Pokemon GO! Analytics Report

Peter Hung (peterkhung@gmail.com)

Business Programming and Analytics

1.0 Web Scraping & Data Organization

1.1 Initial Setup

The first challenge encountered was to iterate through each folder in `2017W2` and each subfolder within. We used `os.walk` to recursively go through each folder and open up each file. However, not all files contained the information needed; there was a `‘.DS_store’` file created by running the code and trying to read further files in it would cause an error. To combat this, an if statement was implemented to check and skip if the file is a `‘.DS_store’` file.

To establish a dictionary following the datetime format, the dt variable was established, breaking each each file/folder name into relevant chunks. We found it unnecessary to use `datetime.datetime` to establish the key and used a slicing method instead in order to not include ‘datetime’ as part of the key.

1.2 Android

Opening File
A file path variable is established by using `os.path.join` the subdir and the rootdir, then a specific Android BeautifulSoup object is created. A minor note, `‘lxml’` was used inplace of the `html.parser` in both iOS and Android to speed up the processing of the code.

Average Score
We recognized that there were two different score values within the html file, rounded and unrounded. Although attaining the rounded value was easier as it was between the `div class=’score’` tags, we wanted to get the most accurate value. The desired value hid under `<meta content="3.8953652381896973" itemprop="ratingValue">`. Without a specific class to find, we established a tag variable that looked for other attributes within:

```
tag = and_soup.find(attrs={'itemprop':'ratingValue'})
```

As `‘ratingValue’` is unique to our desired tag, we did not have to use `.find_all`. Instead, we treated tag as a dictionary and called for value in `content`:

```
avg_scr = tag['content']
```

Number of Reviews
This was not a difficult value to attain as it hid under an unique tag with a specific class `<span class="reviews-num" aria-label=" 1,281,802 ratings ">1,281,802</span>`.

By using the .find method while specifying the class as reviews-num, the desired value can be attained.

File Size

While using similar codes to scrape score and file size values, we noticed certain points at which the code would break. After further investigation, we discovered some html pages to not have the file size, meaning a NoneType object was created when we ran the following code:

```
file_size = and_soup.find('div', itemprop ='fileSize’).gettext().strip()
```

In order to have the code drop the entire `datetime` key when there was no file size, we use a try except method so a value would only be saved to `file_size_and` if no error occured while running the code. Accordingly, we also use try except to append all data gathered to our dictionary variable `d`. If no android data was saved to the dictionary, the datetime key `dt` would not have been saved either. In this situation, the if statement we used to append the ios data will not run. As such, the entire datetime key is not appended in the first place, hence dropped.

Star Ratings

Similar to “Number of Reviews” in Section 1.2, the desired values fell under an unique tag and class for each individual star (5, 4, 3, 2, 1). Thus, establishing an unique variable for each star and using `.find` was sufficient to attain the number of reviews for each star.

1.3 iOS

Current Ratings

Similar to “Number of Reviews” in Section 1.2, there was an unique class and tag and finding the current ratings value was not difficult. To remove the unwanted letters, we used a `forloop` that checked if each character in the result is number or not. If yes, then use an accumulator to build the desired value.

All Ratings

There was no unique tag for All Ratings. Similar to “Average Score” in Section 1.2, we specified the values that we know were unique to the All Ratings tag to locate this value.

File Size

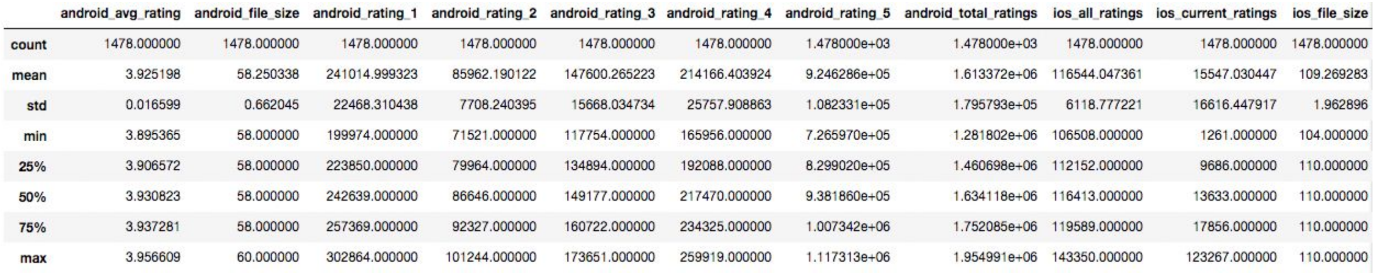
The file size in the iTunes Webstore is in its own separate line of code, making `.find` useless to find this value. We recognized the value fell under the `<li>` tag; however, as there are multiple `<li>` tags, we used `.find_all` and used an if statement containing `‘Size:’`.

1.4 Extra

As the processing code processing time is long, we implemented a progress percentage of completion throughout the code to get a sense of when the program will finish computing. We also included the name of the file it was currently running through so we could see when the code would break, identify the file causing the issue, and resolve accordingly.

2.0 Data Exploration

2.1 Basic Data: Count, Mean, Standard Deviation, Minimum, 25%/50%/75%, Maximum



	android_avg_rating	android_file_size	android_rating_1	android_rating_2	android_rating_3	android_rating_4	android_rating_5	android_total_ratings	ios_all_ratings	ios_current_ratings	ios_file_size
count	1478.000000	1478.000000	1478.000000	1478.000000	1478.000000	1478.000000	1.478000e+03	1.478000e+03	1478.000000	1478.000000	1478.000000
mean	3.925198	58.250338	241014.999323	85962.190122	147600.265223	214166.403924	9.246286e+05	1.613372e+06	116544.047361	15547.030447	109.269283
std	0.016599	0.662045	22468.310438	7708.240395	15668.034734	25757.908863	1.082331e+05	1.795793e+05	6118.777221	16616.447917	1.962896
min	3.895365	58.000000	199974.000000	71521.000000	117754.000000	165956.000000	7.265970e+05	1.281802e+06	106508.000000	1261.000000	104.000000
25%	3.906572	58.000000	223850.000000	79964.000000	134894.000000	192088.000000	8.299020e+05	1.460698e+06	112152.000000	9686.000000	110.000000
50%	3.930823	58.000000	242639.000000	86646.000000	149177.000000	217470.000000	9.381860e+05	1.634118e+06	116413.000000	13633.000000	110.000000
75%	3.937281	58.000000	257369.000000	92327.000000	160722.000000	234325.000000	1.007342e+06	1.752085e+06	119589.000000	17856.000000	110.000000
max	3.956609	60.000000	302864.000000	101244.000000	173651.000000	259919.000000	1.117313e+06	1.954991e+06	143350.000000	123267.000000	110.000000

To analyze the data with basic data description values, we used `pandas dataframe` to turn the dictionary data we had collected into a data frame, then transposed it to have the values in the correct axes before using `.describe()`

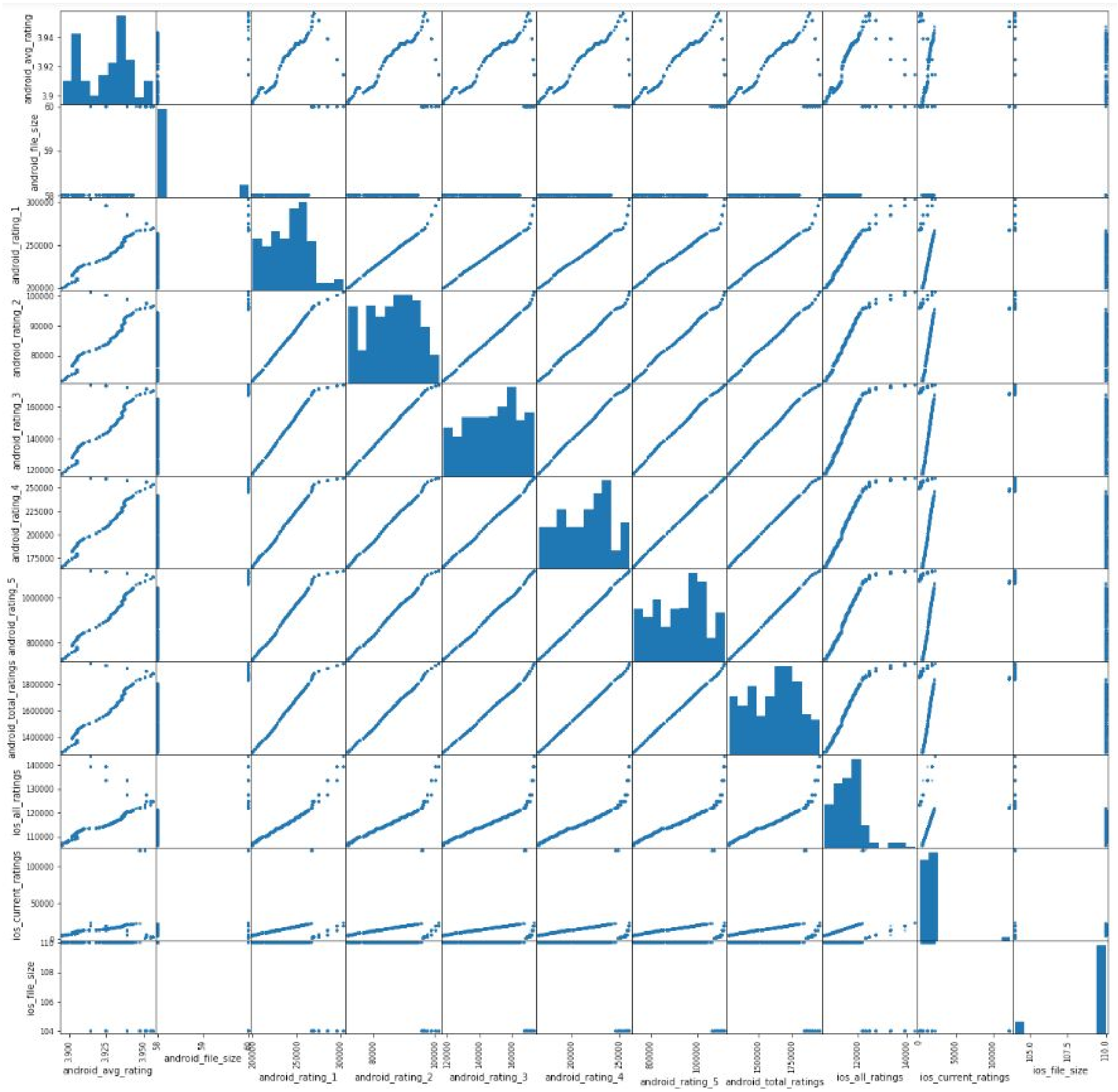
Based on the data displayed above, we can see that there were a total of 1478 values for each variable, which excludes the datetime files we dropped as a result of missing variables.

The android average rating was 3.925, though the standard deviation was only 0.016. This tells us that on a 5 star scale, on any given day, the average android user thought the app was about 3.9/5, and this doesn't change much by the day.

The ios ratings are not read in the same way, as the information displayed in the html files simply show us how many ratings have been received overall for the app and how many ratings have been received for the current version. It is normal for all-time ratings to increase as the app gains more users and receives reviews. Following this line, it makes sense that current version rating numbers increase as the app updates with better features and fixes bugs, as the consumer base increases and there is more possibility for more reviews (i.e., 20% of user reviews from a 2,000 person user base is much smaller than a 200,000 person user base).

Neither android nor ios had too large a jump in terms of file size, though it's interesting to note through later time series analyses found in section 2.4 that while the android file size increases, the ios file sizes actually decrease.

2.2 Correlation: Scatter Matrix



Interpretation
Based on the scatter matrix, it can be observed that most values are very positively correlated. However, looking into the code more closely there are several important observations:

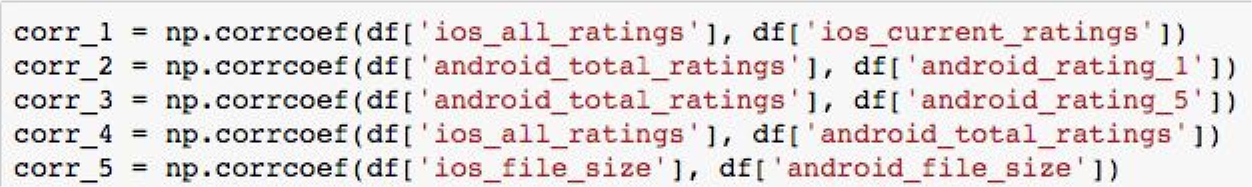
Android Total Ratings vs Android Rating 1

Observation: Despite its general strong positive correlation, near the end of the plot, the plots are more scattered.

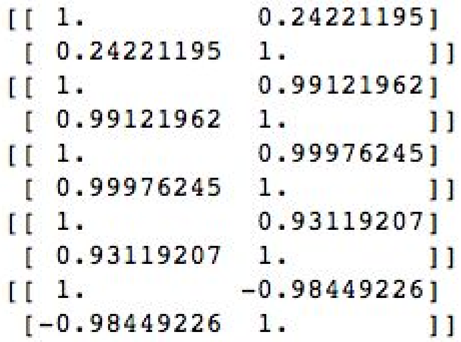
Assumption: As Android total ratings go up, it can be assumed that there are further version releases. With further updates, it can be assumed that prior problems are fixed.

Analysis: Therefore, as improvements are made the correlation between one-star reviews fall off. Users make less one-star reviews and it can be further observed that as stars progress from two to five, the correlation becomes stronger with Android Total Ratings. This is important for the Niantic because due to the initial criticism amongst users during launch, the developer faced harsh feedback for its bugs and battery drain issues. The improvements made over time are recognized by the developer's player base.

2.3 Correlation: Pearson Correlation Coefficients



```
corr_1 = np.corrcoef(df['ios_all_ratings'], df['ios_current_ratings'])
corr_2 = np.corrcoef(df['android_total_ratings'], df['android_rating_1'])
corr_3 = np.corrcoef(df['android_total_ratings'], df['android_rating_5'])
corr_4 = np.corrcoef(df['ios_all_ratings'], df['android_total_ratings'])
corr_5 = np.corrcoef(df['ios_file_size'], df['android_file_size'])
```

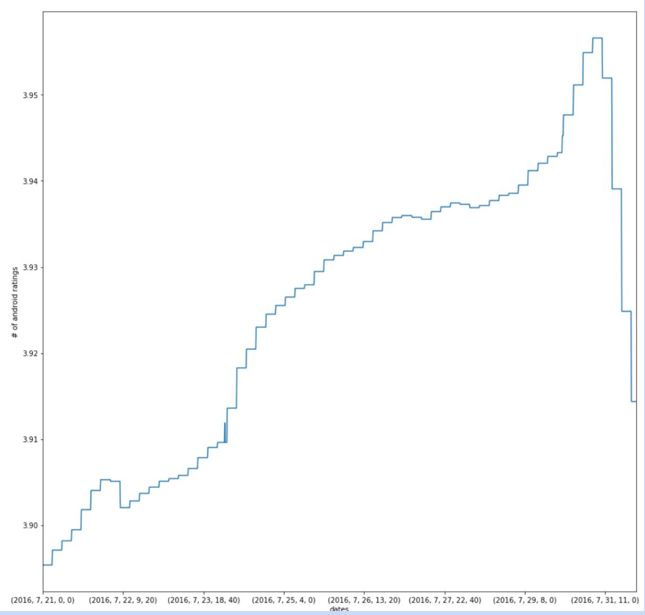


```
[[ 1.          0.24221195]
 [ 0.24221195  1.        ]]
[[ 1.          0.99121962]
 [ 0.99121962  1.        ]]
[[ 1.          0.99976245]
 [ 0.99976245  1.        ]]
[[ 1.          0.93119207]
 [ 0.93119207  1.        ]]
[[ 1.         -0.98449226]
 [-0.98449226  1.        ]]
```

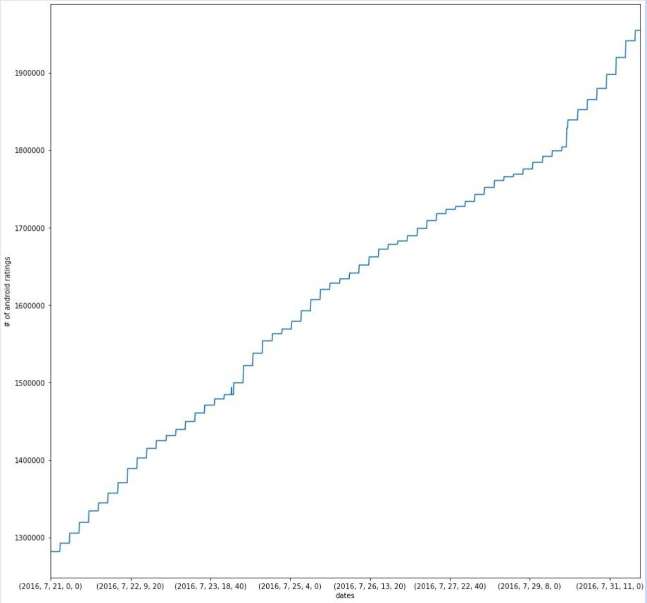
While the pairs observed in this set seem normal, it is interesting to see that there is a strong negative correlation between Android File Size and iOS file size. Despite the limited scope from the data observed, we recognize that there are operating system priorities for the developer. For example, optimization of the app. If platform is optimized (reduction in file size), the other is not.

2.4 Time series graphs

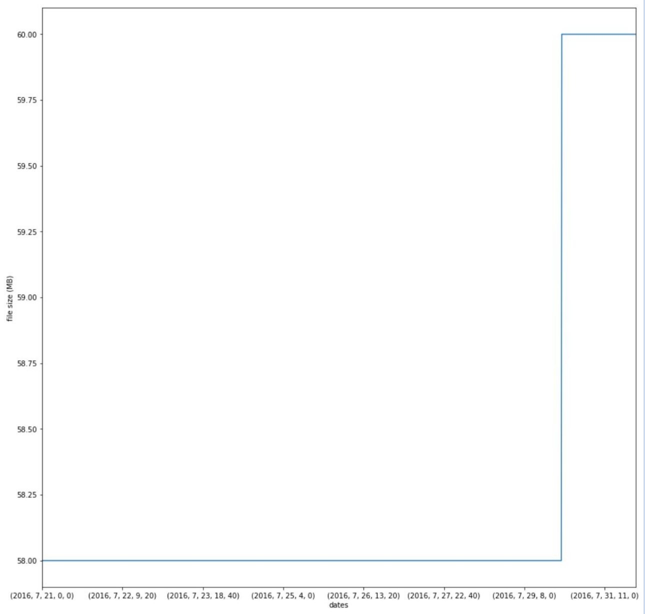
Android_avg_rating



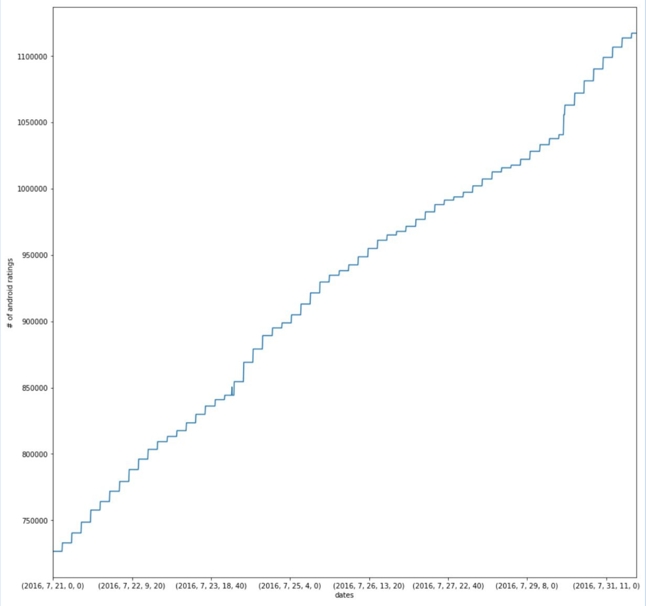
Android_total_ratings



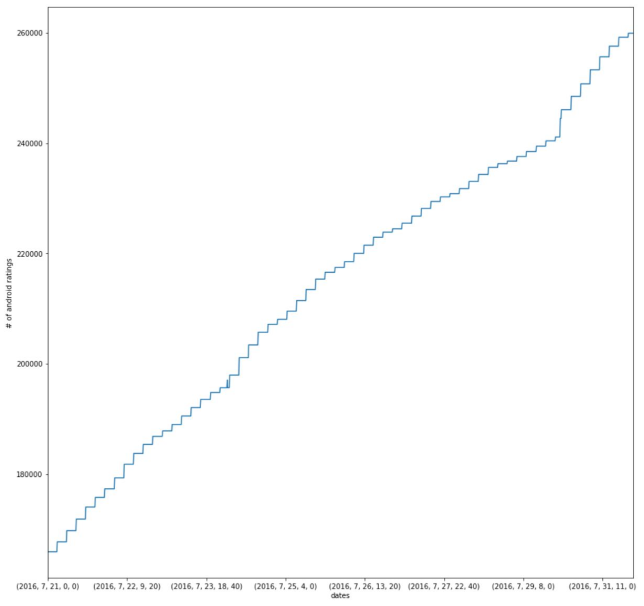
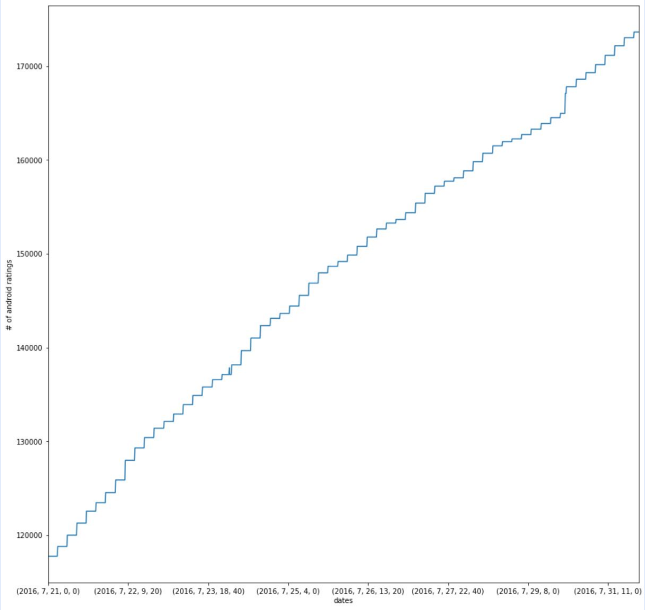
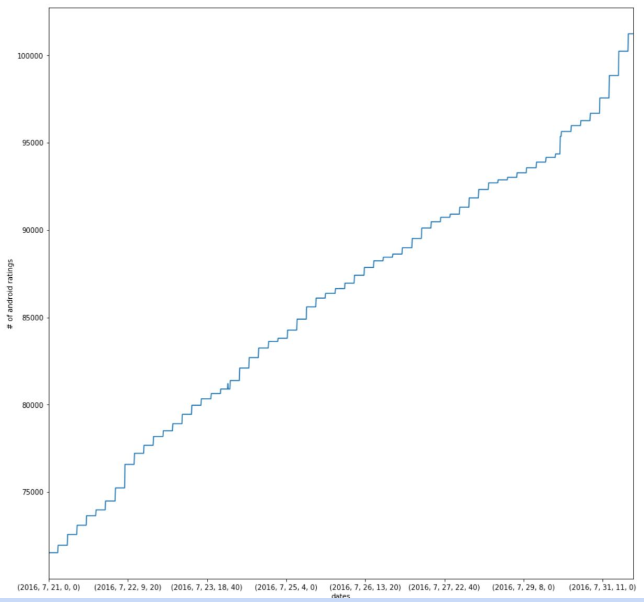
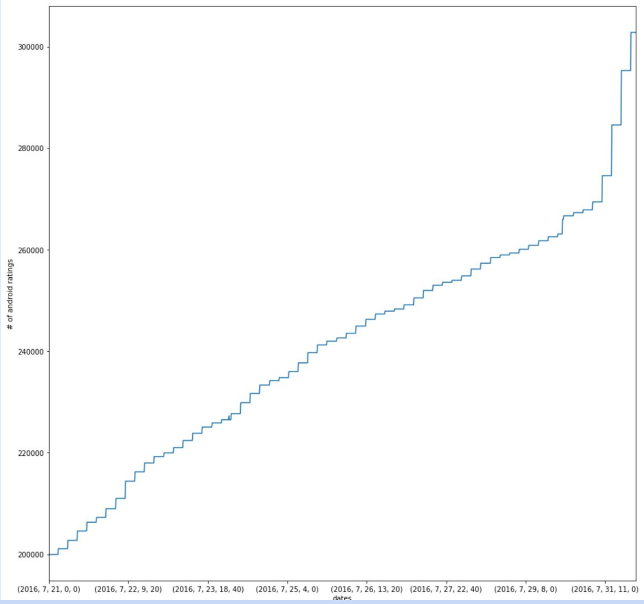
Android_file_size



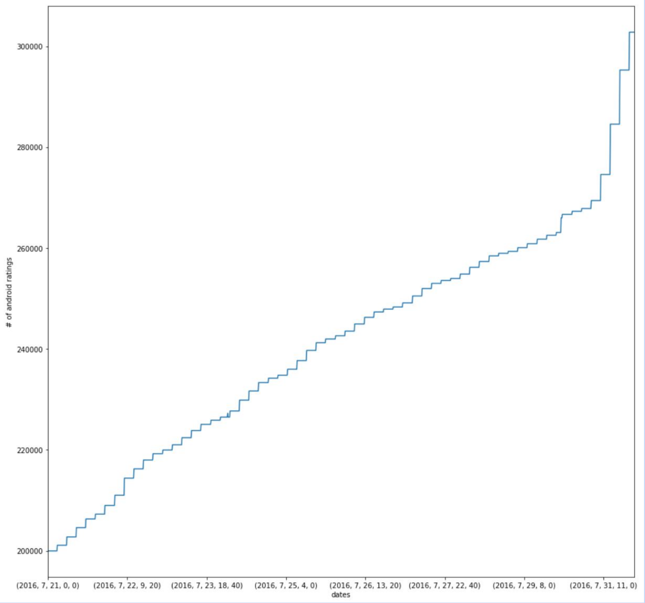
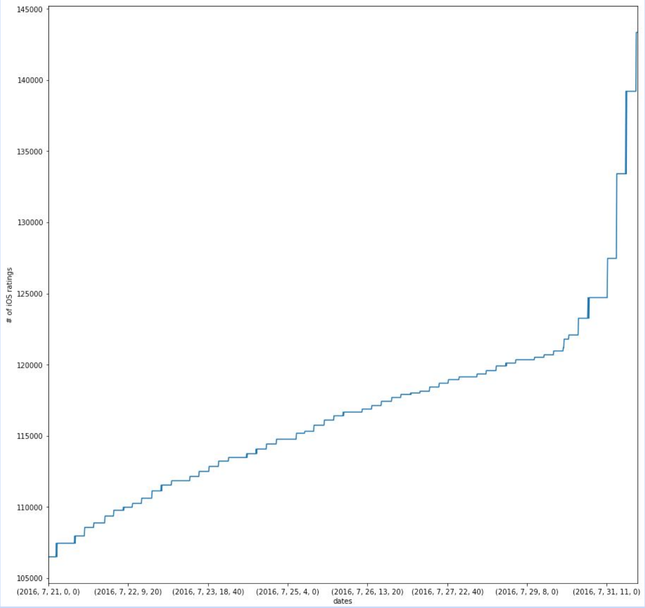
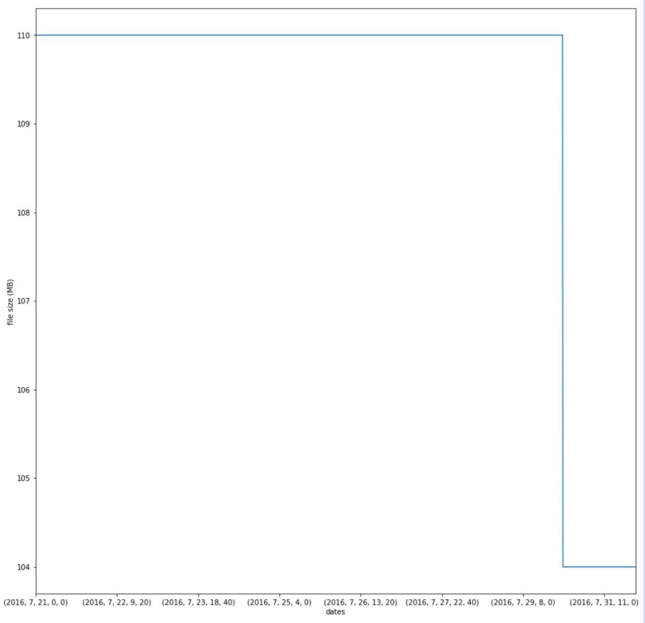
Android_rating_5



Based on the graphs above, we can see that the average android rating continued to increase over the 10 days of data we analyzed. We observed a larger than normal increase of the rating on July 30th, which based on the jump in file size, we can interpret as a bigger version update release. Accordingly, we can conclude the updates made were to users' liking, hence the increase in app rating.

<table>
<tr><th>Android_rating_4</th><th>Android_rating_3</th></tr>
<tr><td>
</td>
<td>
</td></tr>
<tr><th>Android_rating_2</th><th>Android_rating_1</th></tr>
<tr><td>
</td>
<td>
</td></tr>
</table>

While the number of 5, 4, 3, 2, and 1 ratings graph *look* generally the same in terms of the increasing number of ratings received, we can observe on the y axis step level the different amount of ratings received per each star review (i.e., over 1.1million 5 star ratings, around 260,000 thousand 4 star ratings, etc.)

ios_current_ratings	ios_all_ratings
	
ios_file_size	
	

It is interesting that on July 31, 2016, both the current ratings and the total ratings significantly increased. It is likely that it is due to the effect of a new update with significant changes. Unfortunately, we did not attain the ratings on this project for this particular day. There are two possible outcomes of this occurrence: the update lead to negative feedback or the it lead to great changes.

After further investigation, it appears that while Niantic updated new features into the app, it led to a surge of issues throughout. Referring to the Reddit thread here: https://i.redd.it/u9l9wgr2tlcx.png, it appears that crucial problems such as the lost of footprint detection disappeared. This is likely to have resulted in the sudden increase in reviews about bug-related problems.