
whatstk

lucasrodes

Jan 24, 2021

CONTENTS

1	First contact with whatstk	3
2	Installation & compatibility	5
2.1	From source	5
2.2	Develop	5
3	Support	7
4	Why this name, whatstk?	9
5	Content:	11
5.1	About whatstk	11
5.2	Getting started	12
5.3	Code examples	18
5.4	API Reference	24
5.5	Why choose whatstk?	55
5.6	Community & Governance	56
5.7	Contribute	56
5.8	Changelog	57
	Python Module Index	59
	Index	61

whatstk is a python package providing tools to parse, analyze and visualize WhatsApp chats developed under the [sociepy](#) project. Easily convert your chats to csv or simply visualize statistics using the python library. The package uses [pandas](#) to process the data and [plotly](#) to visualise it.

The project is distributed under the [GPL-3.0 license](#) and is available on [GitHub](#).

FIRST CONTACT WITH WHATSTK

whatstk is built around *BaseChat* object interface, which requires class method *from_source* to be implemented. This method loads and parses the source chat file into a `pandas.DataFrame`.

Below, we use the WhatsApp implementation, i.e. *WhatsAppChat*, to load **LOREM chat**. To test it with your own chat, simply export it as a txt file to your computer and then use class argument `filepath`, as shown in the following example.

```
>>> from whatstk.whatsapp.objects import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM)
>>> chat.df.head(5)
```

date	username	message
2020-01-15 02:22:56	Mary	Nostrud exercitation magna id.
2020-01-15 03:33:01	Mary	Non elit irure irure pariatur exercitation.
2020-01-15 04:18:42	+1 123 456 789	Exercitation esse lorem reprehenderit ut ex ve...
2020-01-15 06:05:14	Giuseppe	Aliquip dolor reprehenderit voluptate dolore e...
2020-01-15 06:56:00	Mary	Ullamco duis et commodo exercitation.

INSTALLATION & COMPATIBILITY

This project is on [PyPI](#), install it with pip:

```
pip install whatstk
```

Project has been tested with Python 3.7-3.8.

2.1 From source

Clone the project from the [official repository](#)

```
git clone https://github.com/lucasrodes/whatstk.git
```

and install it locally

```
cd whatstk  
pip install .
```

2.2 Develop

You can also install the version in development directly from github [develop](#) branch.

```
pip install  
pip install git+https://github.com/lucasrodes/whatstk.git@develop
```


SUPPORT

You can ask questions and join the development discussion on [Gitter](#). Use the [GitHub issues](#) section to report bugs or request features. You can also check the [project roadmap](#).

For more details, refer to the *[contribute section](#)*.

WHY THIS NAME, WHATSTK?

whatstk stands for “WhatsApp Toolkit”, since the project was initially conceived as a python library to read and process WhatsApp chats.

CONTENT:

5.1 About whatstk

whatstk is a python package providing tools to parse, analyze and visualize WhatsApp chats developed under the [sociepy](#) project. Easily convert your chats to csv or simply visualize statistics using the python library. The package uses [pandas](#) to process the data and [plotly](#) to visualise it.

The project is distributed under the [GPL-3.0 license](#) and is available on [GitHub](#).

5.1.1 First contact with whatstk

whatstk is built around *BaseChat* object interface, which requires class method *from_source* to be implemented. This method loads and parses the source chat file into a `pandas.DataFrame`.

Below, we use the WhatsApp implementation, i.e. *WhatsAppChat*, to load **LOREM chat**. To test it with your own chat, simply export it as a txt file to your computer and then use class argument `filepath`, as shown in the following example.

```
>>> from whatstk.whatsapp.objects import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM)
>>> chat.df.head(5)
```

date	username	message
2020-01-15 02:22:56	Mary	Nostrud exercitation magna id.
2020-01-15 03:33:01	Mary	Non elit irure irure pariatur exercitation.
2020-01-15 04:18:42	+1 123 456 789	Exercitation esse lorem reprehenderit ut ex ve...
2020-01-15 06:05:14	Giuseppe	Aliquip dolor reprehenderit voluptate dolore e...
2020-01-15 06:56:00	Mary	Ullamco dui et commodo exercitation.

5.1.2 Installation & compatibility

This project is on [PyPI](#), install it with pip:

```
pip install whatstk
```

Project has been tested with Python 3.7-3.8.

From source

Clone the project from the [official repository](#)

```
git clone https://github.com/lucasrodes/whatstk.git
```

and install it locally

```
cd whatstk  
pip install .
```

Develop

You can also install the version in development directly from github [develop](#) branch.

```
pip install  
pip install git+https://github.com/lucasrodes/whatstk.git@develop
```

5.1.3 Support

You can ask questions and join the development discussion on [Gitter](#). Use the [GitHub issues](#) section to report bugs or request features. You can also check the [project roadmap](#).

For more details, refer to the [contribute section](#).

5.1.4 Why this name, whatstk?

whatstk stands for “WhatsApp Toolkit”, since the project was initially conceived as a python library to read and process WhatsApp chats.

5.2 Getting started

Getting started with the library is fairly easy.

5.2.1 Export a WhatsApp chat

Exporting a WhatsApp chat can be easily done from your Android or iOS device. It is done on a chat basis, so if you want to export several chats you will have to export them individually. **When exporting, make sure to select the chats Without Media option.** Once generated, you can send it via mail, so you can save it in your computer.

Android

The export on Android might include several files. We are only interested in the text file (i.e. `.txt` extension file).

Fig. 1: Android 9, WhatsApp v2.20.123

For more details, refer to [official website](#).

iOS

The chat is exported as a [zip](#), which can be easily unzipped in your computer.

Fig. 2: iOS 12, WhatsApp v2.20.31

5.2.2 Load chat

Once you have *exported* a chat it is time to load it in python.

In this example we load the example [LOREM chat](#), which is available online, using library class `WhatsAppChat`.

```
>>> from whatstk import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM)
```

Once loaded, we can check some of the chat messages by accessing its attribute `df`, which is a `pandas.DataFrame` with columns `username` (name of user sending the message), `message` (message sent) and `date` index (timestamp of message).

```
>>> chat.df.head(5)
      date                username  message
↳message
date
2020-01-15 02:22:56          Mary  Nostrud exercitation_
↳magna id.
2020-01-15 03:33:01          Mary  Non elit irure irure pariatur_
↳exercitation.
2020-01-15 04:18:42  +1 123 456 789  Exercitation esse lorem reprehenderit ut ex_
↳ve...
2020-01-15 06:05:14      Giuseppe  Aliquip dolor reprehenderit voluptate dolore_
↳e...
2020-01-15 06:56:00          Mary  Ullamco duis et commodo_
↳exercitation.
```

Getting the start and end date of the chat can give us a good overview of the chat content.

```
>>> print(f"Start date: {chat.start_date}\nEnd date: {chat.end_date}")
Start date: 2020-01-15 02:22:56
End date: 2020-05-11 22:32:48
```

Also, getting a list with the chat members is simple

```
>>> chat.users
['+1 123 456 789', 'Giuseppe', 'John', 'Mary']
```

See also:

- [Load chat from multiple sources](#)
- [Load a chat with specific hformat](#)

5.2.3 Command line

whatstk provides a set of command line tools to obtain quick results using the command line. To use these, make sure that you have previously [installed the library](#).

For instance, convert a WhatsApp text file to a CSV file using

```
whatstk-to-csv [input_filename] [output_filename]
```

For more details, check the [command line tools documentation](#).

5.2.4 The header format

In WhatsApp, a chat file syntax can differ between devices, OS and language settings, which makes it hard to correctly parse the data for all formats.

The header appears for each message sent in the chat and contains the timestamp and name of the user that sent the message.

See it for yourself and open [an exported chat file](#). You will find that the messages have a similar format like the one below:

```
15.04.2016, 15:04 - You created group "Sample Group"
06.08.2016, 13:18 - Messages you send to this group are now secured with end-to-end_
↳ encryption. Tap for more info.
06.08.2016, 13:23 - Ash Ketchum: Hey guys!
06.08.2016, 13:25 - Brock: Hey Ash, good to have a common group!
06.08.2016, 13:30 - Misty: Hey guys! Long time haven't heard anything from you
06.08.2016, 13:45 - Ash Ketchum: Indeed. I think having a whatsapp group nowadays is_
↳ a good idea
06.08.2016, 14:30 - Misty: Definetly
06.08.2016, 17:25 - Brock: I totally agree
07.08.2016, 11:45 - Prof. Oak: Kids, shall I design a smart poke-ball?
```

In this example, the header is **day.month.year, hour:minutes - username:** which corresponds to the header format (a.k.a. **hformat**) '%d.%m.%y, %H:%M - %name:'. However, in your case it may be slightly different depending on you phone settings.

Check the table below to see the codes for each header format unit:

Table 1: header format units

Date unit code	Description
'%Y'	Year
'%m'	Month of the year (1-12)
'%d'	Day of the month (0-31)
'%H'	Hour 24h-clock (0-23)
'%P'	Hour 12h-clock (1-12)
'%M'	Minutes (0-60)
'%S'	Seconds (0-60)
'%name'	Name of user

See also:*Loading chat using hformat*

5.2.5 Library available chats

For the purpose of showcasing code examples and benchmarking different implementations, we have created a pool of chats, hosted in the [official repository page](#). If you want to test the library with one of your own tests, check in the *code examples*.

The chats are available via their corresponding URLs, which are listed in source code `whatstk.data`.

Contents

- *Library available chats*
 - *WhatsApp*
 - * *POKEMON*
 - * *LOREM*
 - * *LOREM1*
 - * *LOREM2*
 - * *LOREM_2000*

WhatsApp

Object `whatsapp_urls` contains all URLs for WhatsApp chats.

```
>>> from whatstk.data import whatsapp_urls
```

POKEMON

Brief fictional chat with Pokemon characters, which was manually designed by [@lucasrodes](#) in commit [666d6ea9cc030c4322fbe44ae64b8f1a0fdb5169](#).

```
>>> from whatstk.data import whatsapp_urls
>>> from whatstk import WhatsAppChat
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.POKEMON)
>>> chat.df.head(5)
```

date	username	message
2016-08-06 13:23:00	Ash Ketchum	Hey guys!
2016-08-06 13:25:00	Brock	Hey Ash, good to have a common group!
2016-08-06 13:30:00	Misty	Hey guys! Long time haven't heard anything fro...
2016-08-06 13:45:00	Ash Ketchum	Indeed. I think having a whatsapp group nowada...
2016-08-06 14:30:00	Misty	Definetly

See also:

[Chat file](#)

LOREM

Chat with 500 interventions of fictional users, generated using [python-lorem](#) library.

```
>>> from whatstk.data import whatsapp_urls
>>> from whatstk import WhatsAppChat
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM)
>>> chat.df.head(5)
```

date	username	message
2020-01-15 02:22:56	Mary	Nostrud exercitation magna id.
2020-01-15 03:33:01	Mary	Non elit irure irure pariatur exercitation.
2020-01-15 04:18:42	+1 123 456 789	Exercitation esse lorem reprehenderit ut ex ve...
2020-01-15 06:05:14	Giuseppe	Aliquip dolor reprehenderit voluptate dolore e...
2020-01-15 06:56:00	Mary	Ullamco dui et commodo exercitation.

See also:

[Chat file](#)

LOREM1

Chat with 300 interventions of fictional users, generated using [python-lorem](#).

```
>>> from whatstk.data import whatsapp_urls
>>> from whatstk import WhatsAppChat
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM1)
>>> chat.df.head(5)
```

date	username	message
2019-10-20 10:16:00	John	Laborum sed excepteur id eu cillum sunt ut.
2019-10-20 11:15:00	Mary	Ad aliquip reprehenderit proident est irure mo...
2019-10-20 12:16:00	+1 123 456 789	Nostrud adipiscing ex enim reprehenderit minim...
2019-10-20 12:57:00	+1 123 456 789	Deserunt proident laborum exercitation ex temp...
2019-10-20 17:28:00	John	Do ex dolor consequat tempor et ex.

See also:[Chat file](#)**LOREM2**

Chat with 300 interventions of fictional users, generated using [python-lorem](#).

Can be used along with **LOREM1** to test *chat merging functionalities* or *multiple-source loading*.

```
>>> from whatstk.data import whatsapp_urls
>>> from whatstk import WhatsAppChat
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM2)
>>> chat.df.head(5)
```

date	username	message
2020-06-20 10:16:00	John	Elit incididunt lorem sed nostrud.
2020-06-20 11:15:00	Maria	Esse do irure dolor tempor ipsum fugiat.
2020-06-20 12:16:00	+1 123 456 789	Cillum anim non eu deserunt consectetur dolor ...
2020-06-20 12:57:00	+1 123 456 789	Non ipsum proident veniam est.
2020-06-20 17:28:00	John	Dolore in cupidatat proident.

See also:[Chat file](#)**LOREM_2000**

Chat with 2000 interventions of fictional users, generated using [python-lorem](#).

```
>>> from whatstk.data import whatsapp_urls
>>> from whatstk import WhatsAppChat
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM_2000)
>>> chat.df.head(5)
```

date	username	message
2019-04-16 02:09:00	+1 123 456 789	Et labore proident laboris do labore ex.
2019-04-16 03:01:00	Mary	Reprehenderit id aute consectetur aliquip nost...
2019-04-17 12:56:00	John	Amet magna officia ullamco pariatur ipsum cupi...
2019-04-17 13:30:00	Mary	Cillum aute et cupidatat ipsum, occaecat lorem...
2019-04-17 15:09:00	John	Eiusmod irure laboris dolore anim, velit velit...

See also:[Chat file](#)

For examples refer to *code examples* section.

5.3 Code examples

5.3.1 Basic examples

Load chat

Once you have *exported* a chat it is time to load it in python.

In this example we load the example **LOREM** chat, which is available online, using library class *WhatsAppChat*.

```
>>> from whatstk import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM)
```

Once loaded, we can check some of the chat messages by accessing its attribute *df*, which is a *pandas.DataFrame* with columns *username* (name of user sending the message), *message* (message sent) and *date* index (timestamp of message).

```
>>> chat.df.head(5)
```

	username	
↪message		
date		
2020-01-15 02:22:56	Mary	Nostrud exercitation
↪magna id.		
2020-01-15 03:33:01	Mary	Non elit irure irure pariatur
↪exercitation.		
2020-01-15 04:18:42	+1 123 456 789	Exercitation esse lorem reprehenderit ut ex
↪ve...		
2020-01-15 06:05:14	Giuseppe	Aliquip dolor reprehenderit voluptate dolore
↪e...		
2020-01-15 06:56:00	Mary	Ullamco duis et commodo
↪exercitation.		

Getting the start and end date of the chat can give us a good overview of the chat content.

```
>>> print(f"Start date: {chat.start_date}\nEnd date: {chat.end_date}")
Start date: 2020-01-15 02:22:56
End date: 2020-05-11 22:32:48
```

Also, getting a list with the chat members is simple

```
>>> chat.users
['+1 123 456 789', 'Giuseppe', 'John', 'Mary']
```

Load chat from multiple sources

You can also load a chat using multiple source files. You might want to use this when several files have been exported from the same chat over the years.

In the example below, we load chats **LOREM1** and **LOREM2**.

```
>>> from whatstk import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_sources(filepaths=[whatsapp_urls.LOREM1, whatsapp_urls.
↪LOREM2])
```

Rename usernames

In the example here, chat **LOREM1** and chat **LOREM2** contain slightly different usernames. In particular, in chat **LOREM2**, user *Mary* appears as *Maria* and *Maria2*:

```
>>> WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM1).users
['+1 123 456 789', 'Giuseppe', 'John', 'Mary']
>>> WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM2).users
['+1 123 456 789', 'Giuseppe', 'John', 'Maria', 'Maria2']
>>> >>> chat.users
['+1 123 456 789', 'Giuseppe', 'John', 'Maria', 'Maria2', 'Mary']
```

To draw some conclusions based on user behaviour we would like to group *Mary*, *Maria* and *Maria2* under the same username. To fix this, we rename *Maria* and *Maria2* as *Mary*:

```
>>> chat = chat.rename_users({'Mary': ['Maria', 'Maria2']})
>>> chat.users
['+1 123 456 789', 'Giuseppe', 'John', 'Mary']
```

Load a chat with specific hformat

If `auto_header` option fails, you can still load your chat manually specifying the `hformat`. In the example below, we have that the `hformat='%d.%m.%y, %H:%M - %name:.'`

```
>>> from whatstk.whatsapp.objects import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.POKEMON, hformat='%d.%m.%y,
↳ %H:%M - %name:')
>>> chat.df.head(5)
```

	username	message
date		
2016-08-06 13:23:00	Ash Ketchum	Hey guys!
2016-08-06 13:25:00	Brock	Hey Ash, good to have a common group!
2016-08-06 13:30:00	Misty	Hey guys! Long time haven't heard anything fro...
2016-08-06 13:45:00	Ash Ketchum	Indeed. I think having a whatsapp group nowada...
2016-08-06 14:30:00	Misty	Definetly

See also:

The header format

5.3.2 Visualisations

With *FigureBuilder* you can get great insights from your chat. Below we provide some examples on the visualizations that you can get with this library with the help of *plotly*.

Counting user interventions

Counting the user interventions can give relevant insights on which users “dominate” the conversation, even more in a group chat. To this end, object *FigureBuilder* has the method *user_interventions_count_linechart*, which generates a plotly figure with the count of user interventions.

First of all, we load a chat and create an instance of *FigureBuilder*.

```
>>> from whatstk import WhatsAppChat, FigureBuilder
>>> from whatstk.graph import plot
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM_2000)
>>> fb = FigureBuilder(chat=chat)
```

Count of user interventions

Default call of the aforementioned method displays the number of interventions sent by each user per day.

```
>>> fig = fb.user_interventions_count_linechart()
>>> plot(fig)
```

As seen in previous plot, the number of messages sent per user in a day tends to oscilate quite a lot from day to day, which might difficult a good visualisation of the data. Hence, we can use *cumulative=True* to illustrate the cumulative count of interventions instead.

```
>>> fig = fb.user_interventions_count_linechart(cumulative=True, title='User_
↳ interventions count (cumulative)')
>>> plot(fig)
```

Additionally, we can obtain the counts for all users combined using *all_users=True*:

```
>>> fig = fb.user_interventions_count_linechart(cumulative=True, all_users=True,
↳ title='Inteventions count (cumulative)')
>>> plot(fig)
```

Count of characters sent per user

Now, instead of counting the number of interventions we might want to explore the number of characters sent. Note that a user might send tons of messages with few words, whereas another user might send few messages with tons of words. Depending on your analysis you might prefer exploring interventions or number of characters. Getting the number of characters sent per user can be done using *msg_len=True* when calling function *user_interventions_count_linechart*.

In the following we explore the cumulative number of characters sent per user.

```
>>> fig = fb.user_interventions_count_linechart(msg_len=True, cumulative=True, title=
↳ 'Characters sent by user (cumulative)')
>>> plot(fig)
```


Other insights

Method `user_interventions_count_linechart` has the argument `date_mode`, which allows for several types of count-grouping methods. By default, the method obtains the counts per date (what has been used in previous examples).

Using `date_mode=hour` illustrates the distribution of user interventions over the 24 hours in a day. In this example, for instance, Giuseppe has their interventions peak in hour ranges [01:00, 02:00] and [20:00, 21:00], with 21 interventions in each.

```
>>> fig = fb.user_interventions_count_linechart(date_mode='hour', title='User_
↳ interventions count (hour)',
xlabel='Hour')
>>> plot(fig)
```

Using `date_mode=weekday` illustrates the distribution of user interventions over the 7 days of the week. In this example, for instance, we see that Monday and Sunday are the days with the most interventions.

```
>>> fig = fb.user_interventions_count_linechart(date_mode='weekday', title='User_
↳ interventions count (weekly)',
xlabel='Week day')
>>> plot(fig)
```

Using `date_mode=month` illustrates the distribution of user interventions over the 12 months of the year. In this example, for instance, we observe that all users have their interventions peak in June (except for Giuseppe, which has their peak in July). Maybe summer calling?

```
>>> fig = fb.user_interventions_count_linechart(date_mode='month', title='User_
↳ interventions count (yearly)', xlabel='Month')
>>> plot(fig)
```

Message length boxplot

Different users send different sort of messages. In particular, the length of the messages (number of characters) can substantially vary depending on the user sending the message.

In this example, we explore the statistics behind the length of user messages. To this end, we can use method `user_msg_length_boxplot`, which illustrates the length of each user's messages by means of **box plots**.

```
>>> from whatstk import WhatsAppChat, FigureBuilder
>>> from whatstk.graph import plot
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM_2000)
>>> fig = FigureBuilder(chat=chat).user_msg_length_boxplot()
>>> plot(fig)
```

User interaction

The user interaction can shed some light on the different kinds of conversations that occur in a chat group. For instance, when a certain topic appears some users might intervene and others will not, forming *user clusters*. To this end, a first approach in detecting such clusters resides in which users respond to which users.

User interaction heatmap

In the following we visualize the *response matrix*, which tells us the number of messages sent by a certain user to the rest of users.

For instance, in this specific example we observe that user *Giuseppe* sends 153 messages to +1 123 456 789 and that *Mary* receives 122 messages from *John*.

```
>>> from whatstk import WhatsAppChat, FigureBuilder
>>> from whatstk.graph import plot
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM_2000)
>>> fig = FigureBuilder(chat=chat).user_message_responses_heatmap()
>>> plot(fig)
```

See also:

- `user_message_responses_heatmap`

User interaction flow

A good way to visualize responses between users are [Sankey diagrams](#). The information conveyed by the graph below is the same as the one in previous section, but the way it is done is slightly different (sankey diagram instead of a heatmap).

```
>>> from whatstk import WhatsAppChat, FigureBuilder
>>> from whatstk.graph import plot
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM_2000)
>>> fig = FigureBuilder(chat=chat).user_message_responses_flow()
>>> plot(fig)
```

See also:

- `user_message_responses_flow`

Custom plot

`FigureBuilder` provides some tools to easily visualize your chat. However, the possible visualizations are infinite. Here, we provide some examples of a custom visualization using some library tools together with pandas and plotly.

Number of messages vs. Number of characters sent

For each user, we will obtain a 2D scatter plot measuring the number of messages and characters sent in a day. That is, for a given user we will have N points, where N is the number of days that the user has sent at least one message. Each point therefore corresponds to a specific day, where the x-axis and the y-axis measure the number of messages sent and the average number of characters per message in that day, respectively.

First of all, let's instantiate objects *WhatsAppChat* (chat loading) and *FigureBuilder* (figure coloring).

```
>>> from whatstk import WhatsAppChat, FigureBuilder
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM_2000)
>>> fb = FigureBuilder(chat=chat)
```

Next, we obtain the number of messages and number of characters sent per user per day.

```
>>> from whatstk.analysis import get_interventions_count
>>> counts_interv = get_interventions_count(chat=chat, date_mode='date', msg_
↳ length=False, cumulative=False)
>>> counts_len = get_interventions_count(chat=chat, date_mode='date', msg_length=True,
↳ cumulative=False)
```

Time to process a bit the data. We obtain a DataFrame with five columns: *username*, *date*, *num_characters*, *num_interventions* and *avg_characters*.

```
>>> import pandas as pd
>>> counts_len = pd.DataFrame(counts_len.unstack(), columns=['num_characters'])
>>> counts_interv = pd.DataFrame(counts_interv.unstack(), columns=['num_interventions
↳ '])
>>> counts = counts_len.merge(counts_interv, left_index=True, right_index=True)
>>> # Remove all zero entries and get average number of characters
>>> counts = counts[~(counts['num_interventions'] == 0)].reset_index()
>>> counts['avg_characters'] = counts['num_characters']/counts['num_interventions']
>>> counts.head(5)
```

	username	date	num_characters	num_interventions	avg_characters
0	+1 123 456 789	2019-04-16	40	1	40.000000
1	+1 123 456 789	2019-04-17	21	1	21.000000
2	+1 123 456 789	2019-04-21	90	2	45.000000
3	+1 123 456 789	2019-04-25	127	3	42.333333
4	+1 123 456 789	2019-04-26	33	1	33.000000

```
[5 rows x 5 columns]
```

So far we have obtained a dataframe *counts*, whose rows correspond to a specific message. However, in this example we are interested in the aggregated values per day. Hence, we group this dataframe by user and date and re-calculate the number of messages sent and average number of characters sent per day.

```
>>> agg_operations = {'avg_characters': 'mean', 'num_interventions': 'mean'}
>>> counts = counts.groupby(['username', counts.date.dt.date]).agg(agg_operations)
>>> counts = counts.rename_axis(index=['username', 'date'])
>>> counts = counts.reset_index()
>>> counts.head(5)
```

	username	date	avg_characters	num_interventions
0	+1 123 456 789	2019-04-16	40.000000	1
1	+1 123 456 789	2019-04-17	21.000000	1
2	+1 123 456 789	2019-04-21	45.000000	2
3	+1 123 456 789	2019-04-25	42.333333	3
4	+1 123 456 789	2019-04-26	33.000000	1

Once the dataframe is obtained, we generate a plot using `Histogram2dContour` by `plotly`.

```
>>> from whatstk.graph import plot
>>> import plotly.graph_objs as go
>>> traces = []
>>> for username in fb.usernames:
>>>     counts_user = counts[counts['username']==username]
>>>     traces.append(
>>>         go.Histogram2dContour(
>>>             contours={'coloring': 'none'},
>>>             x=counts_user['num_interventions'],
>>>             y=counts_user['avg_characters'],
>>>             # mode='markers',
>>>             # marker=dict(color=fb.user_color_mapping[username], opacity=0.2),
>>>             name=username,
>>>             showlegend=True,
>>>             line={'color': fb.user_color_mapping[username]},
>>>             nbinsx=10, nbinsy=20
>>>         )
>>>     )
```

```
>>> layout = {
>>>     'title': 'Average number of characters sent in a day vs Interventions per day',
>>>     'yaxis_title': 'avg characters',
>>>     'xaxis_title': 'num interventions',
>>> }
>>> fig = go.Figure(data=traces, layout=layout)
>>> plot(fig)
```

If you think that something is missing please [raise an issue](#).

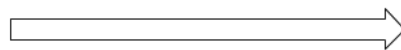
5.4 API Reference

5.4.1 Main objects

WhatsAppChat

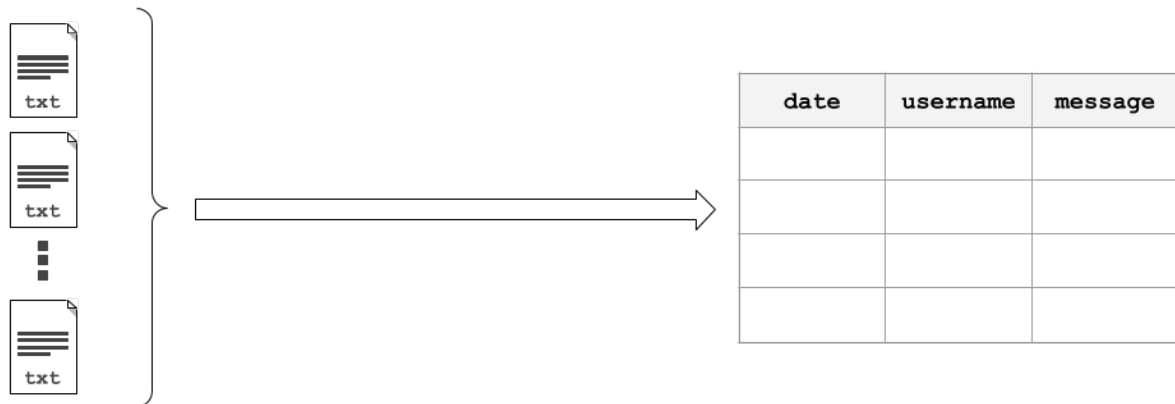
Object `WhatsAppChat` works as a bridge between the python code and the whatsapp chat text file. Easily load a chat from a text file and work with it using all the power of `pandas`.

A chat can be loaded from a single source file using `WhatsAppChat.from_source`



date	username	message

or multiple source files using `WhatsAppChat.from_sources`



class `whatstk.WhatsAppChat` (*df*)
 Bases: `whatstk._chat.BaseChat`
 Load and process a WhatsApp chat file.
Parameters *df* (`pandas.DataFrame`) – Chat.

Attributes

<i>df</i>	Chat as DataFrame.
<i>end_date</i>	Chat end date.
<i>start_date</i>	Chat starting date.
<i>users</i>	List with users.

Methods

<i>from_source</i> (filepath, **kwargs)	Create an instance from a chat text file.
<i>from_sources</i> (filepath[, auto_header, ...])	Load a WhatsAppChat instance from multiple sources.
<i>merge</i> (chat[, rename_users])	Merge current instance with chat.
<i>rename_users</i> (mapping)	Rename users.
<i>to_csv</i> (filepath)	Save chat as csv.
<i>to_txt</i> (filepath[, hformat])	Export chat to a text file.

Example

This simple example loads a chat using `WhatsAppChat`. Once loaded, we can access its attribute `df`, which contains the loaded chat as a DataFrame.

```
>>> from whatstk.whatsapp.objects import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.POKEMON)
>>> chat.df.head(5)
```

	username	message
0		
1		
2		
3		
4		

(continues on next page)

(continued from previous page)

```

date
2016-08-06 13:23:00  Ash Ketchum                      Hey ↵
↵guys!
2016-08-06 13:25:00          Brock                      Hey Ash, good to have a common ↵
↵group!
2016-08-06 13:30:00          Misty  Hey guys! Long time haven't heard anything fro..
↵.
2016-08-06 13:45:00  Ash Ketchum  Indeed. I think having a whatsapp group nowada..
↵.
2016-08-06 14:30:00          Misty                      ↵
↵Definetly

```

property df

Chat as DataFrame.

Returns pandas.DataFrame**property end_date**

Chat end date.

Returns datetime**classmethod from_source** (filepath, **kwargs)

Create an instance from a chat text file.

Parameters

- **filepath** (*str*) – Path to the file. It can be a local file (e.g. 'path/to/file.txt') or an URL to a hosted file (e.g. 'http://www.url.to/file.txt')
- ****kwargs** – Refer to the docs from [df_from_txt_whatsapp](#) for details on additional arguments.

Returns WhatsAppChat – Class instance with loaded and parsed chat.**See also:**

- [df_from_txt_whatsapp](#)
- [WhatsAppChat.from_sources](#)

classmethod from_sources (filepaths, auto_header=None, hformat=None, encoding='utf-8')

Load a WhatsAppChat instance from multiple sources.

Parameters

- **filepaths** (*list*) – List with filepaths.
- **auto_header** (*bool, optional*) – Detect header automatically (applies to all files). If None, attempts to perform automatic header detection for all files. If False, hformat is required.
- **hformat** (*list, optional*) – List with the [header format](#) to be used for each file. The list must be of length equal to len (filenames). A valid header format might be '[%y-%m-%d %H:%M:%S] - %name:'.
- **encoding** (*str*) – Encoding to use for UTF when reading/writing (ex. 'utf-8'). [List of Python standard encodings](#).

Returns WhatsAppChat – Class instance with loaded and parsed chat.**See also:**

- `WhatsAppChat.from_source`
- `merge_chats`

Example

Load a chat using two text files. In this example, we use sample chats (available online, see urls in source code `whatstk.data`).

```
>>> from whatstk.whatsapp.objects import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> filepath_1 = whatsapp_urls.LOREM1
>>> filepath_2 = whatsapp_urls.LOREM2
>>> chat = WhatsAppChat.from_sources(filepaths=[filepath_1, filepath_2])
>>> chat.df.head(5)
```

	username	message
date		
2019-10-20 10:16:00	John	Laborum sed excepteur id eu cillum
2019-10-20 11:15:00	Mary	Ad aliquip reprehenderit proident est
2019-10-20 12:16:00	+1 123 456 789	Nostrud adipiscing ex enim reprehenderit
2019-10-20 12:57:00	+1 123 456 789	Deserunt proident laborum exercitation
2019-10-20 17:28:00	John	Do ex dolor consequat

merge (*chat*, *rename_users=None*)
Merge current instance with chat.

Parameters

- **chat** (`WhatsAppChat`) – Another chat.
- **rename_users** (*dict*) – Dictionary mapping old names to new names. Example: `{ 'John': ['Jon', 'J'], 'Ray': ['Raymond'] }` will map 'Jon' and 'J' to 'John', and 'Raymond' to 'Ray'. Note that old names must come as list (even if there is only one).

Returns `WhatsAppChat` – Merged chat.

See also:

- `rename_users`
- `merge_chats`

Example

Merging two chats can become handy when you have exported a chat in different times with your phone and hence each exported file might contain data that is unique to that file.

In this example however, we merge files from different chats.

```
>>> from whatstk.whatsapp.objects import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> filepath_1 = whatsapp_urls.LOREM1
```

(continues on next page)

(continued from previous page)

```
>>> filepath_2 = whatsapp_urls.LOREM2
>>> chat_1 = WhatsAppChat.from_source(filepath=filepath_1)
>>> chat_2 = WhatsAppChat.from_source(filepath=filepath_2)
>>> chat = chat_1.merge(chat_2)
```

rename_users (*mapping*)

Rename users.

This might be needed in multiple occasions:

- Change typos in user names stored in phone.
- **If a user appears multiple times with different usernames, group these under the same name** (this might happen when multiple chats are merged).

Parameters *mapping* (*dict*) – Dictionary mapping old names to new names, example: {'John': ['Jon', 'J'], 'Ray': ['Raymond']} will map 'Jon' and 'J' to 'John', and 'Raymond' to 'Ray'. Note that old names must come as list (even if there is only one).

Returns *pandas.DataFrame* – DataFrame with users renamed according to *mapping*.

Raises **ValueError** – Raised if mapping is not correct.

ExamplesLoad LOREM2 chat and rename users *Maria* and *Maria2* to *Mary*.

```
>>> from whatstk.whatsapp.objects import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM2)
>>> chat.users
['+1 123 456 789', 'Giuseppe', 'John', 'Maria', 'Maria2']
>>> chat = chat.rename_users(mapping={'Mary': ['Maria', 'Maria2']})
>>> chat.users
['+1 123 456 789', 'Giuseppe', 'John', 'Mary']
```

property start_date

Chat starting date.

Returns *datetime***to_csv** (*filepath*)

Save chat as csv.

Parameters *filepath* (*str*) – Name of file.**to_txt** (*filepath*, *hformat=None*)

Export chat to a text file.

Usefull to export the chat to different formats (i.e. using different hformats).

Parameters

- **filepath** (*str*) – Name of the file to export (must be a local path).
- **hformat** (*str*, *optional*) – Header format. Defaults to '%y-%m-%d, %H:%M - %name:'.

property users

List with users.

Returns list

FigureBuilder

whatstk provides this object to ease the generation of insightful plots from your chat. *FigureBuilder* contains several methods to generate different plots. It assigns a unique color to each user, so that a user can be easily identified in all plots.

To instantiate it, you just need to provide the chat (as `pandas.DataFrame` or *BaseChat*-API-compliant object).

class `whatstk.FigureBuilder` (*df=None, chat=None*)

Bases: `object`

Generate a variety of figures from your loaded chat.

Integrates feature extraction and visualization logic to automate data plots.

Note: Either `df` or `chat` must be provided.

Parameters

- **df** (*pandas.DataFrame, optional*) – Chat data. Attribute *df* of a chat loaded using Chat. If a value is given, `chat` is ignored.
- **chat** (*Chat, optional*) – Chat data. Object obtained when chat loaded using Chat. Required if `df` is `None`.

Attributes

<code>user_color_mapping</code>	Get mapping between user and color.
<code>usernames</code>	Get list with users available in given chat.

Methods

<code>user_interventions_count_linechart([...])</code>	Plot number of user interventions over time.
<code>user_message_responses_flow([title])</code>	Get the flow of message responses.
<code>user_message_responses_heatmap([norm, title])</code>	Get the response matrix heatmap.
<code>user_msg_length_boxplot([title, xlabel])</code>	Generate figure with boxplots of each user's message length.

property `user_color_mapping`

Get mapping between user and color.

Each user is assigned a color automatically, so that this color is preserved for that user in all to-be-generated plots.

Returns *dict* – Mapping from username to color (rgb).

user_interventions_count_linechart (*date_mode='date', msg_length=False, cumulative=False, all_users=False, title='User interventions count', xlabel='Date/Time', cumulative=None*)

Plot number of user interventions over time.

Parameters

- **date_mode** (*str, optional*) – Choose mode to group interventions by. Defaults to 'date'. Available modes are:

- 'date': Grouped by particular date (year, month and day).
- 'hour': Grouped by hours.
- 'month': Grouped by months.
- 'weekday': Grouped by weekday (i.e. monday, tuesday, ..., sunday).
- 'hourweekday': Grouped by weekday and hour.
- **msg_length** (*bool, optional*) – Set to True to count the number of characters instead of number of messages sent.
- **cumulative** (*bool, optional*) – Set to True to obtain commulative counts.
- **all_users** (*bool, optional*) – Obtain number of interventions of all users combined. Defaults to False.
- **title** (*str, optional*) – Title for plot. Defaults to “User interventions count”.
- **xlabel** (*str, optional*) – x-axis label title. Defaults to “Date/Time”.
- **cummulative** (*bool, optional*) – Deprecated, use cumulative.

Returns *plotly.graph_objs.Figure* – Plotly Figure.

See also:

- *get_interventions_count*
- *fig_scatter_time*

Example

```
>>> from whatstk import WhatsAppChat
>>> from whatstk.graph import plot, FigureBuilder
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM)
>>> fig = FigureBuilder(chat=chat).user_interventions_count_
↳ linechart(cumulative=True)
>>> plot(fig)
```

user_message_responses_flow (*title='Message flow'*)

Get the flow of message responses.

A response from user X to user Y happens if user X sends a message right after a message from user Y.

Uses a Sankey diagram.

Parameters **title** (*str, optional*) – Title for plot. Defaults to “Message flow”.

Returns *plotly.graph_objs.Figure* – Plotly Figure.

See also:

- *get_response_matrix*
- *fig_sankey*

Example

```
>>> from whatstk import WhatsAppChat
>>> from whatstk.graph import plot, FigureBuilder
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM)
>>> fig = FigureBuilder(chat=chat).user_message_responses_flow()
>>> plot(fig)
```

user_message_responses_heatmap (*norm='absolute', title='Response matrix'*)

Get the response matrix heatmap.

A response from user X to user Y happens if user X sends a message right after a message from user Y.

Parameters

- **norm** (*str, optional*) – Specifies the type of normalization used for reponse count. Can be:
 - 'absolute': Absolute count of messages.
 - 'joint': Normalized by total number of messages sent by all users.
 - 'sender': Normalized per sender by total number of messages sent by user.
 - 'receiver': Normalized per receiver by total number of messages sent by user.
- **title** (*str, optional*) – Title for plot. Defaults to “Response matrix”.

Returns *plotly.graph_objs.Figure* – Plotly Figure.

See also:

- *get_response_matrix*
- *fig_heatmap*

Example

```
>>> from whatstk import WhatsAppChat
>>> from whatstk.graph import plot, FigureBuilder
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM)
>>> fig = FigureBuilder(chat=chat).user_message_responses_heatmap()
>>> plot(fig)
```

user_msg_length_boxplot (*title='User message length', xlabel='User'*)

Generate figure with boxplots of each user’s message length.

Parameters

- **title** (*str, optional*) – Title for plot. Defaults to “User message length”.
- **xlabel** (*str, optional*) – x-axis label title. Defaults to “User”.

Returns *dict* – Dictionary with data and layout. Plotly compatible.

See also:

- *fig_boxplot_msglen*

Example

```
>>> from whatstk import WhatsAppChat
>>> from whatstk.graph import plot, FigureBuilder
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM)
>>> fig = FigureBuilder(chat=chat).user_msg_length_boxplot()
>>> plot(fig)
```

property usernames

Get list with users available in given chat.

Returns *list* – List with usernames available in chat DataFrame.

5.4.2 Core API

whatstk.whatsapp

WhatsApp parser.

whatstk.whatsapp.objects

Library WhatsApp objects.

Classes

<i>WhatsAppChat</i> (df)	Load and process a WhatsApp chat file.
--------------------------	--

class whatstk.whatsapp.objects.**WhatsAppChat** (df)

Bases: *whatstk._chat.BaseChat*

Load and process a WhatsApp chat file.

Parameters df (*pandas.DataFrame*) – Chat.

Attributes

<i>df</i>	Chat as DataFrame.
<i>end_date</i>	Chat end date.
<i>start_date</i>	Chat starting date.
<i>users</i>	List with users.

Methods

<i>from_source</i> (filepath, **kwargs)	Create an instance from a chat text file.
<i>from_sources</i> (filepaths[, auto_header, ...])	Load a WhatsAppChat instance from multiple sources.
<i>merge</i> (chat[, rename_users])	Merge current instance with chat.
<i>rename_users</i> (mapping)	Rename users.
<i>to_csv</i> (filepath)	Save chat as csv.

continues on next page

Table 8 – continued from previous page

<code>to_txt(filepath[, hformat])</code>	Export chat to a text file.
--	-----------------------------

Example

This simple example loads a chat using *WhatsAppChat*. Once loaded, we can access its attribute *df*, which contains the loaded chat as a DataFrame.

```
>>> from whatstk.whatsapp.objects import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.POKEMON)
>>> chat.df.head(5)
```

	username	
↪message		
date		
2016-08-06 13:23:00	Ash Ketchum	Hey
↪guys!		
2016-08-06 13:25:00	Brock	Hey Ash, good to have a common
↪group!		
2016-08-06 13:30:00	Misty	Hey guys! Long time haven't heard anything fro..
↪.		
2016-08-06 13:45:00	Ash Ketchum	Indeed. I think having a whatsapp group nowada..
↪.		
2016-08-06 14:30:00	Misty	
↪Definetly		

property *df*

Chat as DataFrame.

Returns pandas.DataFrame

property *end_date*

Chat end date.

Returns datetime

classmethod *from_source* (filepath, **kwargs)

Create an instance from a chat text file.

Parameters

- **filepath** (*str*) – Path to the file. It can be a local file (e.g. 'path/to/file.txt') or an URL to a hosted file (e.g. 'http://www.url.to/file.txt')
- ****kwargs** – Refer to the docs from *df_from_txt_whatsapp* for details on additional arguments.

Returns *WhatsAppChat* – Class instance with loaded and parsed chat.

See also:

- *df_from_txt_whatsapp*
- *WhatsAppChat.from_sources*

classmethod *from_sources* (filepaths, auto_header=None, hformat=None, encoding='utf-8')

Load a WhatsAppChat instance from multiple sources.

Parameters

- **filepaths** (*list*) – List with filepaths.

- **auto_header** (*bool, optional*) – Detect header automatically (applies to all files). If *None*, attempts to perform automatic header detection for all files. If *False*, *hformat* is required.
- **hformat** (*list, optional*) – List with the *header format* to be used for each file. The list must be of length equal to `len(filenamees)`. A valid header format might be `['%y-%m-%d %H:%M:%S] - %name:']`.
- **encoding** (*str*) – Encoding to use for UTF when reading/writing (ex. 'utf-8'). [List of Python standard encodings](#).

Returns *WhatsAppChat* – Class instance with loaded and parsed chat.

See also:

- *WhatsAppChat.from_source*
- *merge_chats*

Example

Load a chat using two text files. In this example, we use sample chats (available online, see urls in source code *whatstk.data*).

```
>>> from whatstk.whatsapp.objects import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> filepath_1 = whatsapp_urls.LOREM1
>>> filepath_2 = whatsapp_urls.LOREM2
>>> chat = WhatsAppChat.from_sources(filepaths=[filepath_1, filepath_2])
>>> chat.df.head(5)
```

	username	message
date		
2019-10-20 10:16:00	John	Laborum sed excepteur id eu cillum
↪sunt ut.		
2019-10-20 11:15:00	Mary	Ad aliquip reprehenderit proident est
↪irure mo...		
2019-10-20 12:16:00	+1 123 456 789	Nostrud adipiscing ex enim reprehenderit
↪minim...		
2019-10-20 12:57:00	+1 123 456 789	Deserunt proident laborum exercitation
↪ex temp...		
2019-10-20 17:28:00	John	Do ex dolor consequat
↪tempor et ex.		

merge (*chat, rename_users=None*)
Merge current instance with chat.

Parameters

- **chat** (*WhatsAppChat*) – Another chat.
- **rename_users** (*dict*) – Dictionary mapping old names to new names. Example: `{ 'John': ['Jon', 'J'], 'Ray': ['Raymond'] }` will map 'Jon' and 'J' to 'John', and 'Raymond' to 'Ray'. Note that old names must come as list (even if there is only one).

Returns *WhatsAppChat* – Merged chat.

See also:

- *rename_users*

- `merge_chats`

Example

Merging two chats can become handy when you have exported a chat in different times with your phone and hence each exported file might contain data that is unique to that file.

In this example however, we merge files from different chats.

```
>>> from whatstk.whatsapp.objects import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> filepath_1 = whatsapp_urls.LOREM1
>>> filepath_2 = whatsapp_urls.LOREM2
>>> chat_1 = WhatsAppChat.from_source(filepath=filepath_1)
>>> chat_2 = WhatsAppChat.from_source(filepath=filepath_2)
>>> chat = chat_1.merge(chat_2)
```

rename_users (*mapping*)

Rename users.

This might be needed in multiple occasions:

- Change typos in user names stored in phone.
- **If a user appears multiple times with different usernames, group these under the same name (this might happen when multiple chats are merged).**

Parameters **mapping** (*dict*) – Dictionary mapping old names to new names, example: {'John': ['Jon', 'J'], 'Ray': ['Raymond']} will map 'Jon' and 'J' to 'John', and 'Raymond' to 'Ray'. Note that old names must come as list (even if there is only one).

Returns *pandas.DataFrame* – DataFrame with users renamed according to *mapping*.

Raises **ValueError** – Raised if mapping is not correct.

Examples

Load LOREM2 chat and rename users *Maria* and *Maria2* to *Mary*.

```
>>> from whatstk.whatsapp.objects import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM2)
>>> chat.users
['+1 123 456 789', 'Giuseppe', 'John', 'Maria', 'Maria2']
>>> chat = chat.rename_users(mapping={'Mary': ['Maria', 'Maria2']})
>>> chat.users
['+1 123 456 789', 'Giuseppe', 'John', 'Mary']
```

property **start_date**

Chat starting date.

Returns *datetime*

to_csv (*filepath*)

Save chat as csv.

Parameters **filepath** (*str*) – Name of file.

to_txt (*filepath*, *hformat=None*)

Export chat to a text file.

Usefull to export the chat to different formats (i.e. using different hformats).

Parameters

- **filepath** (*str*) – Name of the file to export (must be a local path).
- **hformat** (*str*, *optional*) – Header format. Defaults to ‘%y-%m-%d, %H:%M - %name:’.

property users

List with users.

Returns list

whatstk.whatsapp.parser

Parser utils.

Functions

<code>df_from_txt_whatsapp(filepath[, ...])</code>	Load chat as a DataFrame.
<code>generate_regex(hformat)</code>	Generate regular expression from hformat.

`whatstk.whatsapp.parser.df_from_txt_whatsapp` (*filepath*, *auto_header=True*, *hformat=None*, *encoding='utf-8'*)

Load chat as a DataFrame.

Parameters

- **filepath** (*str*) – Path to the file. It can be a local file (e.g. ‘path/to/file.txt’) or an URL to a hosted file (e.g. ‘http://www.url.to/file.txt’)
- **auto_header** (*bool*, *optional*) – Detect header automatically. If False, *hformat* is required.
- **hformat** (*str*, *optional*) – *Format of the header*, e.g. ‘[%y-%m-%d %H:%M:%S] - %name:’. Use following keywords:
 - ‘%y’: for year (‘%Y’ is equivalent).
 - ‘%m’: for month.
 - ‘%d’: for day.
 - ‘%H’: for 24h-hour.
 - ‘%I’: for 12h-hour.
 - ‘%M’: for minutes.
 - ‘%S’: for seconds.
 - ‘%P’: for “PM”/”AM” or “p.m.”/”a.m.” characters.
 - ‘%name’: for the username.

Example 1: For the header ‘12/08/2016, 16:20 - username:’ we have the ‘hformat=’%d/%m/%y, %H:%M - %name:’.

Example 2: For the header '2016-08-12, 4:20 PM - username:' we have `hformat='%y-%m-%d, %I:%M %P - %name:'`.

- **encoding** (*str*, *optional*) – Encoding to use for UTF when reading/writing (ex. 'utf-8'). [List of Python standard encodings](#).

Returns *WhatsAppChat* – Class instance with loaded and parsed chat.

See also:

- `WhatsAppChat.from_source`
- `extract_header_from_text`

`whatstk.whatsapp.parser.generate_regex(hformat)`

Generate regular expression from hformat.

Parameters **hformat** (*str*) – Simplified syntax for the header, e.g. '%y-%m-%d, %H:%M:%S - %name:'.

Returns *str* – Regular expression corresponding to the specified syntax.

Example

Generate regular expression corresponding to 'hformat=%y-%m-%d, %H:%M:%S - %name:'.

```
>>> from whatstk.whatsapp.parser import generate_regex
>>> generate_regex('%y-%m-%d, %H:%M:%S - %name:')
('(?P<year>\\d{2,4})-(?P<month>\\d{1,2})-(?P<day>\\d{1,2}), (?P<hour>\\d{1,2}):(?P<minutes>\\d{2}):(?P<seconds>\\d{2}) - (?P<username>[^\:]*): ', '(?P<year>\\d{2,4})-(?P<month>\\d{1,2})-(?P<day>\\d{1,2}), (?P<hour>\\d{1,2}):(?P<minutes>\\d{2}):(?P<seconds>\\d{2}) - ')
```

whatstk.whatsapp.auto_header

Detect header from chat.

Functions

`extract_header_from_text(text[, encoding])` Extract header from text.

`whatstk.whatsapp.auto_header.extract_header_from_text(text, encoding='utf-8')`

Extract header from text.

Parameters

- **text** (*str*) – Loaded chat as string (whole text).
- **encoding** (*str*) – Encoding to use for UTF when reading/writing (ex. 'utf-8'). [List of Python standard encodings](#).

Returns *str* – Format extracted. None if no header was extracted.

Example

Load a chat using two text files. In this example, we use sample chats (available online, see urls in source code *whatstk.data*).

```
>>> from whatstk.whatsapp.parser import extract_header_from_text
>>> from urllib.request import urlopen
>>> from whatstk.data import whatsapp_urls
>>> filepath_1 = whatsapp_urls.POKEMON
>>> with urlopen(filepath_1) as f:
...     text = f.read().decode('utf-8')
>>> extract_header_from_text(text)
'%d.%m.%y, %H:%M - %name:'
```

whatstk.whatsapp.generation

Automatic generation of chat using Lorem Ipsum text and time series statistics.

Classes

<i>ChatGenerator</i> (size[, users, seed])	Generate a chat.
--	------------------

Functions

<i>generate_chats_hformats</i> (output_path[, size, ...])	Generate a chat and export using given header format.
---	---

class whatstk.whatsapp.generation.**ChatGenerator** (size, users=None, seed=100)

Bases: object

Generate a chat.

Parameters

- **size** (*int*) – Number of messages to generate.
- **users** (*list, optional*) – List with names of the users. Defaults to module variable `USERS`.
- **seed** (*int, optional*) – Seed for random processes. Defaults to 100.

Methods

<i>generate</i> ([filepath, hformat, last_timestamp])	Generate random chat as <i>WhatsAppChat</i> .
---	---

Examples

This simple example loads a chat using *WhatsAppChat*. Once loaded, we can access its attribute `df`, which contains the loaded chat as a DataFrame.

```
>>> from whatstk.whatsapp.generation import ChatGenerator
>>> from datetime import datetime
>>> from whatstk.data import whatsapp_urls
>>> chat = ChatGenerator(size=10).generate(last_timestamp=datetime(2020, 1, 1, 0,
↳0))
>>> chat.df.head(5)
```

message	username	
2019-12-31 09:43:04.000525	John	Quis labore laboris proident
et deserunt.		
2019-12-31 10:19:21.980039	+1 123 456 789	Non ullamco esse nulla
voluptate.		
2019-12-31 13:56:45.575426	John	Duis non ut officia, enim enim qui
cupidatat a...		
2019-12-31 15:47:29.995420	Giuseppe	Non ut nulla laboris
nostrud aute.		
2019-12-31 16:23:00.348542	John	Tempor irure in
velit tempor.		

generate (*filepath=None, hformat=None, last_timestamp=None*)

Generate random chat as *WhatsAppChat*.

Parameters

- **filepath** (*str*) – If given, generated chat is saved with name `filepath` (must be a local path).
- **hformat** (*str, optional*) – *Format of the header*, e.g. `'[%y-%m-%d %H:%M:%S] - %name:'`.
- **last_timestamp** (*datetime, optional*) – Datetime of last message. If *None*, defaults to current date.

Returns *WhatsAppChat* – Chat with random messages.

See also:

- *WhatsAppChat.to_txt*

```
whatstk.whatsapp.generation.generate_chats_hformats(output_path, size=2000, hfor-
mats=None, filepaths=None,
last_timestamp=None,
seed=100, verbose=False)
```

Generate a chat and export using given header format.

If no `hformat` specified, chat is generated & exported using all supported header formats.

Parameters

- **output_path** (*str*) – Path to directory to export all generated chats as txt.
- **size** (*int, optional*) – Number of messages of the chat. Defaults to 2000.
- **hformats** (*list, optional*) – List of header formats to use when exporting chat. If *None*, defaults to all supported header formats.

- **filepaths** (*list, optional*) – List with filepaths. If *None*, defaults to *hformat.replace(';', '_').replace('/', '')*.
- **last_timestamp** (*datetime, optional*) – Datetime of last message. If *None*, defaults to current date.
- **seed** (*int, optional*) – Seed for random processes. Defaults to 100.
- **verbose** (*bool*) – Set to *True* to print runtime messages.

See also:

- *ChatGenerator*
 - *ChatGenerator.generate*
-

whatstk.whatsapp.hformat

Header format utils.

Example: Check if header is available.

```
>>> from whatstk.utils.hformat import is_supported
>>> is_supported('%y-%m-%d, %H:%M:%S - %name:')
(True, True)
```

Functions

<code>get_supported_hformats_as_dict()</code>	Get dictionary with supported formats and relevant info.
<code>get_supported_hformats_as_list()</code>	Get list of supported formats.
<code>is_supported(hformat)</code>	Check if header <i>hformat</i> is currently supported.
<code>is_supported_verbose(hformat)</code>	Check if header <i>hformat</i> is currently supported (both manually and using <i>auto_header</i>).

`whatstk.whatsapp.hformat.get_supported_hformats_as_dict()`

Get dictionary with supported formats and relevant info.

Returns

dict –

Dict with two elements:

- **format**: Header format. All formats appearing are supported.
- **auto_header**: 1 if *auto_header* is supported, 0 otherwise.

`whatstk.whatsapp.hformat.get_supported_hformats_as_list()`

Get list of supported formats.

Returns *list* – List with supported formats (as str).

`whatstk.whatsapp.hformat.is_supported(hformat)`

Check if header *hformat* is currently supported.

Parameters **hformat** (*str*) – Header format.

Returns *tuple* – * bool: True if header is supported. * bool: True if header is supported with *auto_header* feature.

`whatstk.whatsapp.hformat.is_supported_verbose(hformat)`

Check if header *hformat* is currently supported (both manually and using *auto_header*).

Result is shown as a string.

Parameters *hformat* (*str*) – Information message.

Example

Check if format '%y-%m-%d, %H:%M - %name:' is supported.

```
>>> from whatstk.whatsapp.hformat import is_supported_verbose
>>> is_supported_verbose('%y-%m-%d, %H:%M - %name:')
"The header '%y-%m-%d, %H:%M - %name:' is supported. `auto_header` for this_
↪header is supported."
```

whatstk.analysis

Analysis tools.

Functions

<code>get_interventions_count(df, chat, ...)</code>	Get number of interventions per user per unit of time.
<code>get_response_matrix(df, chat, zero_own, norm)</code>	Get response matrix for given chat.

`whatstk.analysis.get_interventions_count(df=None, chat=None, date_mode='date', msg_length=False, cumulative=False, all_users=False, cummulative=None)`

Get number of interventions per user per unit of time.

The unit of time can be chosen by means of argument *date_mode*.

Note: Either *df* or *chat* must be provided.

Parameters

- **df** (*pandas.DataFrame*, *optional*) – Chat data. Attribute *df* of a chat loaded using Chat. If a value is given, *chat* is ignored.
- **chat** (*Chat*, *optional*) – Chat data. Object obtained when chat loaded using Chat. Required if *df* is None.
- **date_mode** (*str*, *optional*) – Choose mode to group interventions by. Defaults to *date_mode=date*. Available modes are:
 - 'date': Grouped by particular date (year, month and day).
 - 'hour': Grouped by day hours (24 hours).
 - 'month': Grouped by months (12 months).
 - 'weekday': Grouped by weekday (i.e. monday, tuesday, ..., sunday).
 - 'hourweekday': Grouped by weekday and hour.

- **msg_length** (*bool, optional*) – Set to True to count the number of characters instead of number of messages sent.
- **cumulative** (*bool, optional*) – Set to True to obtain commulative counts.
- **all_users** (*bool, optional*) – Obtain number of interventions of all users combined. Defaults to False.
- **cummulative** (*bool, optional*) – Deprecated, use cumulative.

Returns *pandas.DataFrame* – DataFrame with shape $N \times U$, where N : number of time-slots and U : number of users.

Raises **ValueError** – if `date_mode` value is not supported.

Example

Get number of interventions per user from **POKEMON** chat. The counts are represented as a $N \times U$ matrix, where N : number of time-slots and U : number of users.

```
>>> from whatstk import WhatsAppChat
>>> from whatstk.analysis import get_interventions_count
>>> from whatstk.data import whatsapp_urls
>>> filepath = whatsapp_urls.POKEMON
>>> chat = WhatsAppChat.from_source(filepath)
>>> counts = get_interventions_count(chat=chat, date_mode='date', msg_
↳ length=False)
>>> counts.head(5)
```

username	Ash Ketchum	Brock	Jessie & James	...	Prof. Oak	Raichu	Wobbuffet
date				...			
2016-08-06	2	2	0	...	0	0	0
2016-08-07	1	1	0	...	1	0	0
2016-08-10	1	0	1	...	0	2	0
2016-08-11	0	0	0	...	0	0	0
2016-09-11	0	0	0	...	0	0	0

[5 rows x 8 columns]

```
whatstk.analysis.get_response_matrix(df=None, chat=None, zero_own=True,
norm='absolute')
```

Get response matrix for given chat.

Obtains a DataFrame of shape $[n_users, n_users]$ counting the number of responses between members. Responses can be counted in different ways, e.g. using absolute values or normalised values. Responses are counted based solely on consecutive messages. That is, if $user_i$ sends a message right after $user_j$, it will be counted as a response from $user_i$ to $user_j$.

Axis 0 lists senders and axis 1 lists receivers. That is, the value in cell (i, j) denotes the number of times $user_i$ responded to a message from $user_j$.

Note: Either `df` or `chat` must be provided.

Parameters

- **df** (*pandas.DataFrame, optional*) – Chat data. Attribute `df` of a chat loaded using Chat. If a value is given, `chat` is ignored.
- **chat** (*Chat, optional*) – Chat data. Object obtained when chat loaded using Chat. Required if `df` is None.

- **zero_own** (*bool*, *optional*) – Set to True to avoid counting own responses. Defaults to True.
- **norm** (*str*, *optional*) – Specifies the type of normalization used for response count. Can be:
 - 'absolute': Absolute count of messages.
 - 'joint': Normalized by total number of messages sent by all users.
 - 'sender': Normalized per sender by total number of messages sent by user.
 - 'receiver': Normalized per receiver by total number of messages sent by user.

Returns *pandas.DataFrame* – Response matrix.

Example

Get absolute count on responses (consecutive messages) between users.

```
>>> from whatstk import WhatsAppChat
>>> from whatstk.analysis import get_response_matrix
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.POKEMON)
>>> responses = get_response_matrix(chat=chat)
>>> responses
```

	Ash Ketchum	Brock	...	Raichu	Wobbuffet
Ash Ketchum	0	0	...	1	0
Brock	1	0	...	0	0
Jessie & James	0	1	...	0	0
Meowth	0	0	...	0	0
Misty	2	1	...	1	0
Prof. Oak	0	1	...	0	0
Raichu	1	0	...	0	0
Wobbuffet	0	0	...	0	0

whatstk.graph

Plot tools using plotly.

Import `plot` (by plotly) to plot figures.

```
>>> from whatstk.graph import plot
```

whatstk.graph.base

Build plotly-compatible figures.

Classes

<code>FigureBuilder</code> (<i>df</i> , <i>chat</i>)	Generate a variety of figures from your loaded chat.
--	--

```
class whatstk.graph.base.FigureBuilder (df=None, chat=None)
```

Bases: object

Generate a variety of figures from your loaded chat.

Integrates feature extraction and visualization logic to automate data plots.

Note: Either `df` or `chat` must be provided.

Parameters

- **df** (*pandas.DataFrame, optional*) – Chat data. Attribute *df* of a chat loaded using Chat. If a value is given, `chat` is ignored.
- **chat** (*Chat, optional*) – Chat data. Object obtained when chat loaded using Chat. Required if `df` is None.

Attributes

<code>user_color_mapping</code>	Get mapping between user and color.
<code>usernames</code>	Get list with users available in given chat.

Methods

<code>user_interventions_count_linechart(...)</code>	Plot number of user interventions over time.
<code>user_message_responses_flow([title])</code>	Get the flow of message responses.
<code>user_message_responses_heatmap([norm, title])</code>	Get the response matrix heatmap.
<code>user_msg_length_boxplot([title, xlabel])</code>	Generate figure with boxplots of each user's message length.

property user_color_mapping

Get mapping between user and color.

Each user is assigned a color automatically, so that this color is preserved for that user in all to-be-generated plots.

Returns *dict* – Mapping from username to color (rgb).

user_interventions_count_linechart (*date_mode='date', msg_length=False, cumulative=False, all_users=False, title='User interventions count', xlabel='Date/Time', cumulative=None*)

Plot number of user interventions over time.

Parameters

- **date_mode** (*str, optional*) – Choose mode to group interventions by. Defaults to 'date'. Available modes are:
 - 'date': Grouped by particular date (year, month and day).
 - 'hour': Grouped by hours.
 - 'month': Grouped by months.
 - 'weekday': Grouped by weekday (i.e. monday, tuesday, ..., sunday).
 - 'hourweekday': Grouped by weekday and hour.
- **msg_length** (*bool, optional*) – Set to True to count the number of characters instead of number of messages sent.
- **cumulative** (*bool, optional*) – Set to True to obtain commulative counts.
- **all_users** (*bool, optional*) – Obtain number of interventions of all users combined. Defaults to False.

- **title** (*str*, *optional*) – Title for plot. Defaults to “User interventions count”.
- **xlabel** (*str*, *optional*) – x-axis label title. Defaults to “Date/Time”.
- **cummulative** (*bool*, *optional*) – Deprecated, use cumulative.

Returns *plotly.graph_objs.Figure* – Plotly Figure.

See also:

- *get_interventions_count*
- *fig_scatter_time*

Example

```
>>> from whatstk import WhatsAppChat
>>> from whatstk.graph import plot, FigureBuilder
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM)
>>> fig = FigureBuilder(chat=chat).user_interventions_count_
↳ linechart(cumulative=True)
>>> plot(fig)
```

user_message_responses_flow (*title='Message flow'*)

Get the flow of message responses.

A response from user X to user Y happens if user X sends a message right after a message from user Y.

Uses a Sankey diagram.

Parameters **title** (*str*, *optional*) – Title for plot. Defaults to “Message flow”.

Returns *plotly.graph_objs.Figure* – Plotly Figure.

See also:

- *get_response_matrix*
- *fig_sankey*

Example

```
>>> from whatstk import WhatsAppChat
>>> from whatstk.graph import plot, FigureBuilder
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM)
>>> fig = FigureBuilder(chat=chat).user_message_responses_flow()
>>> plot(fig)
```

user_message_responses_heatmap (*norm='absolute', title='Response matrix'*)

Get the response matrix heatmap.

A response from user X to user Y happens if user X sends a message right after a message from user Y.

Parameters

- **norm** (*str*, *optional*) – Specifies the type of normalization used for reponse count.
Can be:
 - 'absolute': Absolute count of messages.

- 'joint': Normalized by total number of messages sent by all users.
- 'sender': Normalized per sender by total number of messages sent by user.
- 'receiver': Normalized per receiver by total number of messages sent by user.

- **title**(*str*, *optional*) – Title for plot. Defaults to “Response matrix”.

Returns *plotly.graph_objs.Figure* – Plotly Figure.

See also:

- *get_response_matrix*
- *fig_heatmap*

Example

```
>>> from whatstk import WhatsAppChat
>>> from whatstk.graph import plot, FigureBuilder
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM)
>>> fig = FigureBuilder(chat=chat).user_message_responses_heatmap()
>>> plot(fig)
```

user_msg_length_boxplot (*title*='User message length', *xlabel*='User')

Generate figure with boxplots of each user’s message length.

Parameters

- **title**(*str*, *optional*) – Title for plot. Defaults to “User message length”.
- **xlabel**(*str*, *optional*) – x-axis label title. Defaults to “User”.

Returns *dict* – Dictionary with data and layout. Plotly compatible.

See also:

- *fig_boxplot_msglen*

Example

```
>>> from whatstk import WhatsAppChat
>>> from whatstk.graph import plot, FigureBuilder
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM)
>>> fig = FigureBuilder(chat=chat).user_msg_length_boxplot()
>>> plot(fig)
```

property usernames

Get list with users available in given chat.

Returns *list* – List with usernames available in chat DataFrame.

whatstk.graph.figures

Build Plotly compatible Figures.

whatstk.graph.figures.boxplot

Boxplot figures.

Functions

```
fig_boxplot_msglen(df[, username_to_color, Visualize boxplot.
...])
```

```
whatstk.graph.figures.boxplot.fig_boxplot_msglen(df, username_to_color=None, title="", xlabel=None)
```

Visualize boxplot.

Parameters

- **df** (*pandas.DataFrame*) – Chat data.
- **username_to_color** (*dict, optional*) –
- **title** (*str, optional*) – Title for plot. Defaults to “”.
- **xlabel** (*str, optional*) – x-axis label title. Defaults to None.

Returns plotly.graph_objs.Figure

whatstk.graph.figures.heatmap

Heatmap plot figures.

Functions

```
fig_heatmap(df_matrix[, title]) Generate heatmap figure from NxN matrix.
```

```
whatstk.graph.figures.heatmap.fig_heatmap(df_matrix, title="")
```

Generate heatmap figure from NxN matrix.

Parameters

- **df_matrix** (*pandas.DataFrame*) – Matrix as DataFrame. Index values and column values must be equal.
- **title** (*str*) – Title of plot. Defaults to “”.

Returns plotly.graph_objs.Figure

whatstk.graph.figures.sankey

Sankey plot figures.

Functions

<code>fig_sankey</code> (label, color, source, target, value)	Generate sankey image.
---	------------------------

`whatstk.graph.figures.sankey.fig_sankey` (label, color, source, target, value, title="")
Generate sankey image.

Parameters

- **label** (*list*) – List with node labels.
- **color** (*list*) – List with node colors.
- **source** (*list*) – List with link source id.
- **target** (*list*) – List with linke target id.
- **value** (*list*) – List with link value.
- **title** (*str, optional*) – Title. Defaults to “”.

Returns `plotly.graph_objs.Figure`

whatstk.graph.figures.scatter

Scatter plot figures.

Functions

<code>fig_scatter_time</code> (user_data[, ...])	Obtain Figure to plot using plotly.
--	-------------------------------------

`whatstk.graph.figures.scatter.fig_scatter_time` (user_data, username_to_color=None, title="", xlabel=None)

Obtain Figure to plot using plotly.

`user_data` must be a `pandas.DataFrame` with timestamps as index and a column for each user. You can easily generate suitable `user_data` using the function `get_interventions_count` (disclaimer: not compatible with `date_mode='hourweekday'`).

Parameters

- **user_data** (*pandas.DataFrame*) – Input data. Shape `nrows x ncols`, where `nrows` = number of timestamps and `ncols` = number of users.
- **username_to_color** (*dict, optional*) –
- **title** (*str, optional*) – Title of figure. Defaults to “”.
- **xlabel** (*str, optional*) – x-axis label title. Defaults to None.

Returns `plotly.graph_objs.Figure`

See also:

- `get_interventions_count`

whatstk.graph.figures.utils

Utils for library plots.

Functions

<code>hex_color_palette(n_colors)</code>	Get palette of <i>n_colors</i> color hexadecimal codes.
--	---

`whatstk.graph.figures.utils.hex_color_palette(n_colors)`

Get palette of *n_colors* color hexadecimal codes.

Parameters `n_colors` (*int*) – Size of the color palette.

whatstk.utils

Library generic utils.

whatstk.utils.chat_merge

Merging chats.

Functions

<code>merge_chats(dfs)</code>	Merge several chats into a single one.
-------------------------------	--

`whatstk.utils.chat_merge.merge_chats(dfs)`

Merge several chats into a single one.

Can come in handy when you have old exports and new ones, and both have relevant data.

Note: The dataframes must have an index with the timestamps of the messages, as this is required to correctly sort and merge the chats.

Parameters `dfs` (*List[pandas.DataFrame]*) – List with the chats as DataFrames.

Returns *pandas.DataFrame* – Merged chat.

whatstk.utils.exceptions

Library exceptions.

Exceptions

<code>HFormatError</code>	Raised when hformat could not be found.
<code>RegexError</code>	Raised when regex match is not possible.

exception `whatstk.utils.exceptions.HFormatError`

Bases: `Exception`

Raised when hformat could not be found.

exception whatstk.utils.exceptions.RegexError

Bases: Exception

Raised when regex match is not possible.

whatstk.utils.utils

Utils.

Classes

<i>ColnamesDf</i>	Access class constants using variable whatstk. utils.utils.COLNAMES_DF.
-------------------	--

class whatstk.utils.utils.ColnamesDf

Bases: object

Access class constants using variable whatstk.utils.utils.COLNAMES_DF.

Example

Attributes

<i>DATE</i>	Date column
<i>MESSAGE</i>	Message column
<i>MESSAGE_LENGTH</i>	Message length column
<i>USERNAME</i>	Username column

Access constant COLNAMES_DF.DATE:

```
>>> from whatstk.utils.utils import COLNAMES_DF
>>> COLNAMES_DF.DATE
'date'
```

DATE = 'date'

Date column

MESSAGE = 'message'

Message column

MESSAGE_LENGTH = 'message_length'

Message length column

USERNAME = 'username'

Username column

whatstk.data

Load sample chats.

This module contains the links to currently online-available chats. For more details, please refer to the source code.

Classes

```
Urls(POKEMON, LOREM, LOREM1, LOREM2, LOREM_2000)
```

```
class whatstk.data.Urls (POKEMON, LOREM, LOREM1, LOREM2, LOREM_2000)
```

Bases: tuple **Attributes**

<i>LOREM</i>	Alias for field number 1
<i>LOREM1</i>	Alias for field number 2
<i>LOREM2</i>	Alias for field number 3
<i>LOREM_2000</i>	Alias for field number 4
<i>POKEMON</i>	Alias for field number 0

property *LOREM*

Alias for field number 1

property *LOREM1*

Alias for field number 2

property *LOREM2*

Alias for field number 3

property *LOREM_2000*

Alias for field number 4

property *POKEMON*

Alias for field number 0

whatstk._chat

Library objects.

Classes

```
BaseChat(df[, platform])
```

Base chat object.

```
class whatstk._chat.BaseChat (df, platform=None)
```

Bases: object

Base chat object. **Attributes**

<i>df</i>	Chat as DataFrame.
<i>end_date</i>	Chat end date.
<i>start_date</i>	Chat starting date.
<i>users</i>	List with users.

Methods

<code>from_source(**kwargs)</code>	Load chat.
<code>merge(chat[, rename_users])</code>	Merge current instance with <code>chat</code> .
<code>rename_users(mapping)</code>	Rename users.
<code>to_csv(filepath)</code>	Save chat as csv.

df

Chat as `pandas.DataFrame`.

See also:

- `WhatsAppChat`

property df

Chat as `DataFrame`.

Returns `pandas.DataFrame`

property end_date

Chat end date.

Returns `datetime`

classmethod from_source (kwargs)**

Load chat.

Parameters `kwargs` – Specific to the child class.

Raises `NotImplementedError` – Must be implemented in children.

See also:

- `WhatsAppChat.from_source`

merge (chat, rename_users=None)

Merge current instance with `chat`.

Parameters

- **chat** (`WhatsAppChat`) – Another chat.
- **rename_users** (`dict`) – Dictionary mapping old names to new names. Example: { 'John': ['Jon', 'J'], 'Ray': ['Raymond'] } will map 'Jon' and 'J' to 'John', and 'Raymond' to 'Ray'. Note that old names must come as list (even if there is only one).

Returns `WhatsAppChat` – Merged chat.

See also:

- `rename_users`
- `merge_chats`

Example

Merging two chats can become handy when you have exported a chat in different times with your phone and hence each exported file might contain data that is unique to that file.

In this example however, we merge files from different chats.

```
>>> from whatstk.whatsapp.objects import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> filepath_1 = whatsapp_urls.LOREM1
>>> filepath_2 = whatsapp_urls.LOREM2
>>> chat_1 = WhatsAppChat.from_source(filepath=filepath_1)
>>> chat_2 = WhatsAppChat.from_source(filepath=filepath_2)
>>> chat = chat_1.merge(chat_2)
```

rename_users (*mapping*)

Rename users.

This might be needed in multiple occasions:

- Change typos in user names stored in phone.
- **If a user appears multiple times with different usernames, group these under the same name (this might happen when multiple chats are merged).**

Parameters *mapping* (*dict*) – Dictionary mapping old names to new names, example: {'Jon': ['Jon', 'J'], 'Ray': ['Raymond']} will map 'Jon' and 'J' to 'John', and 'Raymond' to 'Ray'. Note that old names must come as list (even if there is only one).

Returns *pandas.DataFrame* – DataFrame with users renamed according to *mapping*.

Raises **ValueError** – Raised if mapping is not correct.

Examples

Load LOREM2 chat and rename users *Maria* and *Maria2* to *Mary*.

```
>>> from whatstk.whatsapp.objects import WhatsAppChat
>>> from whatstk.data import whatsapp_urls
>>> chat = WhatsAppChat.from_source(filepath=whatsapp_urls.LOREM2)
>>> chat.users
['+1 123 456 789', 'Giuseppe', 'John', 'Maria', 'Maria2']
>>> chat = chat.rename_users(mapping={'Mary': ['Maria', 'Maria2']})
>>> chat.users
['+1 123 456 789', 'Giuseppe', 'John', 'Mary']
```

property *start_date*

Chat starting date.

Returns *datetime*

to_csv (*filepath*)

Save chat as csv.

Parameters *filepath* (*str*) – Name of file.

property *users*

List with users.

Returns *list*

5.4.3 Command line tools

whatstk-to-csv

Convert a WhatsApp txt file to csv.

```
usage: whatstk-to-csv [-h] [-f HFORMAT] input_filename output_filename

Convert a Whatsapp chat from csv to txt.

positional arguments:
input_filename          Input txt file.
output_filename         Name of output csv file.

optional arguments:
-h, --help              show this help message and exit
-f HFORMAT, --hformat HFORMAT
                        By default, auto-header detection is attempted. If does
                        not work, you can specify it manually using this
                        argument.
```

whatstk-graph

Get graph from your WhatsApp txt file.

```
usage: whatstk-graph [-h] [-o OUTPUT_FILENAME]
                    [-t {interventions_count,msg_length}]
                    [-id {date,hour,weekday,month}] [-ic] [-il] [-f HFORMAT]
                    input_filename

Visualise a WhatsApp chat. For advance settings, see package
librarydocumentation

positional arguments:
input_filename          Input txt file.

optional arguments:
-h, --help              show this help message and exit
-o OUTPUT_FILENAME, --output_filename OUTPUT_FILENAME
                        Graph generated can be stored as an HTMLfile.
-t {interventions_count,msg_length}, --type {interventions_count,msg_length}
                        Type of graph.
-id {date,hour,weekday,month}, --icount-date-mode {date,hour,weekday,month}
                        Select date mode. Only valid for
                        --type=interventions_count.
-ic, --icount-cumulative
                        Show values in a cumulative fashion. Only valid for
                        --type=interventions_count.
-il, --icount-msg-length
                        Count an intervention with its number of characters.
                        Otherwise an intervention is count as one. Only valid
                        for --type=interventions_count.
-f HFORMAT, --hformat HFORMAT
                        By default, auto-header detection is attempted. If does
                        not work, you can specify it manually using this
                        argument.
```

whatstk-generate-chat

Generate random WhatsApp chat.

```

whatstk-generate-chat --help
usage: Generate chat. [-h] -o OUTPUT_PATH
                        [--filenames FILENAMES [FILENAMES ...]] [-s SIZE]
                        [-f HFORMATS [HFORMATS ...]]
                        [--last-timestamp LAST_TIMESTAMP] [-v]

optional arguments:
-h, --help            show this help message and exit
-o OUTPUT_PATH, --output-path OUTPUT_PATH
                        Path where to store generated chats. Must exist.
--filenames FILENAMES [FILENAMES ...]
                        Filenames. Must be equal length of --hformats.
-s SIZE, --size SIZE  Number of messages to create per chat. Defaults to
                        500.
-f HFORMATS [HFORMATS ...], --hformats HFORMATS [HFORMATS ...]
                        Header format. If None, defaults to all supported
                        hformats. List formats as 'format 1' 'format 2' ...
--last-timestamp LAST_TIMESTAMP
                        Timestamp of last message. Format YYYY-mm-dd
-v, --verbose          Verbosity.

```

5.5 Why choose whatstk?

There are many python libraries to deal with WhatsApp and other platform chat files. Why should you choose **whatstk** over these?

5.5.1 Automatic parser

In WhatsApp, the chat might be exported in *different formats* depending on your phone configuration, which adds complexity when parsing the chat. **whatstk** incorporates a reliable and powerful *parser* to correctly infer the structure of most of the chats. In the rare and improbable case that the automatic parser does not work for a certain chat, you can still use *hformat*.

5.5.2 The power of pandas and plotly

whatstk uses well established and maintained python libraries *pandas* to process the data and *plotly* and exploits their potential to efficiently process and create figures.

5.5.3 Open source and Community oriented

The project is distributed under the [GPL-3.0 license](#), available on [GitHub](#) and open for user contributions.

The project is maintained since 2016 by [@lucasrodes](#).

5.6 Community & Governance

whatstk is a fully open-source project done for and by the community. It is primarily developed at society by the whatstk team, with the help of open-source developers.

For library discussions, consider joining [gitter group](#).

5.6.1 Leadership

BDFL

Role: final call in decisions related to the API.

- [Lucas Rodés-Guirao](#)

Community Contributors

- [Albert Aparicio Isarn](#)
- [Kolmar Kafran](#)
- [Clara Sáez Calabuig](#) (project logo)

5.7 Contribute

We are really open to your thoughts and feedback!

5.7.1 Bug reporting

Please report any bug that you may found to the [issues](#) section.

5.7.2 Requesting a Feature

If you find a new feature could be useful for the community, please try to add it in the [issues](#) section with a clear description.

5.7.3 Submitting a Pull Request

After forking and editing the code, make sure that your code passes all the tests:

```
sh ./run-tests.sh
```

This includes flake8 and pytest.

5.7.4 Adding new examples

To add new examples, consider editing yourself a `rst` file in `docs/source/` directory in the repository. For questions or doubts, join the [gitter](#) group.

5.7.5 API discussions

Consider joining the [gitter](#) group.

5.8 Changelog

5.8.1 Unreleased (2020-06-26)

Full set of changes: `0.2.0...HEAD`

5.8.2 0.2.0 (2020-04-30)

Full set of changes: `0.1.11...0.2.0`

5.8.3 0.1.11 (2019-06-19)

Full set of changes: `0.1.10...0.1.11`

5.8.4 0.1.10 (2019-06-18)

Full set of changes: 0.1.9...0.1.10

5.8.5 0.1.9 (2019-06-17)

Full set of changes: 0.1.8...0.1.9

5.8.6 0.1.8 (2019-06-17)

Full set of changes: 0.1.7...0.1.8

5.8.7 0.1.7 (2019-06-16)

Full set of changes: 0.1.6...0.1.7

5.8.8 0.1.6 (2019-06-16)

Full set of changes: 0.1.1...0.1.6

5.8.9 0.1.1 (2019-06-15)

Full set of changes: 0.1.0...0.1.1

5.8.10 0.1.0 (2019-06-15)

PYTHON MODULE INDEX

W

- `whatstk._chat`, 51
- `whatstk.analysis`, 41
- `whatstk.data`, 51
- `whatstk.graph.base`, 43
- `whatstk.graph.figures`, 47
 - `whatstk.graph.figures.boxplot`, 47
 - `whatstk.graph.figures.heatmap`, 47
 - `whatstk.graph.figures.sankey`, 48
 - `whatstk.graph.figures.scatter`, 48
 - `whatstk.graph.figures.utils`, 49
- `whatstk.utils`, 49
 - `whatstk.utils.chat_merge`, 49
 - `whatstk.utils.exceptions`, 49
 - `whatstk.utils.utils`, 50
- `whatstk.whatsapp`, 32
 - `whatstk.whatsapp.auto_header`, 37
 - `whatstk.whatsapp.generation`, 38
 - `whatstk.whatsapp.hformat`, 40
 - `whatstk.whatsapp.objects`, 32
 - `whatstk.whatsapp.parser`, 36

B

BaseChat (class in *whatstk._chat*), 51

C

ChatGenerator (class in *whatstk.whatsapp.generation*), 38

ColnamesDf (class in *whatstk.utils.utils*), 50

D

DATE (*whatstk.utils.utils.ColnamesDf* attribute), 50

df (*whatstk._chat.BaseChat* attribute), 51

df () (*whatstk._chat.BaseChat* property), 52

df () (*whatstk.whatsapp.objects.WhatsAppChat* property), 33

df () (*whatstk.WhatsAppChat* property), 26

df_from_txt_whatsapp () (in module *whatstk.whatsapp.parser*), 36

E

end_date () (*whatstk._chat.BaseChat* property), 52

end_date () (*whatstk.whatsapp.objects.WhatsAppChat* property), 33

end_date () (*whatstk.WhatsAppChat* property), 26

extract_header_from_text () (in module *whatstk.whatsapp.auto_header*), 37

F

fig_boxplot_msglen () (in module *whatstk.graph.figures.boxplot*), 47

fig_heatmap () (in module *whatstk.graph.figures.heatmap*), 47

fig_sankey () (in module *whatstk.graph.figures.sankey*), 48

fig_scatter_time () (in module *whatstk.graph.figures.scatter*), 48

FigureBuilder (class in *whatstk*), 29

FigureBuilder (class in *whatstk.graph.base*), 43

from_source () (*whatstk._chat.BaseChat* class method), 52

from_source () (*whatstk.whatsapp.objects.WhatsAppChat* class method), 33

from_source () (*whatstk.WhatsAppChat* class method), 26

from_sources () (*whatstk.whatsapp.objects.WhatsAppChat* class method), 33

from_sources () (*whatstk.WhatsAppChat* class method), 26

G

generate () (*whatstk.whatsapp.generation.ChatGenerator* method), 39

generate_chats_hformats () (in module *whatstk.whatsapp.generation*), 39

generate_regex () (in module *whatstk.whatsapp.parser*), 37

get_interventions_count () (in module *whatstk.analysis*), 41

get_response_matrix () (in module *whatstk.analysis*), 42

get_supported_hformats_as_dict () (in module *whatstk.whatsapp.hformat*), 40

get_supported_hformats_as_list () (in module *whatstk.whatsapp.hformat*), 40

H

hex_color_palette () (in module *whatstk.graph.figures.utils*), 49

HFormatError, 49

I

is_supported () (in module *whatstk.whatsapp.hformat*), 40

is_supported_verbose () (in module *whatstk.whatsapp.hformat*), 41

L

LOREM () (*whatstk.data.Url*s property), 51

LOREM1 () (*whatstk.data.Url*s property), 51

LOREM2 () (*whatstk.data.Url*s property), 51

LOREM_2000 () (*whatstk.data.Url*s property), 51

M

merge () (*whatstk._chat.BaseChat* method), 52

merge() (*whatstk.whatsapp.objects.WhatsAppChat method*), 34
merge() (*whatstk.WhatsAppChat method*), 27
merge_chats() (*in module whatstk.utils.chat_merge*), 49
MESSAGE (*whatstk.utils.utils.ColnamesDf attribute*), 50
MESSAGE_LENGTH (*whatstk.utils.utils.ColnamesDf attribute*), 50
module
 whatstk._chat, 51
 whatstk.analysis, 41
 whatstk.data, 51
 whatstk.graph.base, 43
 whatstk.graph.figures, 47
 whatstk.graph.figures.boxplot, 47
 whatstk.graph.figures.heatmap, 47
 whatstk.graph.figures.sankey, 48
 whatstk.graph.figures.scatter, 48
 whatstk.graph.figures.utils, 49
 whatstk.utils, 49
 whatstk.utils.chat_merge, 49
 whatstk.utils.exceptions, 49
 whatstk.utils.utils, 50
 whatstk.whatsapp, 32
 whatstk.whatsapp.auto_header, 37
 whatstk.whatsapp.generation, 38
 whatstk.whatsapp.hformat, 40
 whatstk.whatsapp.objects, 32
 whatstk.whatsapp.parser, 36

P

POKEMON() (*whatstk.data.Urls property*), 51

R

RegexError, 49
rename_users() (*whatstk._chat.BaseChat method*), 53
rename_users() (*whatstk.whatsapp.objects.WhatsAppChat method*), 35
rename_users() (*whatstk.WhatsAppChat method*), 28

S

start_date() (*whatstk._chat.BaseChat property*), 53
start_date() (*whatstk.whatsapp.objects.WhatsAppChat property*), 35
start_date() (*whatstk.WhatsAppChat property*), 28

T

to_csv() (*whatstk._chat.BaseChat method*), 53
to_csv() (*whatstk.whatsapp.objects.WhatsAppChat method*), 35

to_csv() (*whatstk.WhatsAppChat method*), 28
to_txt() (*whatstk.whatsapp.objects.WhatsAppChat method*), 35
to_txt() (*whatstk.WhatsAppChat method*), 28

U

Urls (*class in whatstk.data*), 51
user_color_mapping() (*whatstk.FigureBuilder property*), 29
user_color_mapping() (*whatstk.graph.base.FigureBuilder property*), 44
user_interventions_count_linechart() (*whatstk.FigureBuilder method*), 29
user_interventions_count_linechart() (*whatstk.graph.base.FigureBuilder method*), 44
user_message_responses_flow() (*whatstk.FigureBuilder method*), 30
user_message_responses_flow() (*whatstk.graph.base.FigureBuilder method*), 45
user_message_responses_heatmap() (*whatstk.FigureBuilder method*), 31
user_message_responses_heatmap() (*whatstk.graph.base.FigureBuilder method*), 45
user_msg_length_boxplot() (*whatstk.FigureBuilder method*), 31
user_msg_length_boxplot() (*whatstk.graph.base.FigureBuilder method*), 46
USERNAME (*whatstk.utils.utils.ColnamesDf attribute*), 50
usernames() (*whatstk.FigureBuilder property*), 32
usernames() (*whatstk.graph.base.FigureBuilder property*), 46
users() (*whatstk._chat.BaseChat property*), 53
users() (*whatstk.whatsapp.objects.WhatsAppChat property*), 36
users() (*whatstk.WhatsAppChat property*), 28

W

WhatsAppChat (*class in whatstk*), 25
WhatsAppChat (*class in whatstk.whatsapp.objects*), 32
whatstk._chat
 module, 51
whatstk.analysis
 module, 41
whatstk.data
 module, 51
whatstk.graph.base
 module, 43
whatstk.graph.figures
 module, 47
whatstk.graph.figures.boxplot
 module, 47
whatstk.graph.figures.heatmap
 module, 47

- whatstk.graph.figures.sankey
 - module, [48](#)
- whatstk.graph.figures.scatter
 - module, [48](#)
- whatstk.graph.figures.utils
 - module, [49](#)
- whatstk.utils
 - module, [49](#)
- whatstk.utils.chat_merge
 - module, [49](#)
- whatstk.utils.exceptions
 - module, [49](#)
- whatstk.utils.utils
 - module, [50](#)
- whatstk.whatsapp
 - module, [32](#)
- whatstk.whatsapp.auto_header
 - module, [37](#)
- whatstk.whatsapp.generation
 - module, [38](#)
- whatstk.whatsapp.hformat
 - module, [40](#)
- whatstk.whatsapp.objects
 - module, [32](#)
- whatstk.whatsapp.parser
 - module, [36](#)