Introduction

WhatsApp is a messaging app created by Brian Acton and Jan Koum in 2009. Currently, it’s owned by Facebook, that acquired the app in 2014. In Brazil, WhatsApp is almost ubiquitous, installed on 99% of Brazilian smartphones.

Last month, Mark Zuckerberg announced that WhatsApp is delivering roughly 100 billion messages a day. With this huge amount of data being created everyday, how can we structure everything in order to extract insights from it?

There’s a interesting [GitHub repository by kurasaiteja](https://github.com/kurasaiteja/Whatsapp-Analysis) that shows a path on how to achieve this objective. In this article, I’ll be presenting what you can find in the repository code, as well as some modifications I made when analyzing a group chat of my graduation course.

Method

First of all, we need to get the data we will analyze. This can be done quickly, you just need to open the group you want to analyze, go to Settings and then Export Chat, choose to export without media.

To process the data, we’ll import a bunch of libraries, you can find the full code in the references. I’ll highlight some of them as we use. Now that you got a .txt with the messages from a group, pay attention to the format of the lines, because it can vary. My messages are formatted like this:

[11/3/19 10:47:48 AM] John Doe: Hello world

Each line can be broken in specific blocks, as follows.

[{DATE} {TIME} {PERIOD}] {AUTHOR}: {MESSAGE}

This format may vary if you extracted the data using an Android device or a device with iOS, and even between devices with the same OS. One feature from kurasaiteja’s code involves two functions that detects if the line starts with date and time format from Android or iOS. In order to achieve this, it uses Regex to match the different formats.

def startsWithDateAndTimeios(s):  
 pattern = '^\[([0-9]+)([\/-])([0-9]+)([\/-])([0-9]+)[,]? ([0-9]+):([0-9][0-9]):([0-9][0-9])?[ ]?(AM|PM|am|pm)?\]'   
 result = re.match(pattern, s)  
 if result:  
 return True  
 return False

Another function is responsible to detect if there is an author to this message. But why wouldn’t it? Some warnings, like when someone leaves the chat or changes their number, are shown as messages, but they don’t have an author.

There area two functions to extract the data from each line, one for each OS. I’ll keep showing the examples with the function for iOS, for this is the version I used.

First, we break the line at “]”. The date and time will be before the character. Some formats separates date and time with a comma, mine doesn’t. In this case the code would split date and time at a space character, but here we have a problem.

date, time = dateTime.split(' ')

This line threw an error, because when you split this way, the result has three elements: date, time and period. To fix this, we’ll limit the split to only one split adding a second parameter.

date, time = dateTime.split(' ', 1)

There is a function responsible to convert the date string to a datetime object. I made a little adjustment there too, changing the format from “%d-%m-%y” to “%m/%d/%y”. Be careful to find the correct line that needs to be changed, because there are some conditionals inside the function to detect if the date uses dashes or slashes and whether it writes the year with 2 or 4 digits.

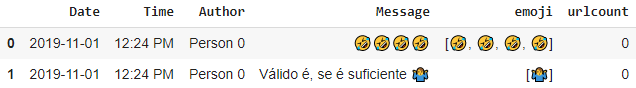
After further processing, a dataframe will be assembled with the columns Date, Time, Author, Message, emoji and urlcount. If you are going to publish the data you are using, you’ll want to hide personal information. Because of this, the Author column is a little problematic and probably will be full of phone numbers.

The original code doesn’t give any method to anonymize this data, so I added a simple way to hide these infos. We’ll create a dictionary to map each author to a generic name like “Person 1”, “Person 2”, etc. Then, we’ll replace each author with its respective generic name. This can be done with one line of code, which I’ll break in two for better readability.

anon\_authors = {key: f'Person {value}' for value, key in enumerate(df.Author.unique())}df.Author.replace(anon\_authors, inplace=True)

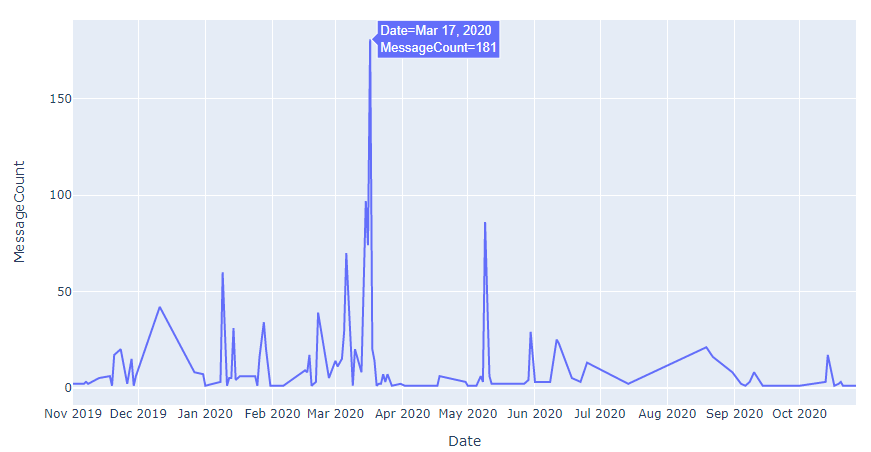
After all of this processing, you’ll have a dataframe that looks like this:

Image for post



Now you can use this dataframe to create analysis and visualizations. The original code has a lot of good examples of things that can be done. One of them is plot with the amount of messages sent each day. Here’s the one I generated with the data from my graduation course group chat:

Image for post



Notice that there is a huge peak on March 17th. That’s the day that was announced that the classes would stop because of COVID-19, a good reason for extra messages.

Conclusion

When you structure the data of messages from a group chat, you can extract insights that are very useful. For example, if you manage a community on WhatsApp, you can see when is the group more active, who is more engaged with the group, what topics are more discussed, etc. These analysis will give you more information to be a better manager.

Anyway, it’s a fun thing to check with your friends group chat and, if you won’t share it with anyone, you can skip the anonymization step and get some author wise analysis.

References

Kurasaiteja’s Original Repository  
<https://github.com/kurasaiteja/Whatsapp-Analysis>

IPython Notebook With My Changes  
[https://github.com/edusrmt/one-figure-projects/](https://github.com/edusrmt/one-figure-projects/blob/master/Project%20%233.ipynb)