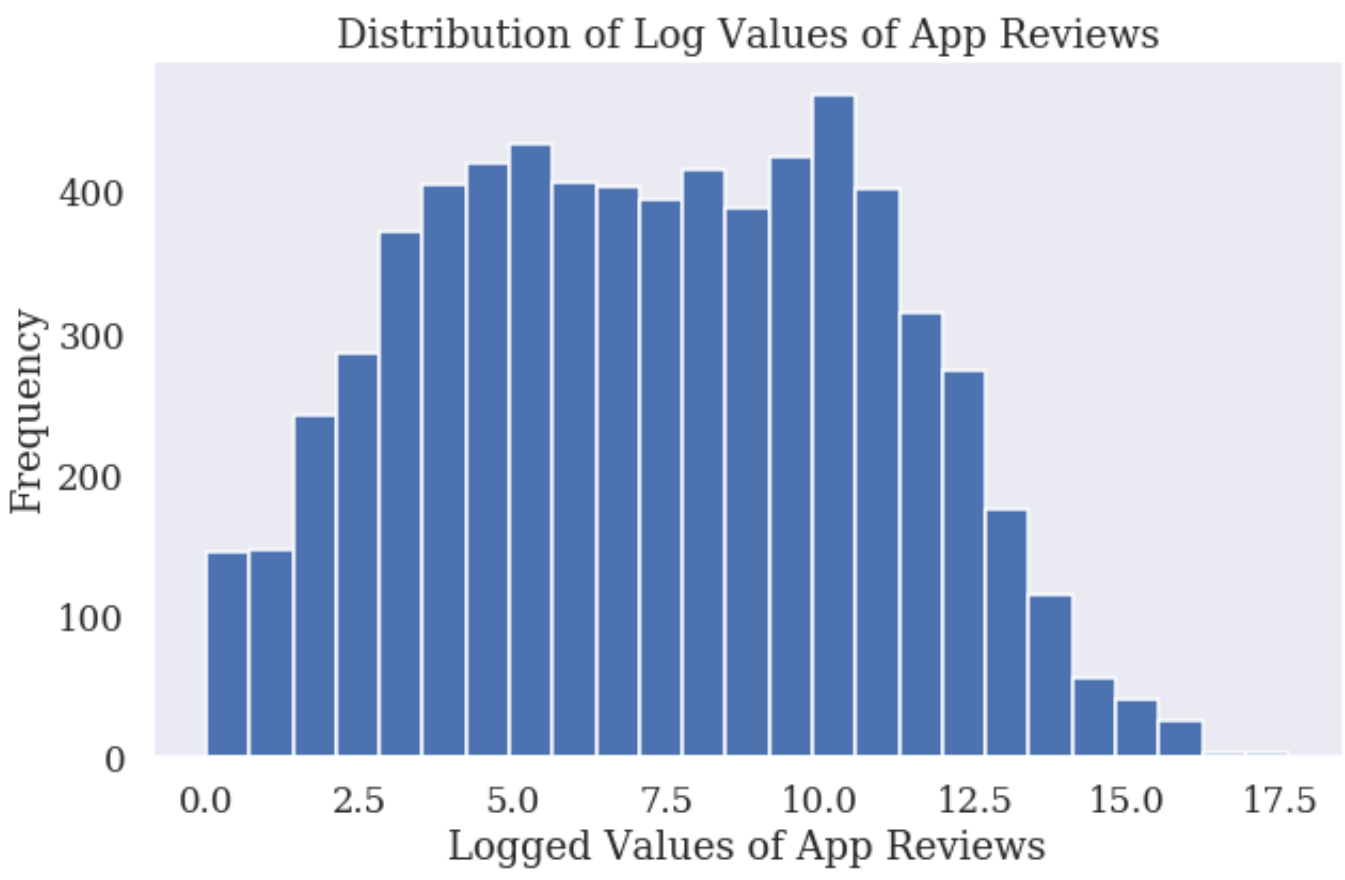
As we can see, most of the reviews cluster around 0. Logging these values might help us better visualize the distribution. Logging the graph minimizes the effect of outliers, due to the fact that log10(100) = 2 and log10(1000) = 3. Large outliers no longer drive all the variation in the distribution, and we can start to visualize differences.

Final Visualization #1

In [293]:
```
# Visualization customization
plt.rcParams["font.family"] = "Serif"
plt.rcParams["font.size"] = "10"
plt.rcParams['figure.titlesize'] = "small"
```

In [294]:
```
df['log_value_Reviews'] = np.log(df['Reviews'])
df['log_value_Reviews'].plot(kind = 'hist', bins = 25, figsize = (10,6))
plt.grid(None)
plt.xlabel('Logged Values of App Reviews')
plt.ylabel('Frequency')
plt.title('Distribution of Log Values of App Reviews')
```

Out[294]: Text(0.5,1,'Distribution of Log Values of App Reviews')



As we can see from the histogram, the data for number of Reviews appears to have a **normal distribution**. Normally distributed data means that the data makes a symmetric bell graph. Our graph above is close to normal distribution, meaning about 50% of the data falls on one side of the mean and 50% falls on the other. The fact that this data is symmetrical and not as extremely skewed as our other variables makes it an appropriate choice for our dependent variable.

We logged the values of app reviews for the same reasons as mentioned earlier, to decrease the effect large outliers have on our visualization.

In [295]:
```
# Print the max and min of 'Reviews'
minValue = df['Reviews'].min()
maxValue = df['Reviews'].max()
print(" The min # Reviews is:", minValue, '\n', "The max # Reviews is:", maxValue)
```

```
 The min # Reviews is: 1.0
 The max # Reviews is: 44891723.0
```

In [296]:
```
# Calculate the mean and median of 'Reviews'
mean = df['Reviews'].mean()
median = df['Reviews'].median()
print(" The mean # Reviews is:", mean, '\n', "The median # Reviews is:", median)
```

```
 The mean # Reviews is: 143632.7518075845
 The median # Reviews is: 1468.0
```

Secondary independent variables

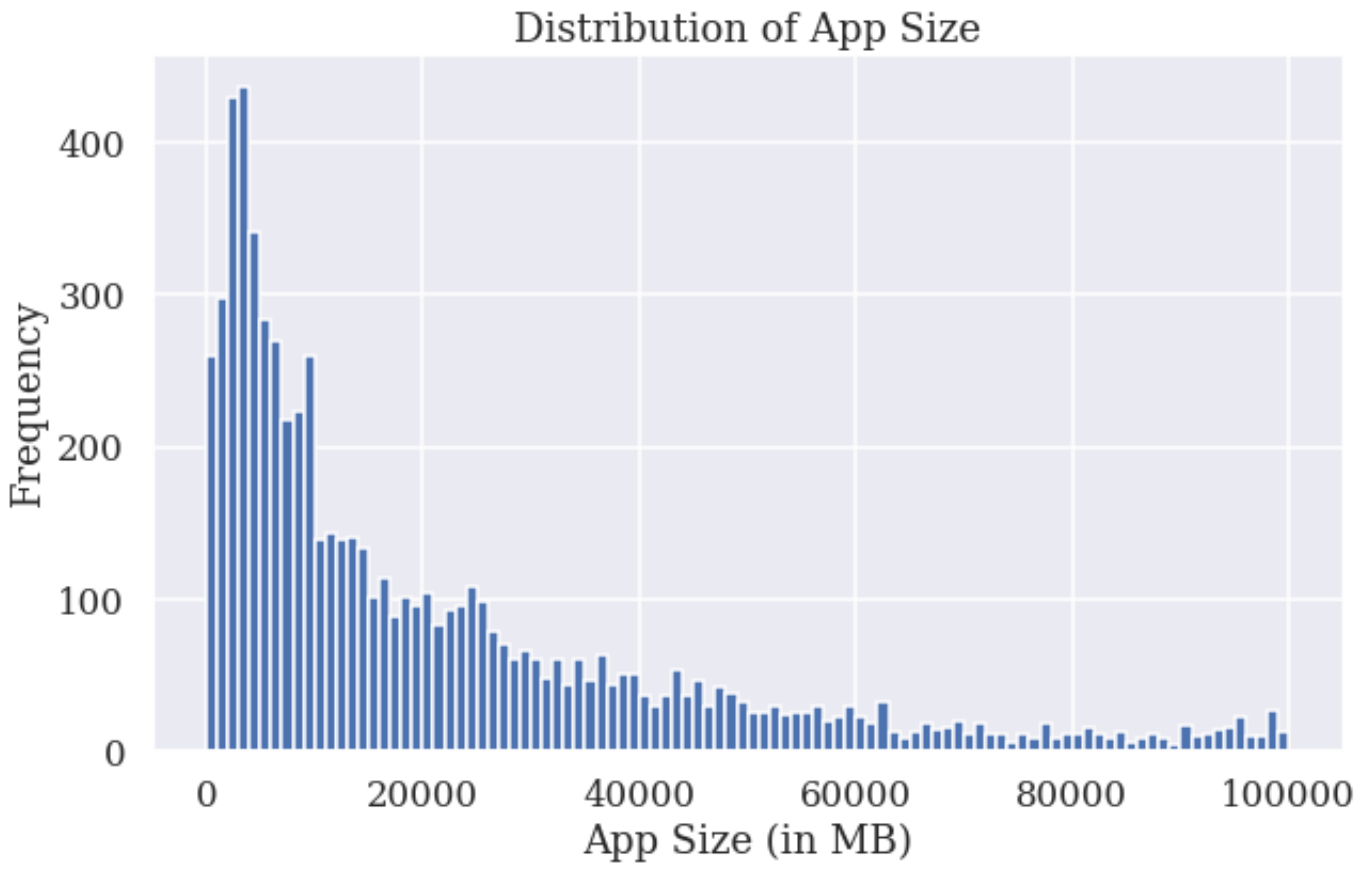
Size

In [297]:
```
# Visualization customization
plt.rcParams["font.family"] = "Serif"
plt.rcParams["font.size"] = "13"
plt.rcParams['figure.titlesize'] = "small"
```

Final Visualization #2

In [298]:
```
df['Size'].plot(kind = 'hist', bins = 100, figsize = (10, 6))
plt.xlabel('App Size (in MB)')
plt.ylabel('Frequency')
plt.title('Distribution of App Size')
```

Out[298]: Text(0.5,1,'Distribution of App Size')



As we can see from the histogram, the data for Size is **heavily right skewed**. This means that median would be a better measure of central tendency. We note that there are an abundance of applications with smaller sizes.

In [299]:
```
# Print the max and min of 'Size'
minValue = df['Size'].min()
maxValue = df['Size'].max()
print(" The min Size is:", minValue, '\n', "The max Size is:", maxValue)
```

```
 The min Size is: 8.5
 The max Size is: 100000.0
```

In [300]:
```
# Print the mean and median of 'Size'
meanValue = df['Size'].mean()
medianValue = df['Size'].median()
print(" The mean Size is:", meanValue, '\n', "The median Size is:", medianValue)
```

```
 The mean Size is: 21686.891028478676
 The median Size is: 13000.0
```

Price

In [301]:
```
# Visualization customization
plt.rcParams["font.family"] = "Sans"
```

In [302]:
```
df['Price'].plot(kind = 'hist', figsize = (7, 5))
plt.xlabel('App Price')
plt.ylabel('Frequency')
plt.title('Distribution of App Price')
```

Out[302]: Text(0.5,1,'Distribution of App Price')

In [303]:
```
# Print the max and min of 'Price'
minValue = df['Price'].min()
maxValue = df['Price'].max()
print(" The min Price is:", minValue, '\n', "The max Price is:", maxValue)
```

```
The min Price is: 0.0
The max Price is: 400.0
```

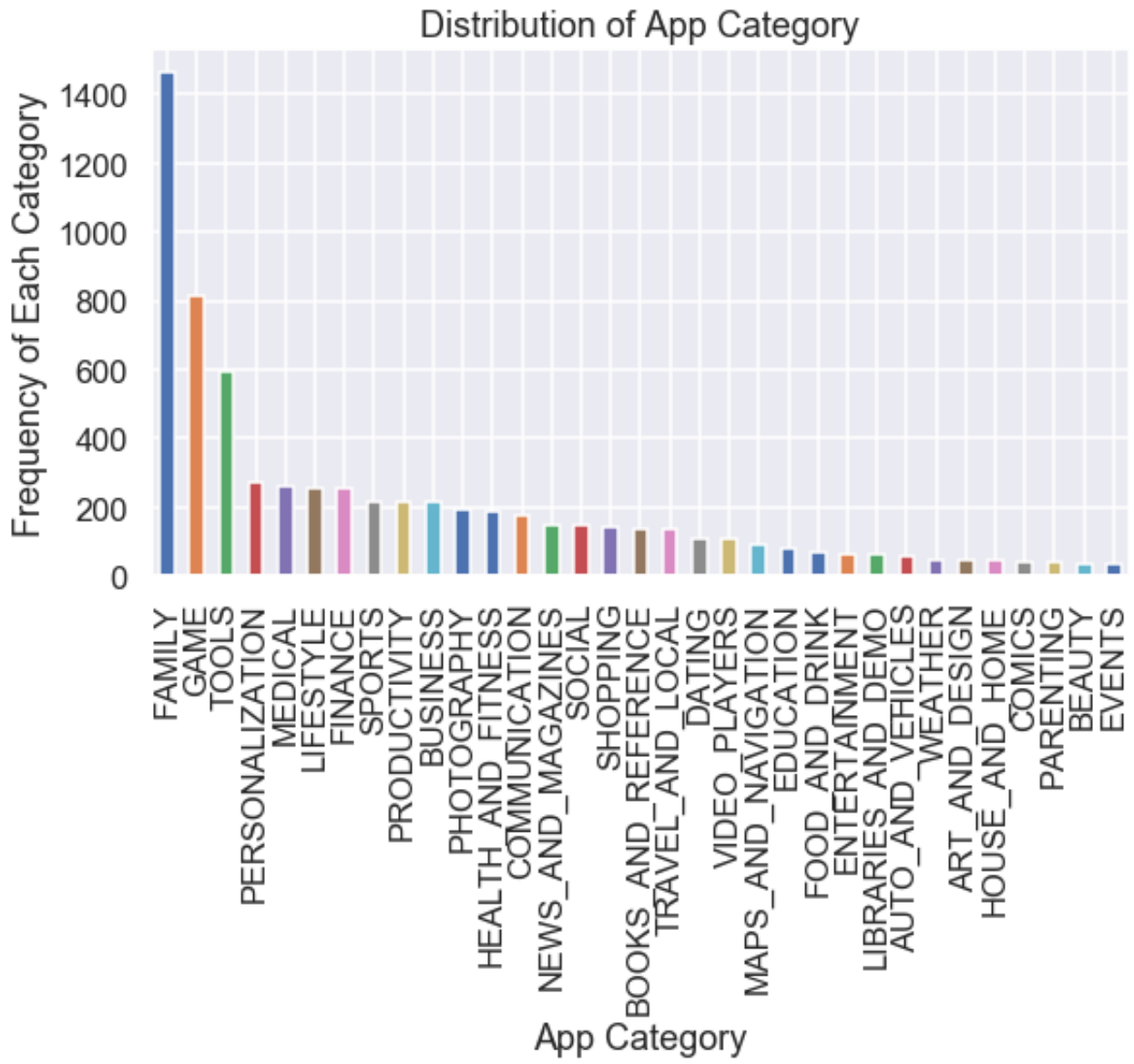
In [304]:
```
# Print the mean and median of 'Price'
meanValue = df['Price'].mean()
medianValue = df['Price'].median()
print(" The mean Price is:", meanValue, '\n', "The median Price is:", medianValue)
```

```
The mean Price is: 1.2103939796369962
The median Price is: 0.0
```

Category

In [305]:
```
df['Category'].value_counts().plot(kind = 'bar', figsize = (9, 5))
plt.xlabel('App Category')
plt.ylabel('Frequency of Each Category')
plt.title('Distribution of App Category')
```

Out[305]: Text(0.5,1,'Distribution of App Category')



As we can see from the plot above, the most frequent app category is 'Family', with 'Game' behind it.

Let's try to explore some relationships between our variables.

In [306]:
```
# Calculates Pearson correlation coefficient between variables with all observations
df.iloc[:,1:6].corr()
```

Out[306]:

	Rating	Installs	Price	Size	Reviews
Rating	1.00	0.05	-2.06e-02	0.06	6.76e-02
Installs	0.05	1.00	-1.06e-02	0.13	5.95e-01
Price	-0.02	-0.01	1.00e+00	-0.03	-9.02e-03
Size	0.06	0.13	-2.59e-02	1.00	1.86e-01
Reviews	0.07	0.60	-9.02e-03	0.19	1.00e+00

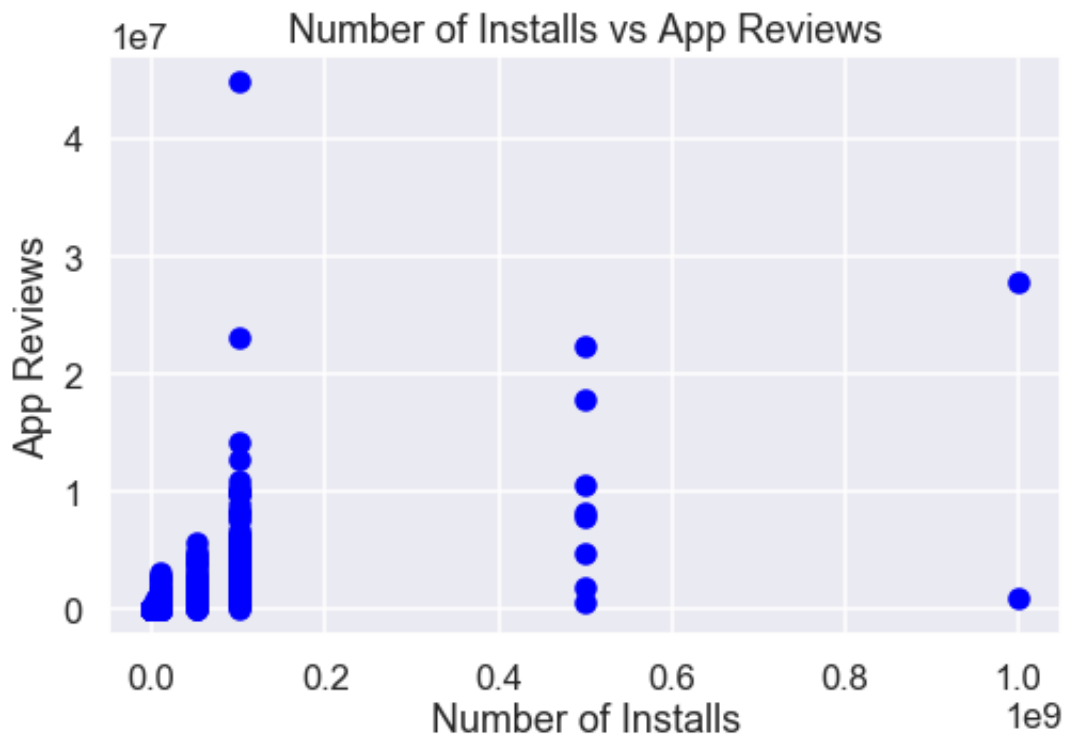
Save for the obvious relationship between word count and char count, there is little correlation between the variables based on Pearson's value.

Is there a relationship between *Reviews and Installs*?

In order to justify the use of # of Reviews as a measure of success, we first have to see if there is a valid relationship between it and Installs.

In [307]:
```
x = df['Installs']
y = df['Reviews']
fig = plt.figure(figsize=(8, 5))
plt.scatter(x, y, c='blue')
plt.xlabel('Number of Installs')
plt.ylabel('App Reviews')
plt.title('Number of Installs vs App Reviews')
```

Out[307]: Text(0.5,1,'Number of Installs vs App Reviews')



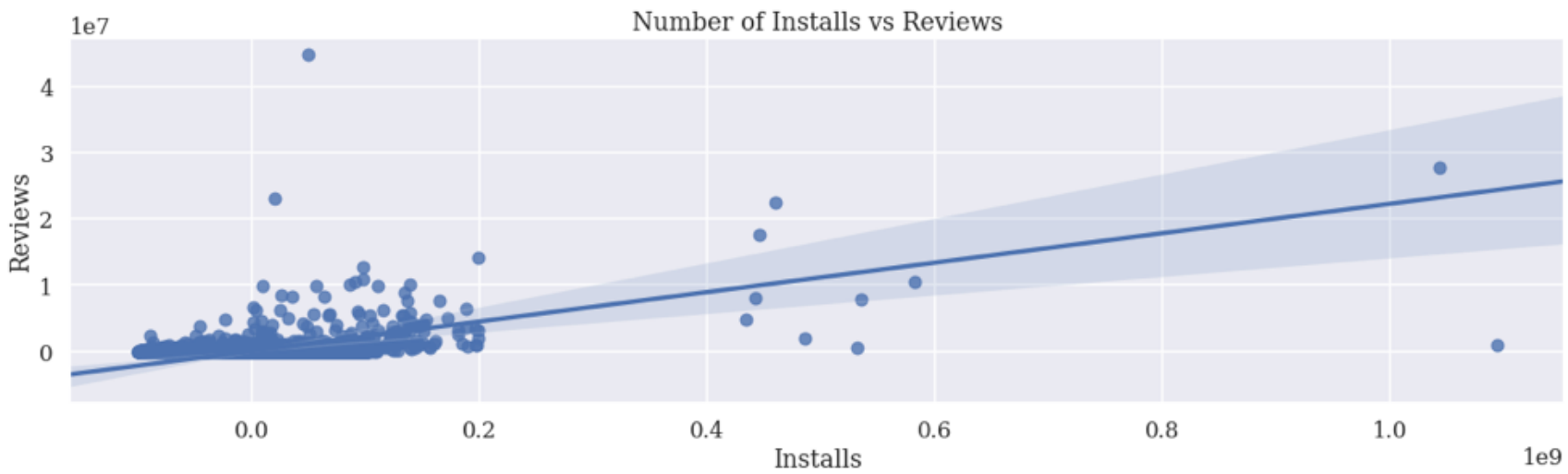
The above scatterplot is pretty hard to interpret, so we will use some jittering to better visualize the graph.

Final Visualzation #3

In [308]:
```
# Visualization customization
plt.rcParams["font.family"] = "Serif"
plt.rcParams["font.size"] = "20"
plt.rcParams['figure.titlesize'] = "small"
```

In [309]:
```
fig = plt.figure(figsize=(20, 5))
sns.regplot(x='Installs', y='Reviews', data=df, x_jitter=100000000)
plt.title('Number of Installs vs Reviews')
```

Out[309]: Text(0.5,1,'Number of Installs vs Reviews')



As we can see, there is a pretty clear positive relationship between reviews and installs. We can see this from the positively-sloped simple linear regression model from regplot() and the positions of the scatter points relative to the model. Thus, a greater number of installs means a greater number of reviews. How correlated are these values?

In [310]:
```
df[['Reviews', 'Installs']].corr()
```

Out[310]:

	Reviews	Installs
Reviews	1.0	0.6
Installs	0.6	1.0

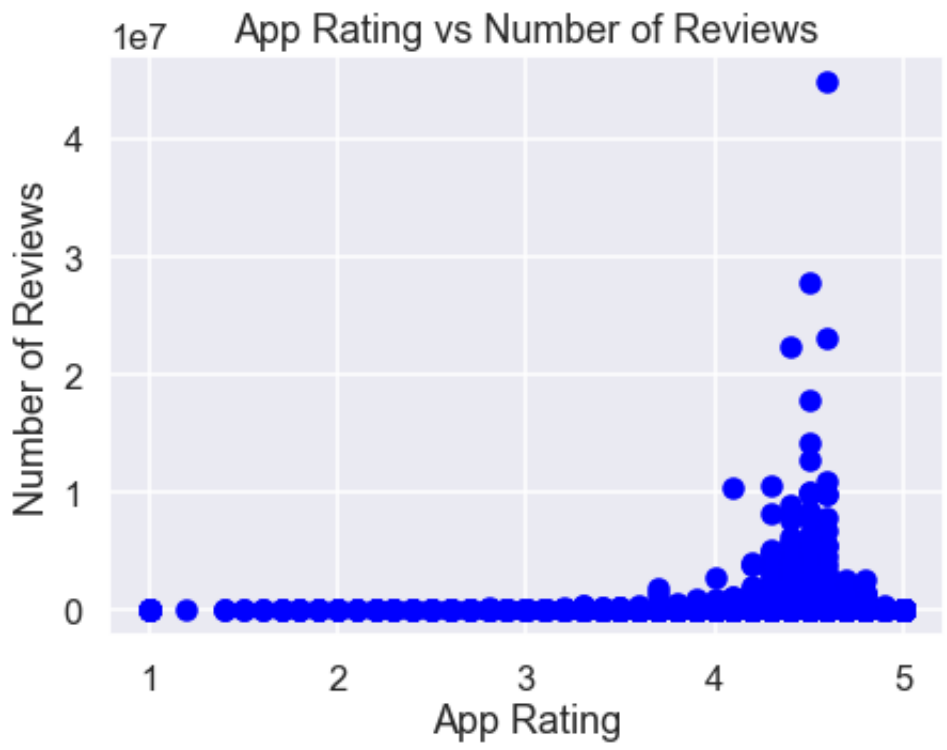
The correlation coefficient between reviews and installs is 0.6. This is a moderately strong positive linear relationship. We believe that 0.6 is a strong enough correlation coefficient to prove that reviews is a good indicator of installs, since this is not even accounting for the variability within the bins of installs. Therefore, we will be using reviews as a measure of success rather than installs.

Is there any relationship between *ratings* and *reviews*?

In [311]:
```
# Visualization customization
plt.rcParams["font.family"] = "Sans"
```

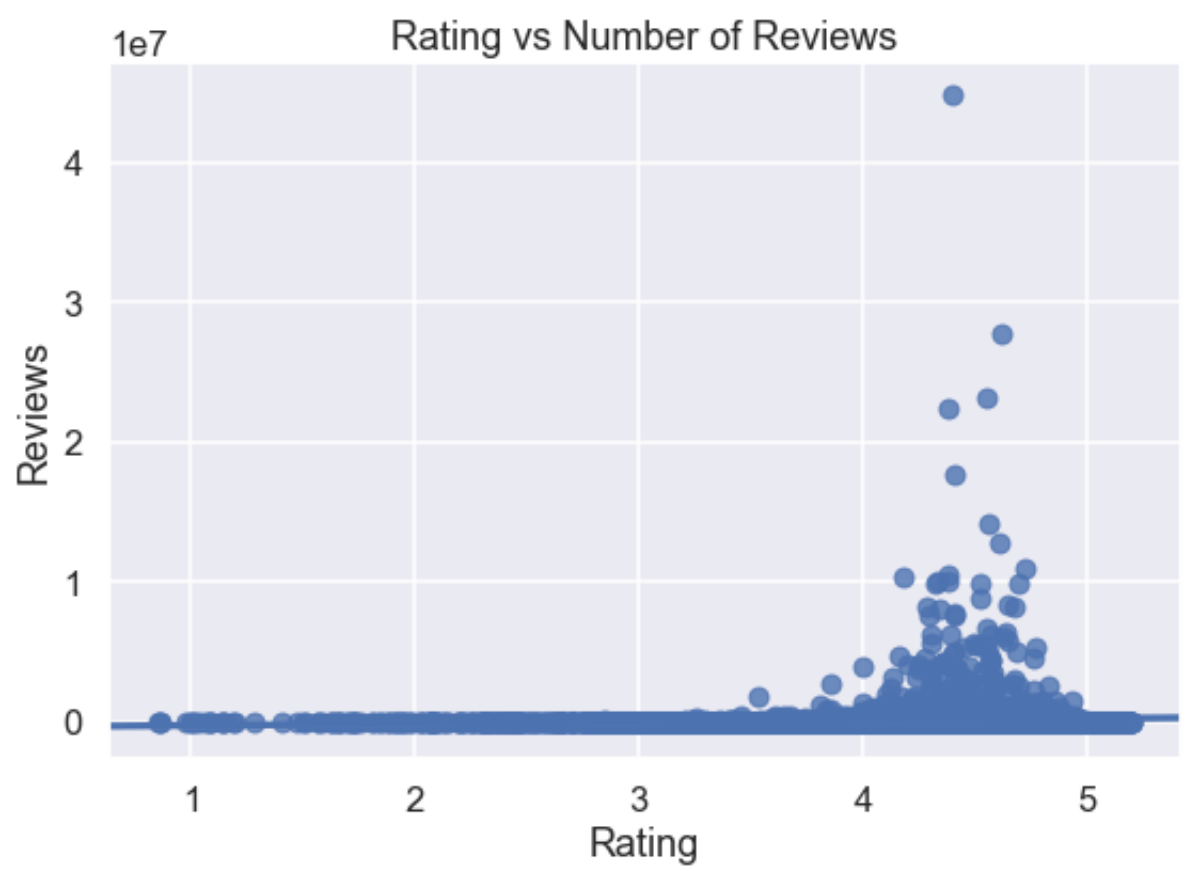
In [312]:
```
x = df['Rating']
y = df['Reviews']
fig = plt.figure(figsize=(7, 5))
plt.scatter(x, y, c='blue')
plt.xlabel('App Rating')
plt.ylabel('Number of Reviews')
plt.title('App Rating vs Number of Reviews')
```

Out[312]: Text(0.5,1,'App Rating vs Number of Reviews')



In [313]:
```
# Use jittering in order to better visualize the above scatter plot
fig = plt.figure(figsize=(9, 6))
sns.regplot(x='Rating', y='Reviews', data=df, x_jitter=0.2, y_jitter=100)
plt.title('Rating vs Number of Reviews')
```

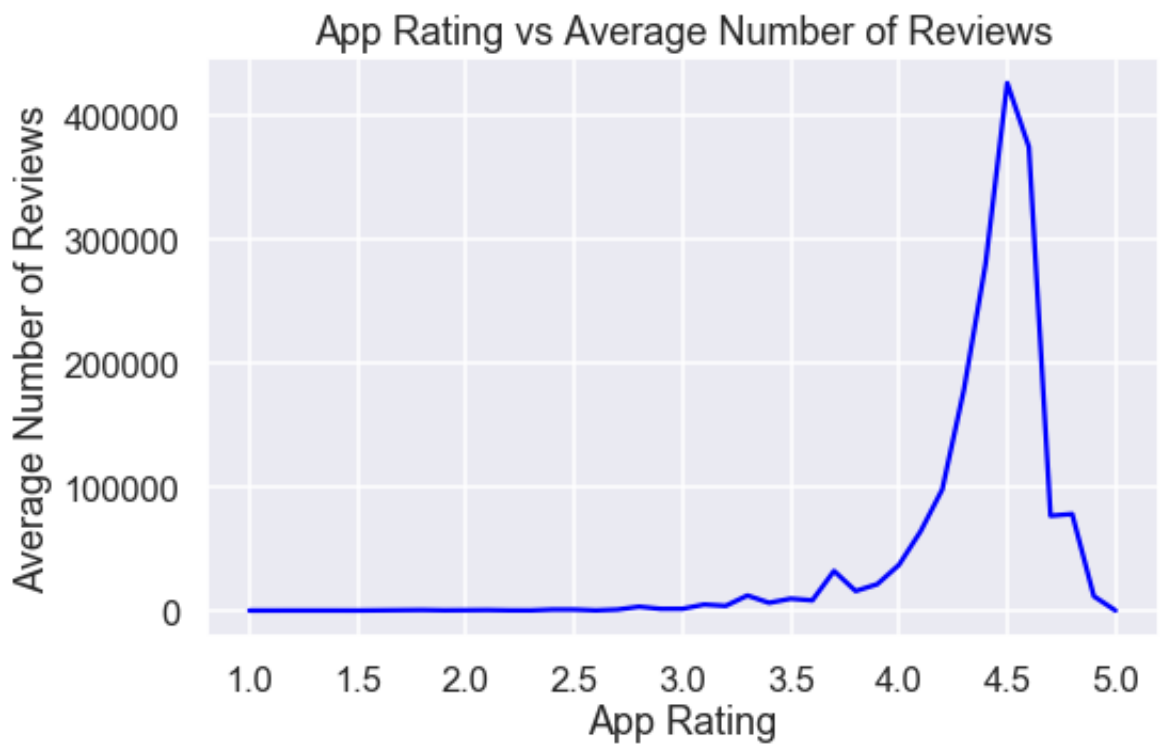
Out[313]: Text(0.5,1,'Rating vs Number of Reviews')



With installs on the y axis and ratings on the x axis, we can see that the graphs above present a trend. Although not a perfect negative skew, we can see that apps with ratings from 3-5 have a slightly higher number of installs than apps with ratings from 0-2.5. Similar to the installs graph, the flat lines are due to the way the number of installs was categorized into ranges/buckets of values above a certain threshold, like 10,000+ or 50,000+. Therefore, we can observe the lines at 0.5 1e8 and 1 1e8, which mean that some apps with ratings 3-5 had that number of installs.

In [314]:
```
# Plots ratings on the x-axis and average number of installs on the y-axis
df_RvsI = df.groupby('Rating').mean()
x = df_RvsI.index
y = df_RvsI['Reviews']
fig = plt.figure(figsize=(8, 5))
plt.plot(x, y, c='blue')
plt.xlabel('App Rating')
plt.ylabel('Average Number of Reviews')
plt.title('App Rating vs Average Number of Reviews')
```

Out[314]:
```
Text(0.5,1,'App Rating vs Average Number of Reviews')
```



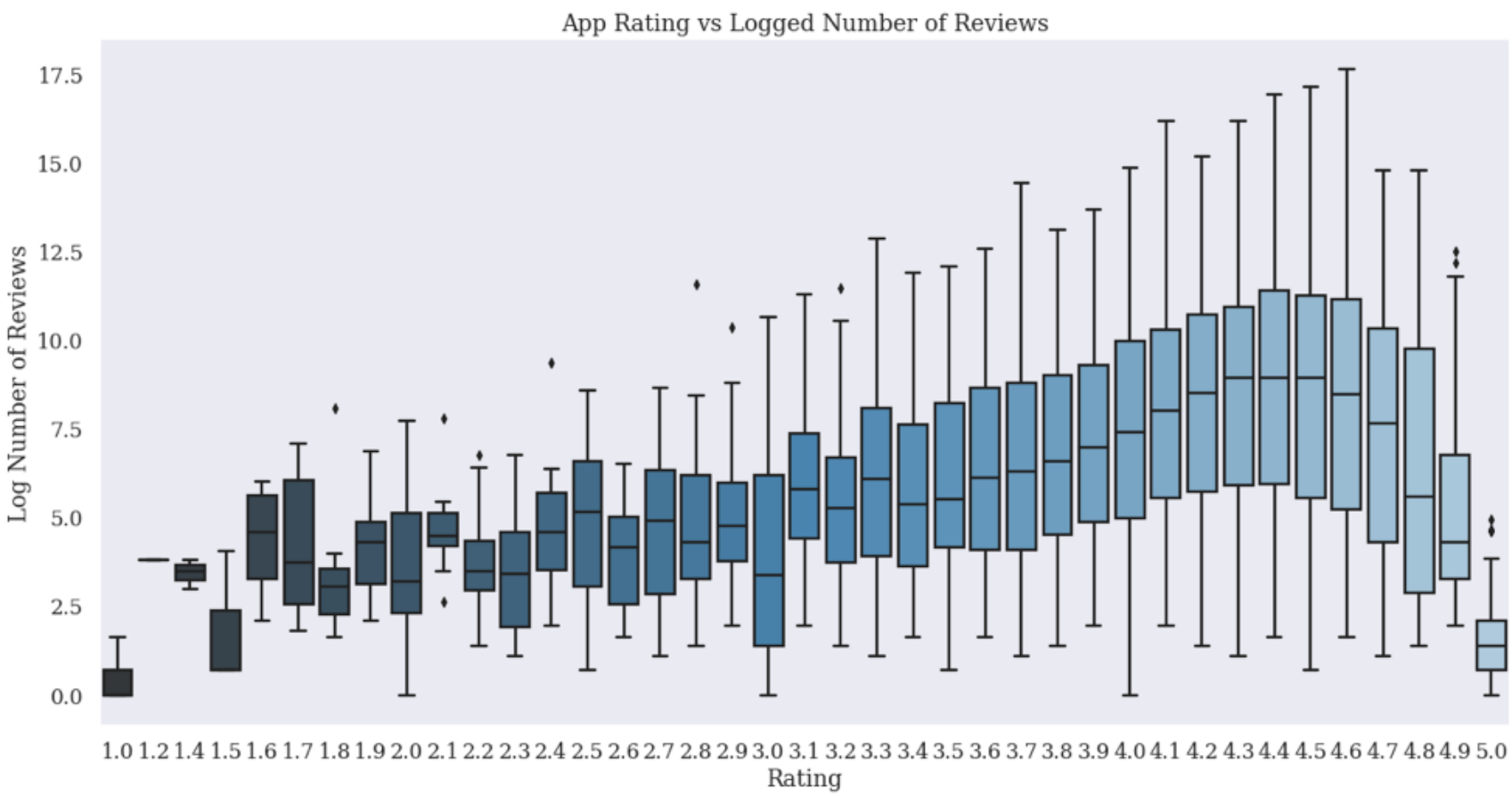
As discussed before, we will log the values for the number of Installs to better visualize the data.

In [315]:
```
# Visualization customization
plt.rcParams["font.family"] = "Serif"
```

Final Visualzation #4

In [316]:
```
# Box-plots ratings on the x-axis and log value of Reviews on the y-axis
fig = plt.figure(figsize=(20, 10))
plt.grid(None)
sns.boxplot(x='Rating', y='log_value_Reviews', data=df, palette=("Blues_d"),)
ax = plt.gca()
plt.ylabel('Log Number of Reviews')
plt.title('App Rating vs Logged Number of Reviews')
```

Out[316]: Text(0.5,1,'App Rating vs Logged Number of Reviews')



The visualization above is **heavily skewed left**. Apps with higher ratings tend to have a higher number of reviews.

From the above four graphs, we can say that there is strong enough relationship between ratings and reviews. As ratings increase, there is a general trend that number of reviews also increases.

Due to this relationship, and the fact that rating as a variable does not vary much (most ratings are between 4 to 5 as demonstrated earlier: with a mean of ~4.17 and a median of 4.3), we decided to utilize only ratings as a measure of success for our project.

We want to see if there is any trend/correlation between *word count* and *reviews*.

In [317]:
```
# Groups apps by the number of words they have, then gets the average number of reviews and displ
ays them
df_new = df.groupby('Word Count').mean()
df_new
```

Out[317]:

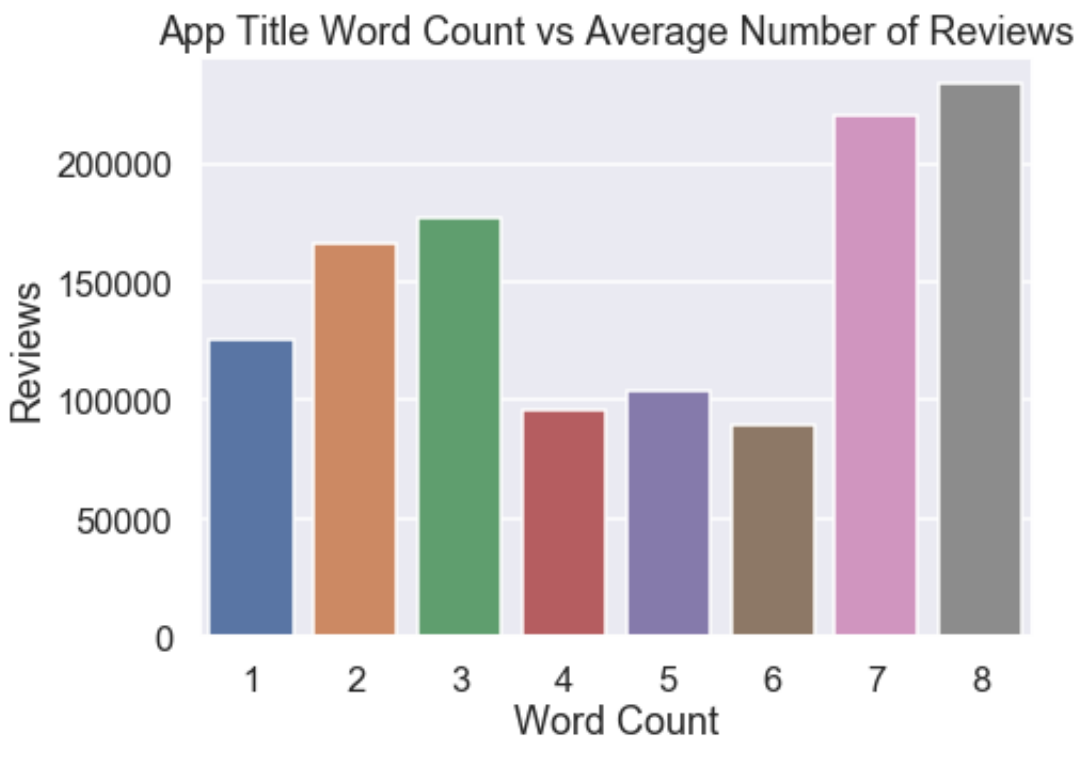
	Rating	Installs	Price	Size	...	zx	zy	zz	log_value_Reviews
Word Count									
1	4.10	6.16e+06	0.18	19551.93	...	0.0	0.00e+00	0.00e+00	7.47
2	4.07	5.60e+06	0.24	21060.65	...	0.0	2.11e-03	8.42e-03	6.74
3	4.12	4.06e+06	2.04	21828.56	...	0.0	2.00e-03	2.66e-03	7.15
...	...	...	...	...	...	...	...	...	...
6	4.25	3.91e+06	0.87	24353.76	...	0.0	3.71e-03	1.48e-02	7.94
7	4.28	4.66e+06	0.24	23646.12	...	0.0	5.52e-03	1.66e-02	7.73
8	4.27	6.98e+06	0.27	23239.11	...	0.0	3.55e-03	7.09e-03	7.92

8 rows × 157 columns

In [318]:
```
# Visualization customization
plt.rcParams["font.family"] = "Sans"
```

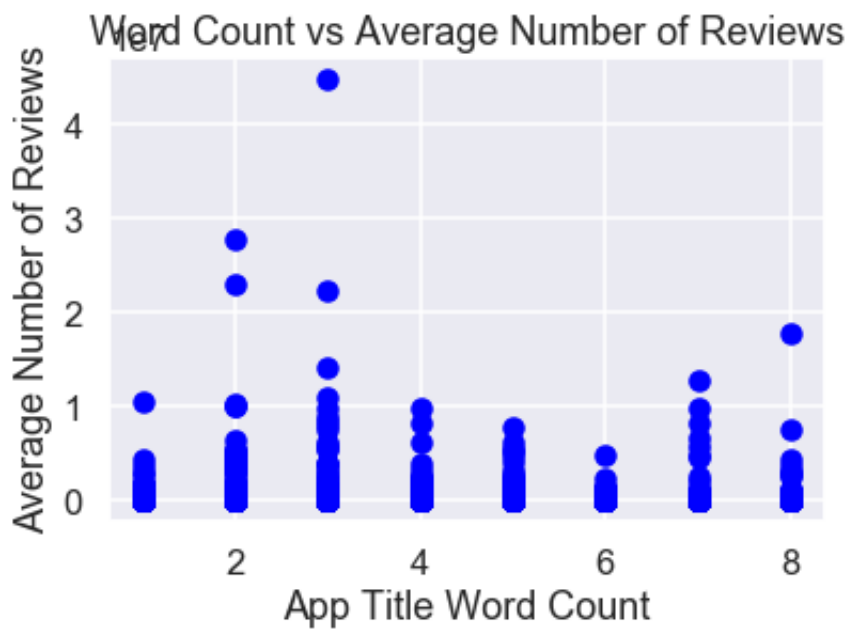
In [319]:
```
fig = plt.figure(figsize=(7, 5))
# Plots the average number of reviews (y-axis) for apps of each word count (x-axis)
x1 = df_new.index
sns.barplot(x=x1,
            y='Reviews',
            data=df_new);
plt.title('App Title Word Count vs Average Number of Reviews')
```

Out[319]: Text(0.5,1,'App Title Word Count vs Average Number of Reviews')



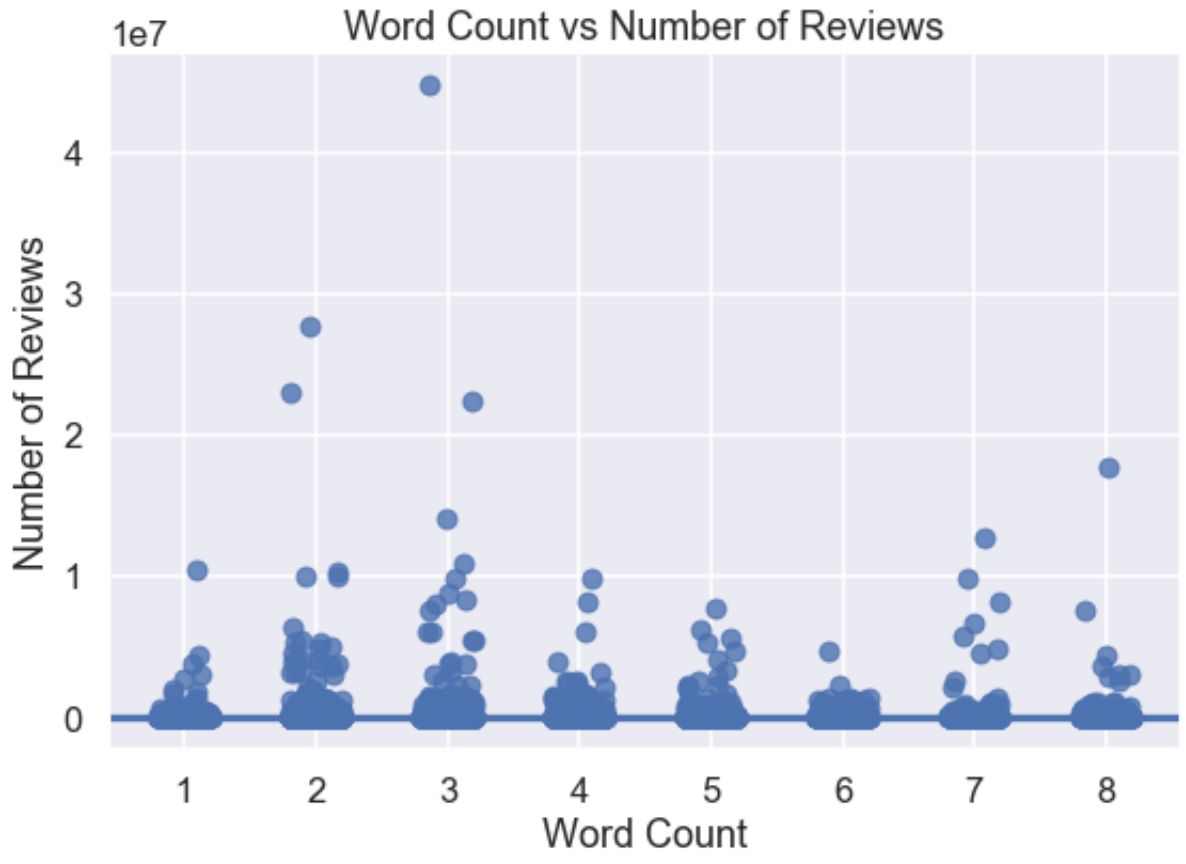
In [320]:
```
x = df['Word Count']
y = df['Reviews']
plt.scatter(x, y, c='blue')
plt.xlabel('App Title Word Count')
plt.ylabel('Average Number of Reviews')
plt.title('Word Count vs Average Number of Reviews')
```

Out[320]: Text(0.5,1,'Word Count vs Average Number of Reviews')



In [321]:
```
fig = plt.figure(figsize=(9, 6))
sns.regplot(x='Word Count', y='Reviews', data=df, x_jitter=0.2, y_jitter=1000)
plt.xlabel('Word Count')
plt.ylabel('Number of Reviews')
plt.title('Word Count vs Number of Reviews')
```

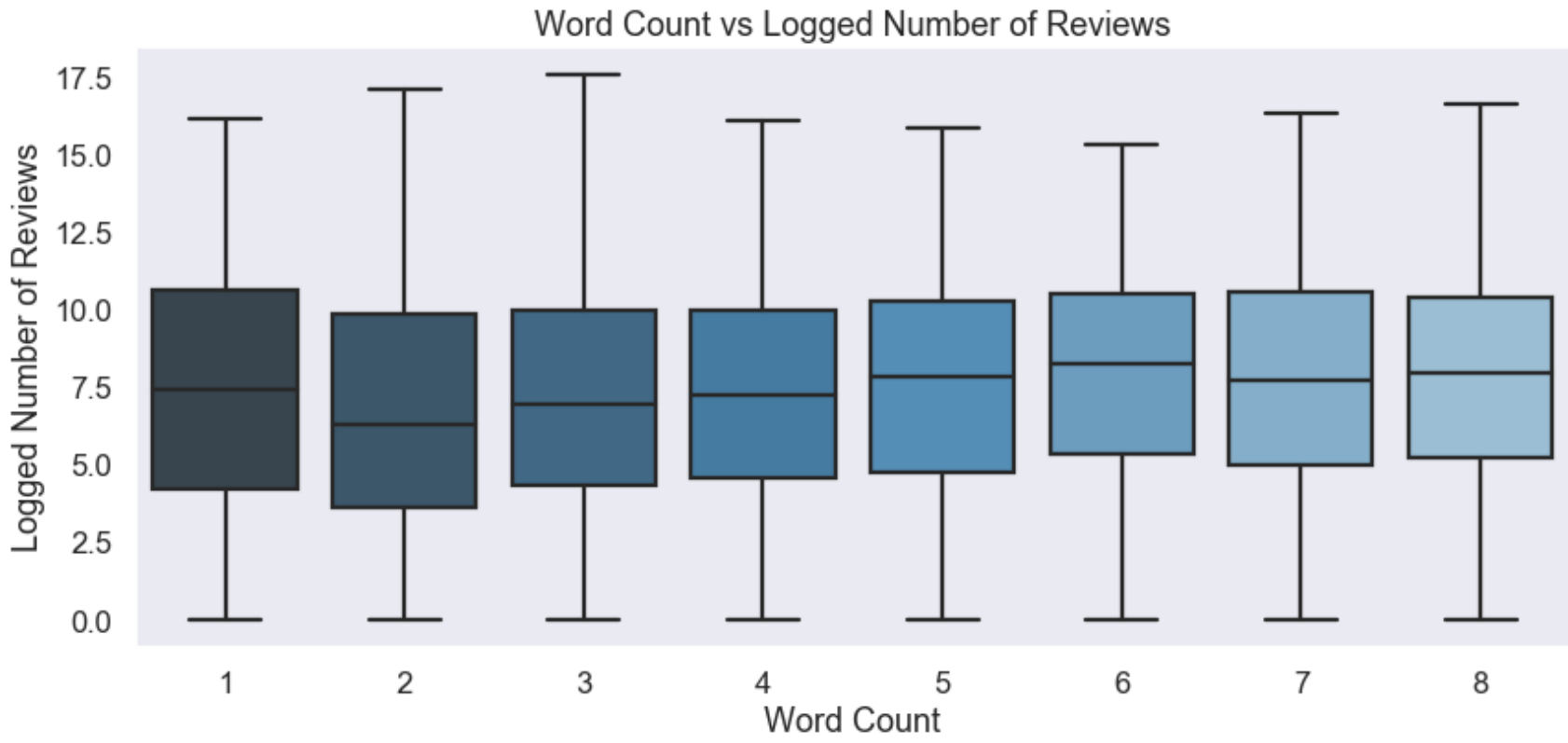
Out[321]: Text(0.5,1,'Word Count vs Number of Reviews')



In [322]:
```
##### make pretty ######

fig = plt.figure(figsize=(14, 6))
plt.grid(None)
sns.boxplot(x='Word Count', y='log_value_Reviews', palette=("Blues_d"), data=df)
ax = plt.gca()
plt.xlabel('Word Count')
plt.ylabel('Logged Number of Reviews')
plt.title('Word Count vs Logged Number of Reviews')
```

Out[322]: Text(0.5,1,'Word Count vs Logged Number of Reviews')



Similarly, what about *char count* and *reviews*?

In [323]:
```
# Groups apps by the number of words they have, then gets the average number of reviews and displ
ays them
df_new = df.groupby('Char Count').mean()
df_new
```

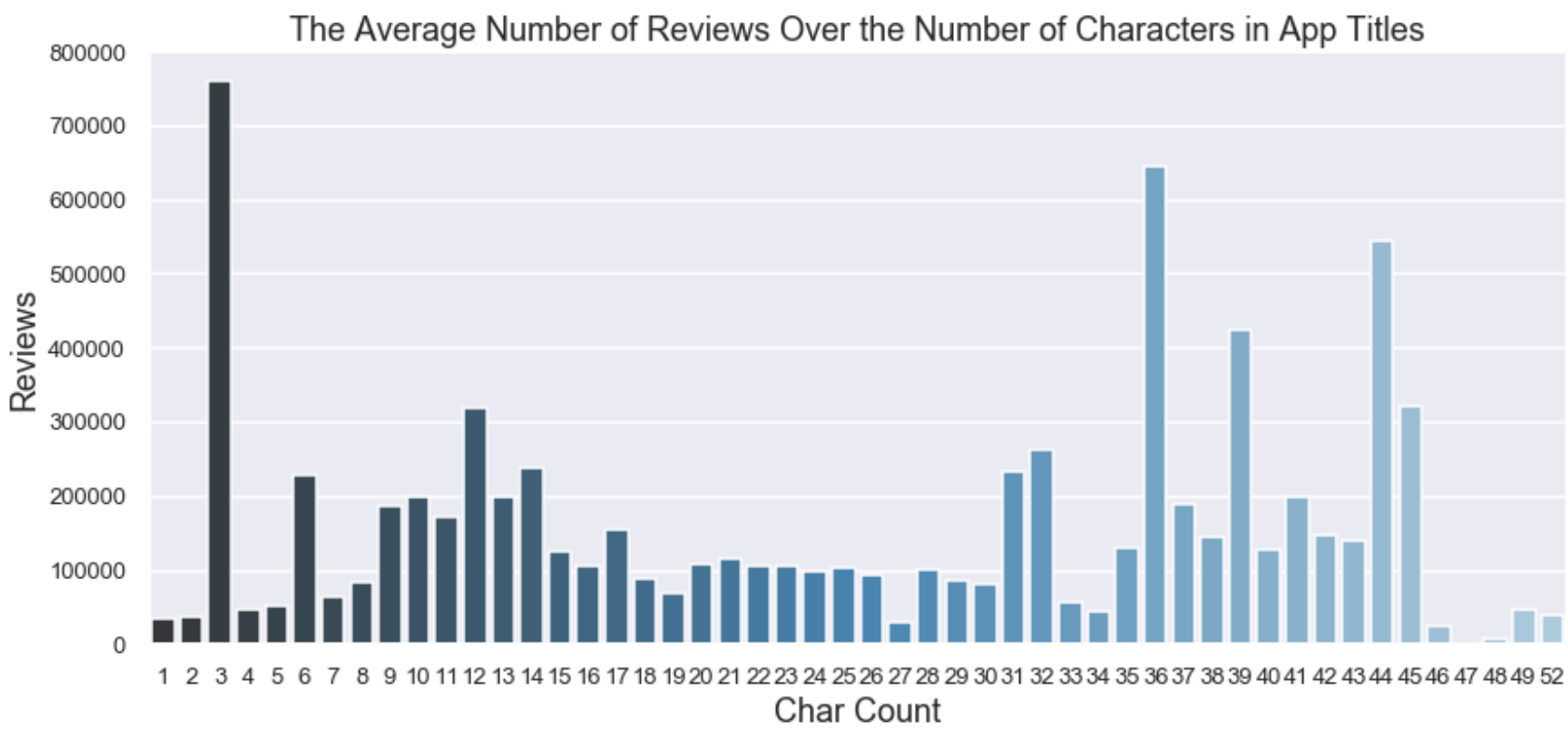
Out[323]:

	Rating	Installs	Price	Size	...	zx	zy	zz	log_value_Reviews
Char Count									
1	3.95	5.25e+05	0.00	46000.00	...	0.0	0.0	0.0	8.91
2	3.94	2.20e+06	0.04	9768.12	...	0.0	0.0	0.0	5.55
3	3.97	3.53e+07	0.00	21750.00	...	0.0	0.0	0.0	10.32
...	...	...	...	...	...	...	...	...	...
48	4.50	1.00e+05	0.00	13000.00	...	0.0	0.0	0.0	8.94
49	3.70	2.75e+06	0.00	10550.00	...	0.0	0.0	0.0	9.64
52	4.30	1.00e+06	0.00	8600.00	...	0.0	0.0	0.0	10.57

50 rows × 157 columns

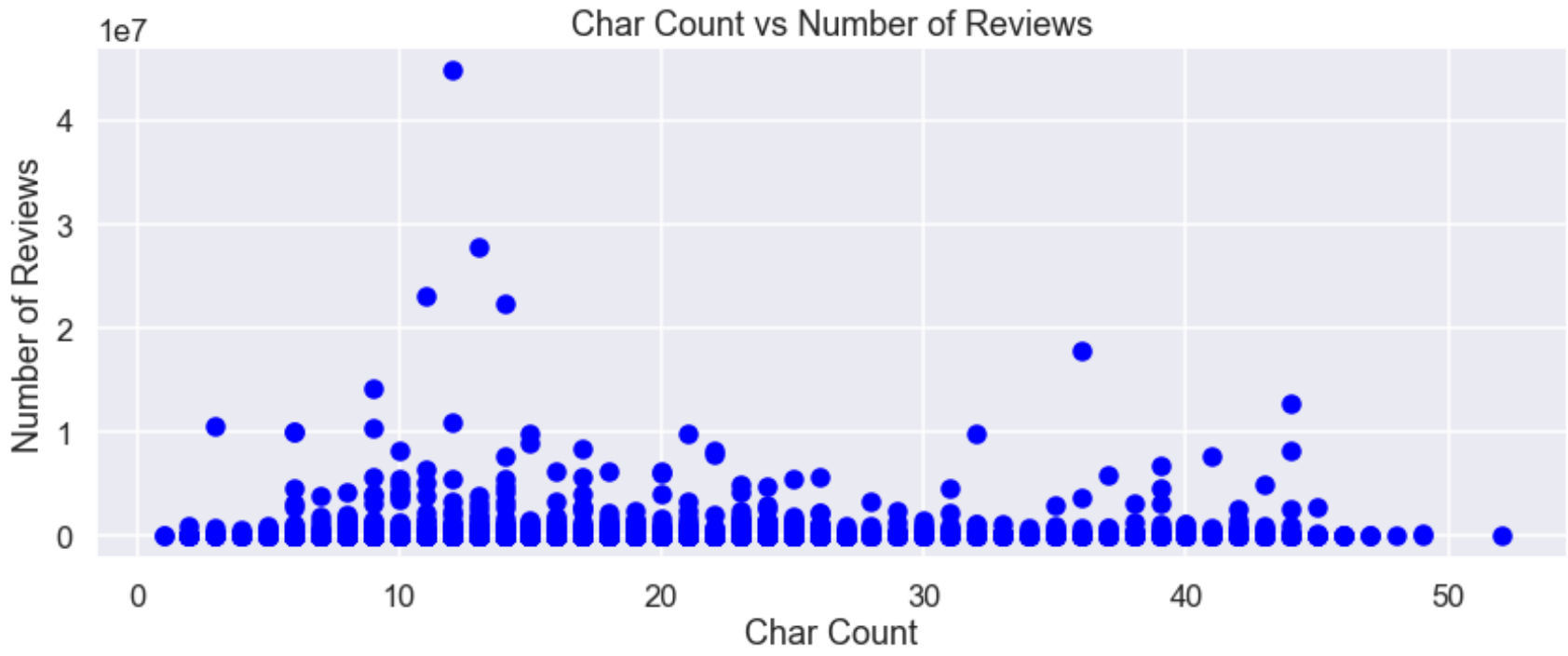
In [324]:
```
# Plots the average number of reviews (y-axis) for apps of each word count (x-axis)
x1 = df_new.index
fig, ax = plt.subplots()
fig.set_size_inches(14, 6)
plt.grid(None, color="white")
sns.barplot(x=x1,
            y='Reviews',
            data=df_new,
            ax = ax,
            palette=("Blues_d"));
ax.tick_params(axis = 'both', which = 'major', labelsize = 13)
plt.title('The Average Number of Reviews Over the Number of Characters in App Titles')
```

Out[324]: Text(0.5,1,'The Average Number of Reviews Over the Number of Characters in App Titles')



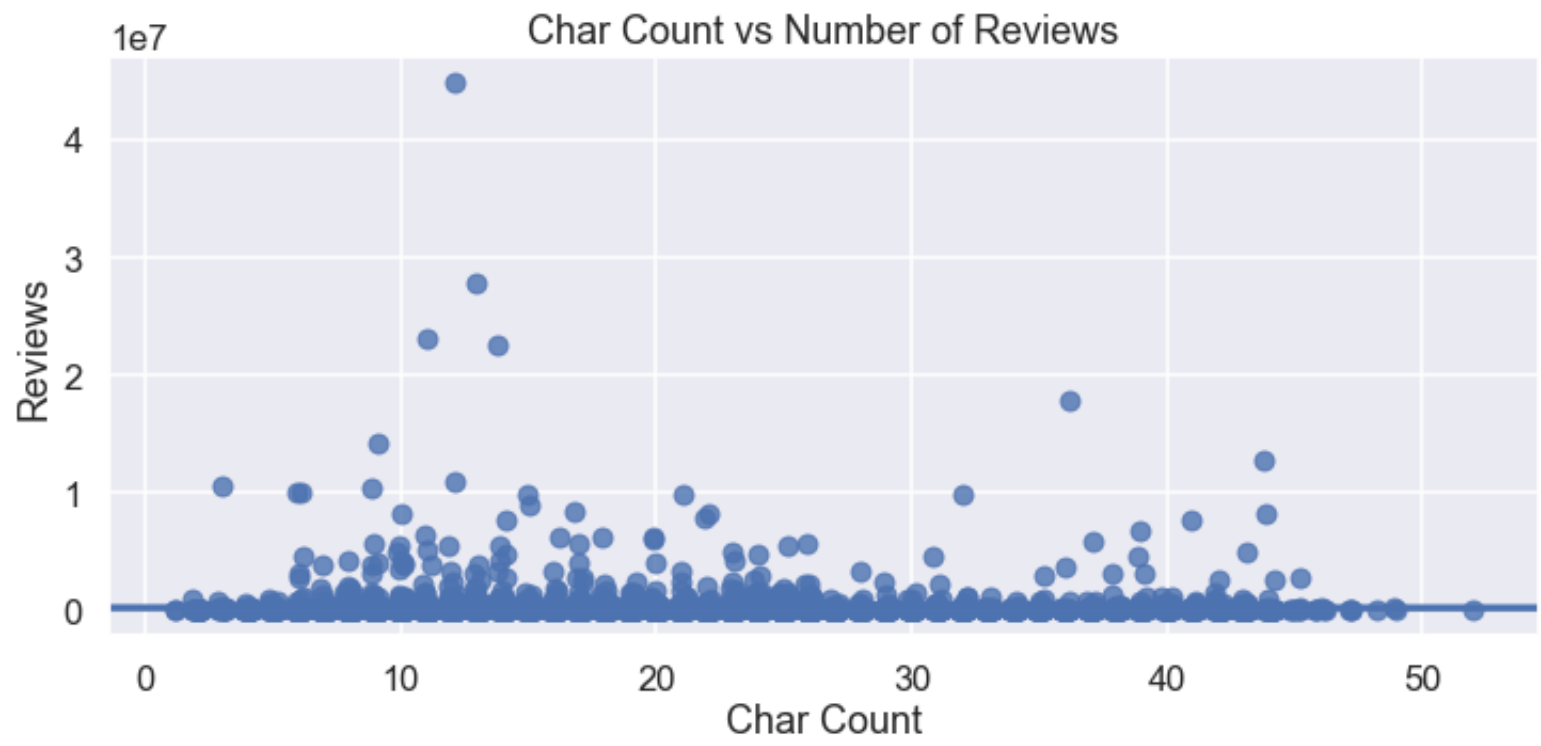
In [325]:
```
x = df['Char Count']
y = df['Reviews']
fig = plt.figure(figsize=(14, 5))
plt.scatter(x, y, c='blue')
plt.xlabel('Char Count')
plt.ylabel('Number of Reviews')
plt.title('Char Count vs Number of Reviews')
```

Out[325]: Text(0.5,1,'Char Count vs Number of Reviews')



In [326]:
```
# Use jitering to better visualize the above scatter plot
fig = plt.figure(figsize=(12, 5))
sns.regplot(x='Char Count', y='Reviews', data=df, x_jitter=0.2, y_jitter=1000)
plt.title('Char Count vs Number of Reviews')
```

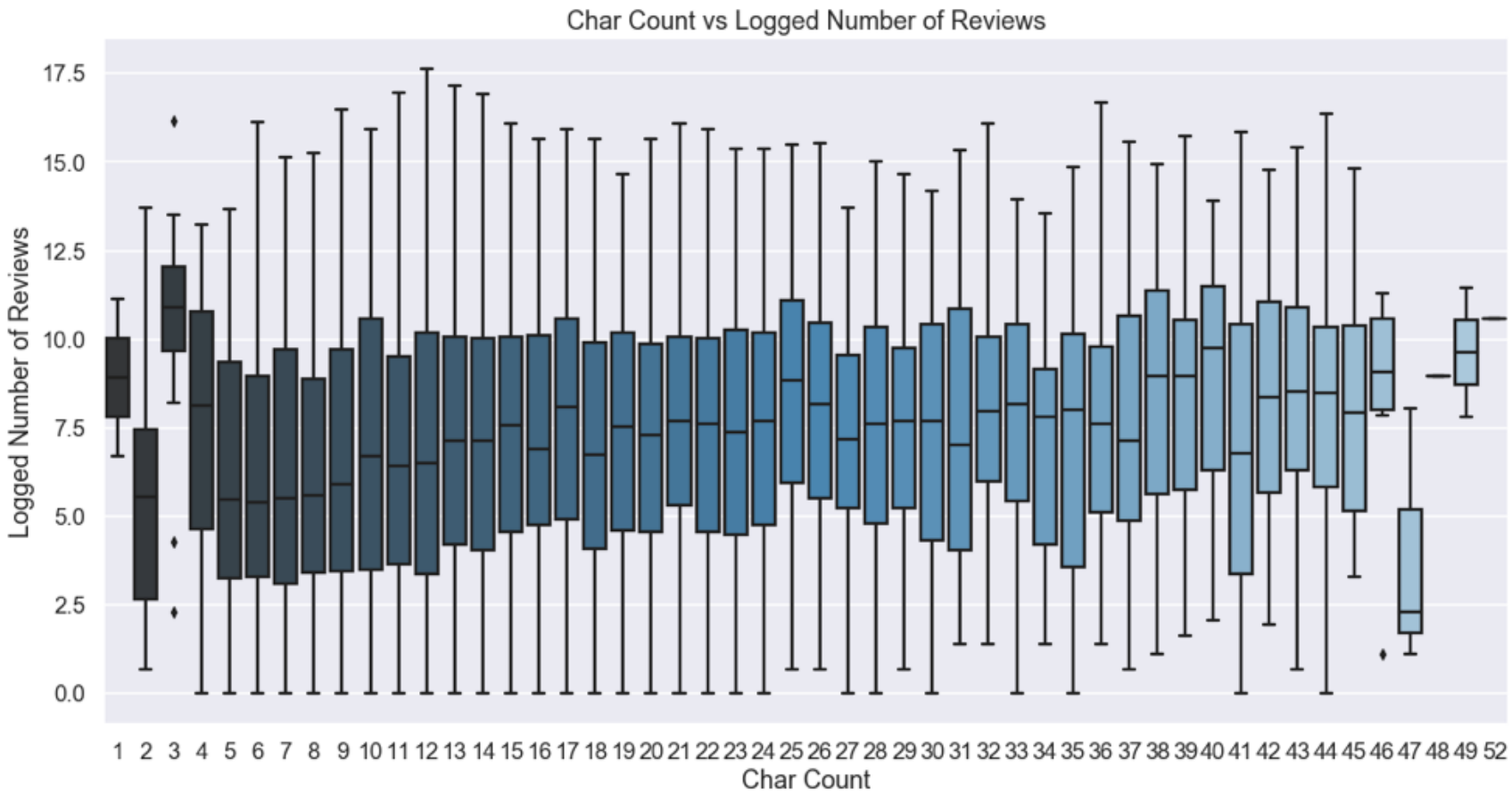
Out[326]: Text(0.5,1,'Char Count vs Number of Reviews')



With installs on the y axis and char count on the x axis, we can see that the graph above presents a clear trend. Although not a perfect positive skew, we can see that apps with a char count from 0-50 have a higher number of installs than apps with char counts greater than 50. Similar to the installs graph, the flat lines are due to the way the number of installs was categorized into ranges/buckets of values above a certain threshold, like 10,000+ or 50,000+. Therefore, we can observe the lines at 0.5 1e8 and 1 1e8, which mean that some apps with a char count of 0-50 had that number of installs.

In [327]:
```
plt.rcParams["font.size"] = "15"
fig = plt.figure(figsize=(18, 9))
sns.boxplot(x='Char Count', y='log_value_Reviews', data=df, palette="Blues_d")
ax = plt.gca()
plt.xlabel('Char Count')
plt.ylabel('Logged Number of Reviews')
plt.title('Char Count vs Logged Number of Reviews')
```

Out[327]: `Text(0.5,1,'Char Count vs Logged Number of Reviews')`



Now that we found a trend between word count and installs, is there any relationship between rating and word count?

Data Analysis & Results

Firstly, we will find by Wilcoxon rank sum test which of the letters/combinations of letters are not significant. Remember, we are doing so in order to decrease a chance of a potential multicollinearity problem in our main regression, because all these variables we will apply Wilcoxon test to are binary variables. Also, to account for outliers, we will use the log scale values of our review data for all further analysis.

In [328]:
```
#First we need a list of insignificant variables that will be useless in our regression

#vector that will contain insignificant columns concerning letters/combinations of letters
insignificant_letters = []

#following peace of code loops through every column from 'a' to 'zz'
#and tests whether 'Reviews' where this letter is present comes
#from the same distribution as distribution of 'Reviews' where this letter is absent
for col_name in df.columns[9:159]:

    presence_vector = df[df[col_name]==1]['log_value_Reviews']#reviews for which value of col_nam
e is 1
    absence_vector = df[df[col_name]==0]['log_value_Reviews']#reviews for which value of col_name
is 0

    test_stat, pval = scipy.stats.ranksums(presence_vector,absence_vector)

    #appending insignificant_letters list if the current column is insignificant at 0.01 level
    if pval > 0.01:
        insignificant_letters.append(col_name)

print(insignificant_letters)
```

```
['c', 'f', 'x', 'z', 'ac', 'ai', 'al', 'ap', 'at', 'ay', 'bi', 'ca', 'ce', 'ch', 'ci', 'co', 'c
t', 'de', 'di', 'ec', 'ed', 'el', 'em', 'en', 'es', 'et', 'ew', 'fi', 'fo', 'ha', 'he', 'hi', '
ia', 'ic', 'id', 'ie', 'il', 'io', 'is', 'it', 'la', 'le', 'li', 'lo', 'ma', 'mi', 'na', 'nc',
'ni', 'ns', 'nt', 'ob', 'od', 'ol', 'on', 'or', 'ou', 'pa', 'pe', 'pp', 'ri', 'ro', 'rs', 'rt',
'ry', 'se', 'si', 'ss', 'st', 'ta', 'te', 'th', 'ti', 'tr', 'tt', 'tu', 'ty', 'ul', 'um', 'ur',
'us', 'ut', 'vi', 'zy', 'zz']
```

So, based on the findings in the previous cell we can deduce that columns, that are present in insignificant_letters list are insignificant in explaining success measured as total number of reviews of an app. This fact allows us not to include those variables into regression in order to avoid potential severe multicollinearity.

Thus, we can drop these columns from our dataset.

In [329]:
```
df.drop(insignificant_letters, axis = 1, inplace = True)
```

Remember, that we will need to write huge amount of text inside **sm.OLS(*here*)** . To avoid typing this long text with all variables we need, we will create one list that will caontain all information we need.

In [330]:
```
df.columns
```

Out[330]:
```
Index(['App', 'Rating', 'Installs', 'Price', 'Size', 'Reviews', 'Category',
       'Word Count', 'Char Count', 'a', 'b', 'd', 'e', 'g', 'h', 'i', 'k', 'l',
       'm', 'n', 'o', 'p', 'r', 's', 't', 'u', 'v', 'w', 'y', 'ad', 'ag', 'am',
       'an', 'ar', 'as', 'bo', 'ck', 'ds', 'ea', 'ee', 'eo', 'er', 'ge', 'ho',
       'ig', 'im', 'in', 'ir', 'iv', 'ke', 'll', 'me', 'nd', 'ne', 'ng', 'oa',
       'oc', 'ok', 'om', 'oo', 'op', 'os', 'ot', 'ow', 'ra', 'rd', 're', 'sh',
       'to', 'ts', 'un', 've', 'zw', 'zx', 'log_value_Reviews'],
      dtype='object')
```

In [331]:
```
# reg_list will is final list of all regressors we need
reg_list = list(df.columns[7:-1])
reg_list.append('Price')
reg_list.append('Size')
```

Now, we are ready to perform OLS regression.

In [332]:
```
df.columns
```

Out[332]:
```
Index(['App', 'Rating', 'Installs', 'Price', 'Size', 'Reviews', 'Category',
       'Word Count', 'Char Count', 'a', 'b', 'd', 'e', 'g', 'h', 'i', 'k', 'l',
       'm', 'n', 'o', 'p', 'r', 's', 't', 'u', 'v', 'w', 'y', 'ad', 'ag', 'am',
       'an', 'ar', 'as', 'bo', 'ck', 'ds', 'ea', 'ee', 'eo', 'er', 'ge', 'ho',
       'ig', 'im', 'in', 'ir', 'iv', 'ke', 'll', 'me', 'nd', 'ne', 'ng', 'oa',
       'oc', 'ok', 'om', 'oo', 'op', 'os', 'ot', 'ow', 'ra', 'rd', 're', 'sh',
       'to', 'ts', 'un', 've', 'zw', 'zx', 'log_value_Reviews'],
      dtype='object')
```

In [333]:
```
model = sm.OLS(endog = df['log_value_Reviews'], exog = sm.add_constant(df[reg_list]))
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:      log_value_Reviews   R-squared:                       0.178
Model:                            OLS   Adj. R-squared:                  0.170
Method:                 Least Squares   F-statistic:                     21.70
Date:                Wed, 05 Jun 2019   Prob (F-statistic):          2.31e-232
Time:                        15:42:23   Log-Likelihood:                -17608.
No. Observations:                6777   AIC:                         3.535e+04
Df Residuals:                    6709   BIC:                         3.582e+04
Df Model:                          67
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.2726      0.124     42.530      0.000       5.030       5.516
Word Count    -0.1445      0.044     -3.275      0.001      -0.231      -0.058
Char Count    -0.0004      0.011     -0.036      0.971      -0.023       0.022
a              0.1691      0.109      1.545      0.122      -0.045       0.384
b              0.0975      0.125      0.780      0.435      -0.147       0.342
d              0.0621      0.116      0.538      0.591      -0.164       0.289
e              0.2013      0.118      1.710      0.087      -0.029       0.432
g              0.0583      0.167      0.349      0.727      -0.269       0.386
h             -0.0679      0.109     -0.626      0.531      -0.281       0.145
i             -0.1846      0.102     -1.809      0.070      -0.385       0.015
k              0.0523      0.163      0.322      0.748      -0.266       0.371
l             -0.0264      0.095     -0.278      0.781      -0.213       0.160
m              0.2395      0.132      1.818      0.069      -0.019       0.498
n              0.1340      0.111      1.212      0.225      -0.083       0.351
o              0.3144      0.103      3.066      0.002       0.113       0.515
p              0.0210      0.105      0.200      0.842      -0.185       0.227
r             -0.0337      0.119     -0.283      0.777      -0.267       0.199
s             -0.1026      0.097     -1.062      0.288      -0.292       0.087
t              0.0266      0.098      0.272      0.786      -0.165       0.218
u              0.0223      0.097      0.229      0.819      -0.168       0.213
v             -0.0008      0.193     -0.004      0.997      -0.380       0.378
w              0.7171      0.166      4.309      0.000       0.391       1.043
y              0.2238      0.098      2.282      0.023       0.032       0.416
ad             0.2375      0.222      1.071      0.284      -0.197       0.672
ag             0.6749      0.250      2.694      0.007       0.184       1.166
am            -0.0695      0.190     -0.365      0.715      -0.443       0.304
an            -0.0243      0.137     -0.178      0.859      -0.293       0.244
ar             0.0848      0.133      0.635      0.525      -0.177       0.346
as             0.5948      0.170      3.505      0.000       0.262       0.927
bo             0.4424      0.299      1.480      0.139      -0.143       1.028
ck             0.0807      0.214      0.377      0.706      -0.338       0.500
```

```
ds              0.0733      0.264      0.278      0.781      -0.443       0.590
ea              0.3010      0.151      1.996      0.046       0.005       0.597
ee              1.1807      0.173      6.812      0.000       0.841       1.520
eo              0.6463      0.274      2.355      0.019       0.108       1.184
er              0.2770      0.112      2.478      0.013       0.058       0.496
ge             -0.0633      0.208     -0.305      0.761      -0.470       0.344
ho              1.0014      0.216      4.633      0.000       0.578       1.425
ig              0.8176      0.252      3.242      0.001       0.323       1.312
im              0.1263      0.225      0.563      0.574      -0.314       0.567
in             -0.2299      0.149     -1.542      0.123      -0.522       0.062
ir              0.5997      0.214      2.808      0.005       0.181       1.018
iv              0.8673      0.242      3.577      0.000       0.392       1.343
ke              0.3877      0.209      1.853      0.064      -0.023       0.798
ll              0.6417      0.165      3.879      0.000       0.317       0.966
me             -0.1218      0.169     -0.721      0.471      -0.453       0.209
nd              0.1224      0.175      0.700      0.484      -0.220       0.465
ne              0.2757      0.162      1.700      0.089      -0.042       0.594
ng              0.4999      0.202      2.474      0.013       0.104       0.896
oa              0.3275      0.314      1.043      0.297      -0.288       0.943
oc              0.9045      0.220      4.110      0.000       0.473       1.336
ok              0.8105      0.315      2.575      0.010       0.193       1.428
om              0.0056      0.207      0.027      0.978      -0.399       0.411
oo              0.0126      0.183      0.069      0.945      -0.347       0.372
op              0.1536      0.264      0.581      0.561      -0.364       0.672
os              0.0580      0.238      0.243      0.808      -0.409       0.525
ot              0.1627      0.182      0.893      0.372      -0.194       0.520
ow             -0.1164      0.282     -0.412      0.680      -0.670       0.437
ra              0.1906      0.140      1.366      0.172      -0.083       0.464
rd              0.1549      0.250      0.619      0.536      -0.336       0.645
re              0.1017      0.137      0.741      0.459      -0.167       0.371
sh              0.8700      0.232      3.744      0.000       0.414       1.326
to              0.0322      0.147      0.219      0.826      -0.255       0.320
ts              0.6346      0.198      3.201      0.001       0.246       1.023
un              0.8139      0.194      4.189      0.000       0.433       1.195
ve             -0.0984      0.238     -0.413      0.680      -0.566       0.369
zw          2.513e-17    7.7e-17      0.327      0.744   -1.26e-16    1.76e-16
zx                  0          0        nan        nan           0           0
Price         -0.0021      0.002     -0.973      0.331      -0.006       0.002
Size        5.294e-05    1.8e-06     29.444      0.000    4.94e-05    5.65e-05
==============================================================================
Omnibus:                      141.439   Durbin-Watson:                   1.203
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               76.474
Skew:                          -0.039   Prob(JB):                     2.48e-17
Kurtosis:                       2.485   Cond. No.                     1.39e+16
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.47e-20. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

As we can see from Warnings in OLS summary, there is still a problem with multicollinearity. We will now build F-test for testing mutual insignificance of all letter regressors whose p-values are greater than 0.15 to see whether we can delete them from the set of the regressors.

In [334]:
```
hypothesis = '(b = 0),(d = 0), (g = 0), (h = 0), (k = 0), (l = 0), (n = 0), (p = 0), (r = 0), (s
= 0), (t = 0), (u = 0), (v = 0), (ad = 0), (am = 0), (an = 0), (ar = 0), (ck = 0), (ds = 0), (ge
= 0), (im = 0), (me = 0), (nd = 0), (oa = 0), (om = 0), (oo = 0), (op = 0), (os = 0), (ot = 0), (
ow = 0), (ra = 0), (rd = 0), (re = 0), (to = 0), (ve = 0), (zw = 0)'
f_test = results.f_test(hypothesis)
print(f_test)
```

```
<F test: F=array([[0.50275145]]), p=0.9936837972355893, df_denom=6709, df_num=35>
```

The p-value of F-test is nearly one, thus, we cannot reject the null hypothesis that the variables contained in *hypothesis* in the cell above are mutually insignificant in our regression. So we can delete them from our regression without loss of main results.

In [335]:
```
reg_list.remove('nd')
reg_list.remove('oa')
reg_list.remove('om')
reg_list.remove('oo')
reg_list.remove('op')
reg_list.remove('os')
reg_list.remove('ot')
reg_list.remove('ow')
reg_list.remove('ra')
reg_list.remove('rd')
reg_list.remove('re')
reg_list.remove('to')
reg_list.remove('ve')
reg_list.remove('zw')
reg_list.remove('b')
reg_list.remove('d')
reg_list.remove('g')
reg_list.remove('h')
reg_list.remove('k')
reg_list.remove('l')
reg_list.remove('n')
reg_list.remove('p')
reg_list.remove('r')
reg_list.remove('s')
reg_list.remove('t')
reg_list.remove('u')
reg_list.remove('v')
reg_list.remove('ad')
reg_list.remove('am')
reg_list.remove('an')
reg_list.remove('ar')
reg_list.remove('ck')
reg_list.remove('ds')
reg_list.remove('ge')
reg_list.remove('im')
reg_list.remove('me')
```

Now we are ready to run new regression with corrected list of independent variables to hope to avoid multicollinearity problem.

In [336]:
```
model = sm.OLS(endog = df['log_value_Reviews'], exog = sm.add_constant(df[reg_list]))
results = model.fit()
print(results.summary())
```

OLS Regression Results

Dep. Variable:	log_value_Reviews	R-squared:	0.176
Model:	OLS	Adj. R-squared:	0.172
Method:	Least Squares	F-statistic:	45.00
Date:	Wed, 05 Jun 2019	Prob (F-statistic):	1.18e-254
Time:	15:42:24	Log-Likelihood:	-17617.
No. Observations:	6777	AIC:	3.530e+04
Df Residuals:	6744	BIC:	3.552e+04
Df Model:	32		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.2622	0.120	43.915	0.000	5.027	5.497
Word Count	-0.1593	0.043	-3.704	0.000	-0.244	-0.075
Char Count	0.0077	0.010	0.780	0.435	-0.012	0.027
a	0.2380	0.098	2.434	0.015	0.046	0.430
e	0.1792	0.111	1.620	0.105	-0.038	0.396
i	-0.1814	0.099	-1.840	0.066	-0.375	0.012
m	0.1931	0.094	2.054	0.040	0.009	0.377
o	0.3570	0.094	3.804	0.000	0.173	0.541
w	0.6887	0.136	5.047	0.000	0.421	0.956
y	0.2225	0.097	2.298	0.022	0.033	0.412
ag	0.6942	0.209	3.322	0.001	0.285	1.104
as	0.5346	0.162	3.308	0.001	0.218	0.851
bo	0.7018	0.253	2.770	0.006	0.205	1.198
ea	0.3142	0.144	2.179	0.029	0.032	0.597
ee	1.2286	0.157	7.845	0.000	0.922	1.536
eo	0.6908	0.266	2.601	0.009	0.170	1.211
er	0.2446	0.098	2.503	0.012	0.053	0.436
ho	1.0034	0.182	5.521	0.000	0.647	1.360
ig	0.7958	0.209	3.803	0.000	0.386	1.206
in	-0.1795	0.140	-1.281	0.200	-0.454	0.095
ir	0.5697	0.208	2.742	0.006	0.162	0.977
iv	0.7841	0.193	4.058	0.000	0.405	1.163
ke	0.4631	0.180	2.579	0.010	0.111	0.815
ll	0.5845	0.158	3.709	0.000	0.276	0.893
ne	0.2857	0.156	1.832	0.067	-0.020	0.592
ng	0.5477	0.149	3.664	0.000	0.255	0.841
oc	0.9109	0.204	4.476	0.000	0.512	1.310
ok	0.7764	0.267	2.912	0.004	0.254	1.299
sh	0.7982	0.217	3.684	0.000	0.374	1.223
ts	0.5820	0.189	3.074	0.002	0.211	0.953
un	0.8681	0.173	5.017	0.000	0.529	1.207
zx	-1.116e-15	3.53e-16	-3.161	0.002	-1.81e-15	-4.24e-16
Price	-0.0024	0.002	-1.121	0.262	-0.007	0.002
Size	5.303e-05	1.78e-06	29.837	0.000	4.95e-05	5.65e-05

Omnibus:	153.110	Durbin-Watson:	1.203
Prob(Omnibus):	0.000	Jarque-Bera (JB):	80.983
Skew:	-0.041	Prob(JB):	2.60e-18
Kurtosis:	2.471	Cond. No.	1.39e+16

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.47e-20. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

As we can see again from Warnings in OLS summary, there is still a problem with multicollinearity. We will now again build F-test for testing mutual insignificance of all remaining letter regressors whose p-values are greater than 0.05 to see whether we can delete them from the set of the regressors.

In [337]:
```
hypothesis = '(e = 0),(i = 0), (in = 0), (ne = 0)'
f_test = results.f_test(hypothesis)
print(f_test)
```

```
<F test: F=array([[2.97926401]]), p=0.01804632325145511, df_denom=6744, df_num=4>
```

As we can see from the results of the F-test, low p-value (lower than 0.025) allows us to reject the null hypothesis that 'e', 'i', 'in', 'ne' are mutually insignificant. Now we find such a subset from these variables that they will be mutually independent.

In [338]:
```
hypothesis = '(e = 0),(in = 0),(ne = 0)'
f_test = results.f_test(hypothesis)
print(f_test)
```

```
<F test: F=array([[2.53227147]]), p=0.0552139135028934, df_denom=6744, df_num=3>
```

By the results of another F-test, we are allowed to drop 'e', 'in','ne' from list of regressors(p-value is greater than 0.05).

In [339]:
```
reg_list.remove('ne')
reg_list.remove('in')
reg_list.remove('e')
```

In [340]:
```
model = sm.OLS(endog = df['log_value_Reviews'], exog = sm.add_constant(df[reg_list]))
results = model.fit()
print(results.summary())
```

OLS Regression Results

Dep. Variable:	log_value_Reviews	R-squared:	0.175
Model:	OLS	Adj. R-squared:	0.171
Method:	Least Squares	F-statistic:	49.36
Date:	Wed, 05 Jun 2019	Prob (F-statistic):	1.43e-255
Time:	15:42:24	Log-Likelihood:	-17621.
No. Observations:	6777	AIC:	3.530e+04
Df Residuals:	6747	BIC:	3.551e+04
Df Model:	29		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.3380	0.112	47.782	0.000	5.119	5.557
Word Count	-0.1599	0.043	-3.719	0.000	-0.244	-0.076
Char Count	0.0097	0.010	0.985	0.325	-0.010	0.029
a	0.2425	0.098	2.481	0.013	0.051	0.434
i	-0.1970	0.095	-2.078	0.038	-0.383	-0.011
m	0.1901	0.094	2.030	0.042	0.006	0.374
o	0.3752	0.094	4.013	0.000	0.192	0.559
w	0.7004	0.136	5.137	0.000	0.433	0.968
y	0.2273	0.097	2.346	0.019	0.037	0.417
ag	0.7043	0.209	3.371	0.001	0.295	1.114
as	0.5182	0.161	3.209	0.001	0.202	0.835
bo	0.6900	0.253	2.724	0.006	0.193	1.187
ea	0.3497	0.143	2.450	0.014	0.070	0.630
ee	1.2640	0.155	8.132	0.000	0.959	1.569
eo	0.7159	0.265	2.705	0.007	0.197	1.235
er	0.3010	0.094	3.201	0.001	0.117	0.485
ho	1.0239	0.181	5.647	0.000	0.668	1.379
ig	0.7983	0.209	3.820	0.000	0.389	1.208
ir	0.5981	0.207	2.886	0.004	0.192	1.004
iv	0.7887	0.193	4.088	0.000	0.410	1.167
ke	0.4753	0.179	2.654	0.008	0.124	0.826
ll	0.5948	0.157	3.779	0.000	0.286	0.903
ng	0.4379	0.125	3.512	0.000	0.193	0.682
oc	0.9013	0.204	4.428	0.000	0.502	1.300
ok	0.7605	0.267	2.852	0.004	0.238	1.283
sh	0.8149	0.216	3.768	0.000	0.391	1.239
ts	0.5833	0.189	3.081	0.002	0.212	0.955
un	0.8829	0.173	5.109	0.000	0.544	1.222
zx	6.923e-16	7.85e-16	0.882	0.378	-8.46e-16	2.23e-15
Price	-0.0025	0.002	-1.153	0.249	-0.007	0.002
Size	5.298e-05	1.78e-06	29.803	0.000	4.95e-05	5.65e-05

Omnibus:	157.892	Durbin-Watson:	1.202
Prob(Omnibus):	0.000	Jarque-Bera (JB):	82.857
Skew:	-0.043	Prob(JB):	1.02e-18
Kurtosis:	2.465	Cond. No.	1.39e+16

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.47e-20. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

As it can be seen fron Warnings of the regression, there is still a multicollinearity problem. We have made good efforts to avoid it, but it is still present. We will continue our analysis having in mind this problem.

Now let's test another OLS assumption, i.e. homoscedasticity of errors. To do so, we will buils a Breusch-Pagan test. The null hypothesis is that the variance of errors of regression is homoscedastic.

In [341]:
```
br_pagan_results = statsmodels.stats.diagnostic.het_breuschpagan(results.resid,statsmodels.tools.
add_constant(df[reg_list]))
print(br_pagan_results)
```

```
(109.9594782074008, 4.669512718688483e-11, 3.8371810216619266, 1.8480737584868566e-11)
```

The number we need to take into account is the second number from the list above. This is the p-value of Breusch-Pagan test that indicates that the null hypothesis of homoscedasticity in errors can be rejected(p_value is almost zero). Thus, we need to build another regression with standard errors robust to heteroscedasticity.

In [342]:
```
model = sm.OLS(endog = df['log_value_Reviews'], exog = sm.add_constant(df[reg_list]))
results = model.fit(cov_type = 'HC3', use_t = None)
print(results.summary())
```

OLS Regression Results

Dep. Variable:	log_value_Reviews	R-squared:	0.175
Model:	OLS	Adj. R-squared:	0.171
Method:	Least Squares	F-statistic:	51.55
Date:	Wed, 05 Jun 2019	Prob (F-statistic):	1.16e-266
Time:	15:42:24	Log-Likelihood:	-17621.
No. Observations:	6777	AIC:	3.530e+04
Df Residuals:	6747	BIC:	3.551e+04
Df Model:	29		
Covariance Type:	HC3		

	coef	std err	z	P>|z|	[0.025	0.975]
const	5.3380	0.115	46.486	0.000	5.113	5.563
Word Count	-0.1599	0.043	-3.715	0.000	-0.244	-0.076
Char Count	0.0097	0.010	0.948	0.343	-0.010	0.030
a	0.2425	0.099	2.461	0.014	0.049	0.436
i	-0.1970	0.095	-2.083	0.037	-0.382	-0.012
m	0.1901	0.093	2.052	0.040	0.009	0.372
o	0.3752	0.095	3.951	0.000	0.189	0.561
w	0.7004	0.138	5.072	0.000	0.430	0.971
y	0.2273	0.097	2.333	0.020	0.036	0.418
ag	0.7043	0.207	3.401	0.001	0.298	1.110
as	0.5182	0.160	3.232	0.001	0.204	0.832
bo	0.6900	0.253	2.724	0.006	0.194	1.187
ea	0.3497	0.140	2.505	0.012	0.076	0.623
ee	1.2640	0.155	8.170	0.000	0.961	1.567
eo	0.7159	0.266	2.687	0.007	0.194	1.238
er	0.3010	0.093	3.228	0.001	0.118	0.484
ho	1.0239	0.183	5.585	0.000	0.665	1.383
ig	0.7983	0.199	4.010	0.000	0.408	1.188
ir	0.5981	0.213	2.813	0.005	0.181	1.015
iv	0.7887	0.202	3.895	0.000	0.392	1.186
ke	0.4753	0.170	2.801	0.005	0.143	0.808
ll	0.5948	0.168	3.537	0.000	0.265	0.924
ng	0.4379	0.125	3.507	0.000	0.193	0.683
oc	0.9013	0.217	4.162	0.000	0.477	1.326
ok	0.7605	0.235	3.232	0.001	0.299	1.222
sh	0.8149	0.215	3.781	0.000	0.393	1.237
ts	0.5833	0.184	3.165	0.002	0.222	0.945
un	0.8829	0.166	5.323	0.000	0.558	1.208
zx	6.923e-16	7.94e-16	0.872	0.383	-8.64e-16	2.25e-15
Price	-0.0025	0.002	-1.649	0.099	-0.005	0.000
Size	5.298e-05	1.86e-06	28.520	0.000	4.93e-05	5.66e-05

Omnibus:	157.892	Durbin-Watson:	1.202
Prob(Omnibus):	0.000	Jarque-Bera (JB):	82.857
Skew:	-0.043	Prob(JB):	1.02e-18
Kurtosis:	2.465	Cond. No.	1.39e+16

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC3)
[2] The smallest eigenvalue is 3.47e-20. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

Regression above is the final version of regression; unfortunately we were unable to avoid multicollinearity problem, but still our results are valid. Here we discuss partially our main findings: (we assume significance level to be 0.05)

1) variables that are not connected with title(i.e. Size and Price): Size appears to be significant explanator of apps success measured as total number of reviews for the app, whereas Price is not significant.

2) length of title(Word count) appears to be strongly significant in explaining apps success, whereas Char count has extremely large p-values, thus it has no effect on apps success.

Ethics & Privacy

The dataset we obtained has an adequate amount of observations and very related variables that helps us to measure the success of the app. We obtained the dataset from Kaggle, a site that allows user to use and publish dataset, and the dataset we use is attached with license that granted user freedom to copy, redistribute, and build upon the material. (license link: https://creativecommons.org/licenses/by/3.0/ (https://creativecommons.org/licenses/by/3.0/)) Since the dataset is built upon data that is public in the Google Play Store where all users can see directly, there is not much potential privacy issue. In fact, the reviews for applications did not have any personal information attached to them.

Since the dataset we used contains pre-processed information like sentiment, sentiment polarity and sentiment subject, it may contain bias that is unknown to us because we can not see how they are calculated. Therefore, we did not use this information in our analysis. Since our data comes from Google Play Store, it mainly focus on the population using Android system phones. The other most prominent source of bias in our dataset is from the number of installs variable. As we have discussed before and will discuss more about in our conclusion, this number of installs only has 19 unique categorical "buckets" that must have been chosen by whoever made the dataset.

We intended to build a tool that measures the success of an app to offer reference or advice for people who might develop an app themselves. Since our data only contains public information and not any personal information, and we are only analyzing the existing data, there is little possibility for any exploitation or use of our work for harmful purpose. Even though prominent app developers can use our work for the possibility of making future apps they publish even more successful, essentially enforicng a monopoly in the Play Store, already famous and published applications cannot use our work, as changing the titles of apps while they already are on the market will have a greater chance of decreasing the apps' success rather than increasing. Possible impacts of our work are that people may change their decisions about the names of the app they are developing, or people may change their opinions about certain apps. However, our analysis shows that title can be a limited factor deciding on an app's success, so there will be few unintended consequences.

Conclusion & Discussion

Our project focuses on determining factors of success for an app. Unlike the previous work we discussed in the background, our research revolves around the number of reviews. We came to use this feature as our measure of success by analyzing the correlation between different features that are possibly related to the success of an app, such as ratings and number of installs. After examining correlations between the different variables, we reached the conclusion that reviews will be an appropriate measure of success.

From our point of view, the number of reviews is an accurate measure of an application's success; At first, we came up with the idea of using installs as measure of success, since we believe that installs directly tells how successful an app is. Other possible options such as reviews and rating seemed less reliable than installs. Rating is in some sense a biased and limited variable; it can be seen from our project that the majority of its values are concentrated in the interval between 3 and 5. We believe that this is because people tend to either give more positive ratings or not giving ratings at all. Then, we recognized that our main candidate for being an accurate measure of success (i.e. the number of installs) is also a limited variable. As a result of Installs being rounded into bins in our dataset, the number of installs could even be seen as categorical. There are only 19 unique values of the number of installs, and the variable does not represent a more exact, accurate amount of installs for each app as we had expected. Consequently, we decided on not using installs as our measure of success. Fortunately, we found that the reviews variable (the other candidate for measure of success) is strongly correlated with installs, so we were able to utilize it as a dependent variable.

Despite a partially successful hypothesis, we came across pitfalls. As mentioned in the paragraph above, we were not able to use number of installations as a measure of success due to its properties in our dataset. Also, we suspect successful apps are more likely to snowball and become even exponentially more successful, thus creating a large gap between apps with very little success and apps with significant success. However, it is inevitable that there are very few of these highly successful apps relative to all of the apps in the Google Play Store or even in our dataset. Thus, an obvious limitation in our project is that the weights of these highly successful apps may not be optimal for the best analysis of the causes of success. To account for these outliers, we used the log scale values of our review data in our analysis.

Findings:

Our hypothesis was supported by the analysis. Something we suspected was that app title length would be a significant variable in predicting app success. After analysis we saw that title word count was a very significant variable in determining app success, with a p-value of 0.000 which leaves minimal probability for this to happen by chance alone. This was not as surprising to us, as we thought that apps with smaller names would appeal to the population more (for example, most of the apps on our phone range from 1-2 words).

For our independent variable the price of applications, however, we did not find a significant effect at our alpha level, meaning a difference in price did not indicate a significant difference in the success of an app. We expected this variable to be significant, but our results did not support our hypothesis.

The size of an application, another independent variable that is not related with title, appears to be strongly significant. The interesting fact is that the coefficient on this variable is positive. This means that the bigger the size of an application, the more successful the app is(in terms of number of reviews). We believe that It can be explained by the following logic: the more the size of an app, the more functionality it can offer; the more functionality it can offer, the more people would potentially use this app, thus, it would potentially be more successful.

In addition to word count and size we noticed that there are a lot of letter combinations that are also a significant indication of app success. An interesting finding that we noticed was that among the significant combinations were 'ee' and 'll'; we believe this may be due to the fact that these are double-lettered combinations. Further, having the letter 'i' in an app title can negatively affect the success of the app!

Potential further research based on our analysis would include understanding whether or not these letters are significant because they are present in successful apps, or if apps are successful because these combinations are present.

The contribution of this project to the topic domain will be that future app developers can use this research to possibly help their app get more noticed. If our analyses and hypotheses are correct, that means that there are certain features that lead to an app success (in comparison to other features), such as specific letter combinations and the size of an app. Similarly, this can help future app developers to tailor applications to what the public desires (maybe smaller/bigger sizes) which will help them appeal to customers more. That being said, measures of success can vary with each app, and our research does not guarantee an app will be successful if the appropriate specific features are implemented.

A complete summary of our findings can be found below:

In [343]:
```
print(results.summary())
```

OLS Regression Results

Dep. Variable:	log_value_Reviews	R-squared:	0.175
Model:	OLS	Adj. R-squared:	0.171
Method:	Least Squares	F-statistic:	51.55
Date:	Wed, 05 Jun 2019	Prob (F-statistic):	1.16e-266
Time:	15:42:25	Log-Likelihood:	-17621.
No. Observations:	6777	AIC:	3.530e+04
Df Residuals:	6747	BIC:	3.551e+04
Df Model:	29		
Covariance Type:	HC3		

	coef	std err	z	P>|z|	[0.025	0.975]
const	5.3380	0.115	46.486	0.000	5.113	5.563
Word Count	-0.1599	0.043	-3.715	0.000	-0.244	-0.076
Char Count	0.0097	0.010	0.948	0.343	-0.010	0.030
a	0.2425	0.099	2.461	0.014	0.049	0.436
i	-0.1970	0.095	-2.083	0.037	-0.382	-0.012
m	0.1901	0.093	2.052	0.040	0.009	0.372
o	0.3752	0.095	3.951	0.000	0.189	0.561
w	0.7004	0.138	5.072	0.000	0.430	0.971
y	0.2273	0.097	2.333	0.020	0.036	0.418
ag	0.7043	0.207	3.401	0.001	0.298	1.110
as	0.5182	0.160	3.232	0.001	0.204	0.832
bo	0.6900	0.253	2.724	0.006	0.194	1.187
ea	0.3497	0.140	2.505	0.012	0.076	0.623
ee	1.2640	0.155	8.170	0.000	0.961	1.567
eo	0.7159	0.266	2.687	0.007	0.194	1.238
er	0.3010	0.093	3.228	0.001	0.118	0.484
ho	1.0239	0.183	5.585	0.000	0.665	1.383
ig	0.7983	0.199	4.010	0.000	0.408	1.188
ir	0.5981	0.213	2.813	0.005	0.181	1.015
iv	0.7887	0.202	3.895	0.000	0.392	1.186
ke	0.4753	0.170	2.801	0.005	0.143	0.808
ll	0.5948	0.168	3.537	0.000	0.265	0.924
ng	0.4379	0.125	3.507	0.000	0.193	0.683
oc	0.9013	0.217	4.162	0.000	0.477	1.326
ok	0.7605	0.235	3.232	0.001	0.299	1.222
sh	0.8149	0.215	3.781	0.000	0.393	1.237
ts	0.5833	0.184	3.165	0.002	0.222	0.945
un	0.8829	0.166	5.323	0.000	0.558	1.208
zx	6.923e-16	7.94e-16	0.872	0.383	-8.64e-16	2.25e-15
Price	-0.0025	0.002	-1.649	0.099	-0.005	0.000
Size	5.298e-05	1.86e-06	28.520	0.000	4.93e-05	5.66e-05

Omnibus:	157.892	Durbin-Watson:	1.202
Prob(Omnibus):	0.000	Jarque-Bera (JB):	82.857
Skew:	-0.043	Prob(JB):	1.02e-18
Kurtosis:	2.465	Cond. No.	1.39e+16

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC3)
[2] The smallest eigenvalue is 3.47e-20. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.