Chat Analyzer

Release 1.0.1b1

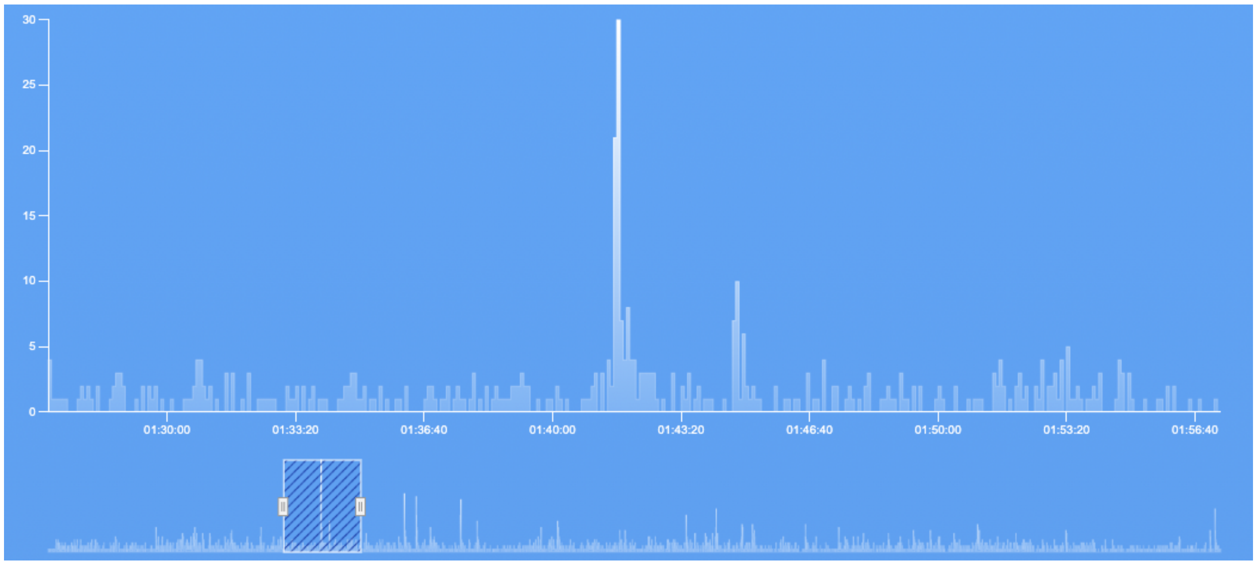
David Fryd

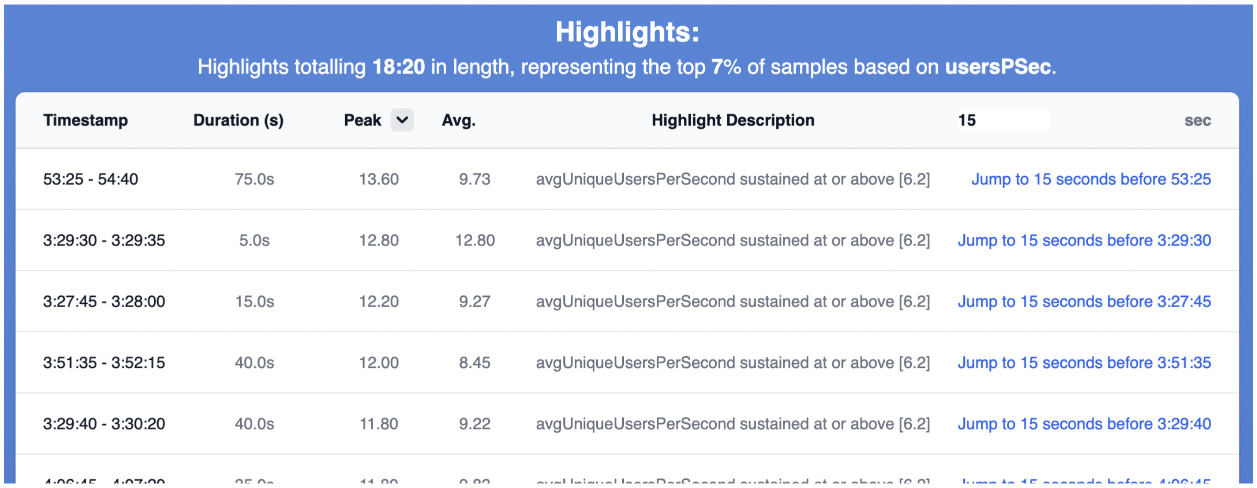
Aug 16, 2022

CONTENTS

Chat Analyzer is a tool used to process and analyze chat data from past live streams, providing summarized information about chat activity over the stream's lifetime. The information currently reported primarily revolves around the activity per second for various metrics, with future plans to incorporate semantic analysis into the program (happy chats? sad? excited, wholesome, etc…).

Here's a video covering the start-to-end usage of the software





Highlights:

Highlights totalling **18:20** in length, representing the top **7**% of samples based on **usersPSec**.

Timestamp	Duration (s)	Peak	Avg.	Highlight Description	15 sec
53:25 - 54:40	75.0s	13.60	9.73	avgUniqueUsersPerSecond sustained at or above [6.2]	Jump to 15 seconds before 53:25
3:29:30 - 3:29:35	5.0s	12.80	12.80	avgUniqueUsersPerSecond sustained at or above [6.2]	Jump to 15 seconds before 3:29:30
3:27:45 - 3:28:00	15.0s	12.20	9.27	avgUniqueUsersPerSecond sustained at or above [6.2]	Jump to 15 seconds before 3:27:45
3:51:35 - 3:52:15	40.0s	12.00	8.45	avgUniqueUsersPerSecond sustained at or above [6.2]	Jump to 15 seconds before 3:51:35
3:29:40 - 3:30:20	40.0s	11.80	9.22	avgUniqueUsersPerSecond sustained at or above [6.2]	Jump to 15 seconds before 3:29:40
4:06:45 - 4:07:20	35.0s	11.80	9.83	avgUniqueUsersPerSecond sustained at or above [6.2]	Jump to 15 seconds before 4:06:45

Paired with the visualizer hosted on chatanalyze.com, easily understand and interpret how your chat responds to your performance!

Currently both YouTube and Twitch VODs are supported.

CHAPTER

ONE

TARGET AUDIENCE:

Anyone can use this tool on anyone's streams, enabling people beyond the creators themselves to view and use chat analytics data.

- Editors
  - Making a clips channel? Need to edit down an 18 hour stream into a 20 minute youtube video? Quickly find parts of the stream where chat went crazy and look there first! Chat activity is generally a great proxy for how exciting/engaging the stream is at any moment in time. Not only will you more quickly find the most interesting sections of the stream, but you'll ensure that you don't miss any moments that you might have might have slept through during your 18 hour-long editing session.
  - Are you a creator that edits your own videos? Focus more on creating the content you love, and less on the tedious work of editing.
- Streamers / Creators
  - Draw specific connections between the content you make and how it engages your community. What type of content makes your chat go wild? What strategies/types of content are more effective than others?
  - See the exact moments people decide to subscribe/become members- what type of content moves people so much that they decide to support you and your stream?
- Small/Upcoming streamers
  - Immitation is the best form of flattery. Pick a popular streamer and see what parts of their stream generate the most engagement! What type of content/strategies can you use to make your stream more engaging?

CHAPTER TWO

INSTALLATION

This tool is distributed on PyPI and can be installed with pip:

```
pip install chat-analyzer
```

To update to the latest version, add the phrase `--upgrade` to the end of the above command.

Alternatively, the tool can be installed with `git`:

```
git clone https://github.com/David-Fryd/chat-analyzer.git
cd chat-analyzer
python setup.py install
```

CHAPTER THREE

USAGE

Basic to intermediate/advanced usage can be found on the Getting Started page of the documentation.

A simple command might look like:

```
chat_analyzer 'https://www.twitch.tv/videos/{VIDEO_ID}' -o
```

3.1 Command line

```
usage: chat_analyzer [-h] [--version] [--platform {youtube,twitch}]
                     [--mode {url,chatfile,reanalyze}]
                     [--save-chatfile-output SAVE_CHATFILE_OUTPUT]
                     [--interval INTERVAL] [--print-interval PRINT_INTERVAL]
                     [--highlight-percentile HIGHLIGHT_PERCENTILE]
                     [--highlight-metric {usersPSec,chatsPSec,activityPSec}]
                     [--description DESCRIPTION] [--output OUTPUT] [--nojson]
                     [--debug] [--break BREAK]
                     source
```

More complete documentation can be found on the Command Line Usage page.

CHAPTER FOUR

OUTPUT

For non-developers, I highly recommend you use the visualizer on chatanalyze.com to view the output of the program. Simply follow the instructions and upload the output json file to the visualizer.

All of the analytical data is output in a single `.json` file. Certain datapoints exist regardless of the platform the VOD is from, some datapoints are specific to the platform.

More complete documentation can be found on the Output Specifications page.

CHAPTER FIVE

KNOWN ISSUES

- Reported Users Per Second (UPSec/avgUniqueUsersPerSecond) is actually Users Per Sample (UPSample).
  - UPSec is not as simple as dividing unique users per sample by sample length
  - As sample size -> video duration, UPSample -> Total unique users.
  - For now it is a fine approximation for small sample durations, but in the future should be remedied either by improving UPSec calc algorithm or simply by reporting UPSample instead.

CHAPTER

SIX

CONTRIBUTING

If you would like to help improve the tool, you’ll find more information on contributing in the Contributing Guide.

CHAPTER

SEVEN

SPECIAL THANKS

This project wouldn't exist without Xenova and their chat-downloader! All of the platform-standardization and downloading logic that they worked on for their downloader made the analyzer infinitely easier to write. In order to avoid compatability issues, this software comes packaged with a frozen version of the downloader src, and all credit goes to Xenova for the contents in the `chat_downloader` directory. Since this was also my first ever python project made for distribution, their organizational structure and style was invaluable reference for the packaging/distribution process. If you are willing, go on over to their repo and show them some support as well :)

7.1 Getting Started

Welcome to the getting started guide for Chat Analyzer! In this guide, we'll cover the basics of how to use the software, some intermediate and advanced uses, and overall best practices when using the software.

Warning: This guide, the software itself, and the website visualization are still in the beta phase of development and not yet complete. Because Twitch Rivals started recently, I decided to make the software available because I believe it will be useful to some people even in its current state. If there are any questions/concerns, feel free to contact me at info@chatanalyze.com, or shoot me a DM on discord `NaCl-y#1117`.

Note: Even though this guide is not yet complete, the available arguments/flags are fully documented by running the `--help` flag, and can also be found on the *Command Line Interface Specification page*.

7.1.1 Basic Usage

In this section, we'll cover the basic steps necessary to use the Chat Analyzer software.

Step 1 - Installation

If you haven't already installed the software, you can easily install the software one of two ways:

This tool is distributed on PyPI and can be installed with pip:

```
pip install chat-analyzer
```

To update to the latest version, run:

```
pip install chat-analyzer --upgrade
```

Alternatively, the tool can be installed with `git`:

```
git clone https://github.com/David-Fryd/chat-analyzer.git
cd chat-analyzer
python setup.py install
```

Step 2 - Pick a Past Stream

By default, we will be downloading the chatlog data to analyze from a past stream's url. Currently we support Twitch and YouTube streams/VODs. For a Twitch stream, simply copy the `twitch.tv...` link to use with the analyzer. For a YouTube stream, it is best to use the "share" button underneath the video player to get the `youtu.be` link, but the analyzer itself will still work with a standard YouTube link.

For example, a youtube/twitch link might look like:

```
https://youtu.be/d6JXhg1GBKs
https://www.twitch.tv/videos/1552248469
```

For help picking the right type of link, you can reference the rules & criteria under the *url* section.

Step 3 - Run the Chat Analyzer

Now that we have the link, all we have to do is run the analyzer and give it the link as an input. Open up a terminal/command prompt and run the following command:

```
chat_analyzer '<link>'
```

The program will produce an output file in the directory that the program was run in (the current directory). The output file/filepath can also be assigned using the `-o` flag:

```
chat_analyzer '<link>' -o '<output_filepath>'
```

After starting the program, the chatlog download will initiate and you should see output that looks something like this:

```
Getting chatlog using Xenonvas chat-downloader (https://github.com/xenova/chat-
↪downloader)...
Successfully retrieved chat generator:
    Title: <video_title>
    Duration: ... (... seconds)
NOTICE: Downloading chats from a url is the largest rate-limiting factor.
            If you intend to sample the data differently multiple times, consider using␣
↪chatfile mode, or saving the chat data with --save-chatfile.

Processing (Sampling) chat data...
  Completion    |   Processed Media Time    |   Messages Processed
    (...%)      |         ... / ...         |         ...
```

As messages are downloaded, you will see constant updates indicating the progress of the download.

Note: The downloading of chat data is the slowest part of the entire process. Twitch/YouTube limits the rate at which chat data can be downloaded. If you want to resample the chat multiple times, look into using *chatfile* mode

After the download has finished, you should see the following report:

```
Downloaded & Processed ... messages.
Post-processing (Analyzing)...
Post-processing (Analyzing) complete!
Successfully wrote chat analytics to `<output_filepath>`
```

The analyzed output file can now be found at `<output_filepath>`!

Final Step - Visualize the Chat Data

Now that we have generated the output file, we will use the visualizer found at chatanalyze.com/visualize to nicely visualize the analytical data. Once on the page, all you have to do is select the output file on the page and the visualization data will automatically appear on screen.

The two core features currently available are the **graph** representation of chat activity, and the **highlights** table. The graph provides a quick visual reference to the chat activity at any given point throughout the whole stream. The highlights table provides a useful way to examine the highest-activity portions of the video, and to quickly jump to those sections of the stream using the "Jump To" functionality.

7.1.2 Usage Modes & Source

In its simplest form, this software outputs information about a chatlog associated with a livestream. Regardless of the mode that is used, the output file format is the same (More details under *Output Specification*).

The three modes that can be used all refer to the type of input the program receieves.

url

The default mode, `url` accepts a link from a supported streaming site, downloads the raw chat data, processes the raw chat data into samples, and analyzes the samples.

Streaming services like Twitch & YouTube limit the rate at which we can download chat messages, The slowest part of the analytics process is downloading the chats themselves.

The link provided **must**:

- Be a link to a past (finished) livestream.
- Come from a supported streaming site.
- Be the original video with the chatlog/replay. (It can't be a reposted video.)

Note: If you want to analyze a YouTube stream, it is recommended you provide the `youtu.be` link generated through the "share" feature of the video.

While the standard YouTube video link will work to download the chat and produce the data, the `youtu.be` short link works better with the visualizer at chatanalyze.com, enabling "Jump to" functionality (quickly jumping to highlighted points in the video).

chatfile

`chatfile` mode . . .

Warning: This section has not yet been written. Even though this guide is not yet complete, the available arguments/flags are fully documented by running the `--help` flag, and can also be found on the *Command Line Interface Specification page*.

reanalyze

`reanalyze` mode . . .

Warning: This section has not yet been written. Even though this guide is not yet complete, the available arguments/flags are fully documented by running the `--help` flag, and can also be found on the *Command Line Interface Specification page*.

7.2 Command Line Usage

7.2.1 Overview

A full list of command line arguments can be obtained by running the help command:

```
$ chat_analyzer -h
```

The output of which is as follows:

```
usage: chat_analyzer [-h] [--version] [--platform {youtube,twitch}]
                     [--mode {url,chatfile,reanalyze}]
                     [--save-chatfile-output SAVE_CHATFILE_OUTPUT]
                     [--interval INTERVAL] [--print-interval PRINT_INTERVAL]
                     [--highlight-percentile HIGHLIGHT_PERCENTILE]
                     [--highlight-metric {usersPSec,chatsPSec,activityPSec}]
                     [--description DESCRIPTION] [--output OUTPUT] [--nojson]
                     [--debug] [--break BREAK]
                     source

A tool used to process and analyze chat data from past live streams, providing
summarized information about chat activity over the stream's lifetime.

Required Arguments:
  source
                        Raw chat data to process and analyze, or processed
```

(continues on next page)

(continued from previous page)

```
                        sample data to re-analyze.

                        In mode=[1m'url'[0m, (default) source is a url to a
                        past stream/VOD.
                        We currently only support links from: www.youtube.com,
                        www.twitch.tv, youtu.be. The link must
                        be to the original stream/VOD with that attached chat
                        replay.

                        In mode=[1m'chatfile'[0m, source is a filepath to a
                        .json containing [3mraw chat data[0m,
                        produced by Xenonva's chat-downloader, or by this
                        program's `--save-chatfile-output` flag. NOTE: the
                        --platform argument is required
                        when using this mode.

                        In mode=[1m'reanalyze'[0m, source is a filepath to a
                        .json file previously produced by this program
                        which contains [3mexisting sample data to
                        reanalyze[0m.
                        (Highlights and spikes are regenerated, the existing
                        samples are not affected).

Optional Arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  --platform {youtube,twitch}
                        When reading from a chatfile, specify the platform the
                        chat was downloaded from. By default, Xenova's chat
                        downloader does not store this information with the
                        chat data so it must be manually specified. If the
                        incorrect platform is entered, site-specific data will
                        be innacurate but the program will still run and
                        analyze common attributes. (default: None)

Program Behavior (Mode):
  --mode {url,chatfile,reanalyze}, -m {url,chatfile,reanalyze}
                        The program can be run in three modes:

                        [3mNOTE: All modes result in chat analytics output as
                        a .json file.[0m

                        [1m'url'[0m mode (default) downloads raw chat data
                        from an appropriate source url, processes the raw
                        chat data into samples, and then analyzes the
                        samples.

                        [1m'chatfile'[0m mode reads raw chat data from a
                        .json file, processes the raw chat data into
                        samples, and then analyzes the samples.
                        (We accept raw chat files produced by Xenonva's chat-
                        downloader, or by this program through '--save-
```

(continues on next page)

(continued from previous page)

```
                        chatfile-output').

                        [1m'reanalyze'[0m mode reads existing sample data
                        from a .json file produced by this program in a
                        previous run, and recalculates ONLY the post-
                        processed data based on the existing samples.
                        (Highlights and spikes are regenerated, the existing
                        samples are not affected).

                        (default: url)
  --save-chatfile-output SAVE_CHATFILE_OUTPUT, -sc SAVE_CHATFILE_OUTPUT
                        Filepath of the raw chat data to save. If downloading
                        chat data from a URL, save the raw chat data to the
                        provided filepath in addition to processing it, so
                        that the raw data can be [3mfully[0m reprocessed and
                        analyzed again quickly (using mode='chatfile'). NOTE:
                        Chatfiles are *much* larger in comparison to the
                        analytics file. NOTE: json file extension is enforced
                        because it affects the content that the chat
                        downloader writes to the file. (default: None)

Processing (Sampling):
  --interval INTERVAL, -i INTERVAL
                        The time interval (in seconds) at which to compress
                        datapoints into samples. i.e. Duration of the samples.
                        The smaller the interval, the more granular the
                        analytics are. At interval=5, each sample contains 5
                        seconds of cumulative data. *(With the exception of
                        the last sample, which may be shorter than the
                        interval).* (default: 5)
  --print-interval PRINT_INTERVAL
                        Number of messages between progress updates to the
                        console. If <= 0, progress is not printed. (default:
                        100)

Post Processing (Analyzing):
  --highlight-percentile HIGHLIGHT_PERCENTILE, -ep HIGHLIGHT_PERCENTILE
                        A number between 0 and 100, representing the cutoff
                        percentile that a sample's attribute must meet to be
                        considered a 'highlight' of the chatlog. Samples in
                        the top HIGHLIGHT_PERCENTILE% of the selected
                        highlight metric will be considered high-engagement
                        samples and included within the constructed
                        highlights. The larger the percentile, the greater the
                        metric requirement before being reported. If
                        'highlight-percentile'=93.0, only samples in the 93rd
                        percentile (top 7.0%) of the selected metric will be
                        included in the highlights. (default: 93.0)
  --highlight-metric {usersPSec,chatsPSec,activityPSec}, -em {usersPSec,chatsPSec,
↪activityPSec}
                        The metric to use for engagement analysis when
                        constructing highlights. Samples in the top
```

(continues on next page)

(continued from previous page)

```
                        HIGHLIGHT_PERCENTILE% of the selected metric will
                        be considered high-engagement samples and included
                        within the constructed highlights.
                        Each highlight metric choice corresponds to a
                        datapoint for each sample.

                        [1m'usersPSec'[0m compares samples based off of the
                        average number unique users that send a chat per
                        second of the sample.

                        [1m'chatsPSec'[0m compares samples based off of the
                        average number of chats per second of the sample
                        (not necessarily sent by unique users).

                        [1m'activityPSec'[0m compares samples based off of
                        the average number of any type of message that
                        appears in the chat per second of the sample.

                        (default: usersPSec)

Output:
  --description DESCRIPTION, -d DESCRIPTION
                        A description included in the output file to help
                        distinguish it from other output files.
                        ex: -d "Ludwig product announcement, small intervals"
                        (default: None)
  --output OUTPUT, -o OUTPUT
                        The filepath to write the output to. If not specified,
                        the output is written to '[MEDIA TITLE].json.' If the
                        provided file path does not end in '.json', the
                        '.json' file extension is appended automaticaly to the
                        filepath (disable with --nojson). (default: None)
  --nojson              Disable the automatic appending of the '.json' file
                        extension to the provided output filepath. (default:
                        False)

Debugging:
  --debug, -db          Enable debug mode (debug info is printed) (default:
                        False)
  --break BREAK, -b BREAK
                        Stop processing messages after BREAK number of
                        messages have been processed. No effect if val < 0
                        (default: -1)
```

See *Getting Started* for examples and detailed use-cases.

7.3 Output Specifications

All of the analytical data is output in a single .json file. Certain datapoints exist regardless of the platform the VOD is from, some datapoints are specific to the platform.

7.3.1 Common fields:

Chat Analytics Data

The Chat Analytics object is directly transformed into JSON data.

`class chat_analyzer.dataformat.`**`ChatAnalytics`**(*duration: float*, *interval: int*, *description: str*, *program_version: str*, *platform: str*, *duration_text: str = ''*, *interval_text: str = ''*, *mediaTitle: str = 'No Media Title'*, *mediaSource: str = 'No Media Source'*, *samples: ~typing.List[~chat_analyzer.dataformat.Sample] = <factory>*, *totalActivity: int = 0*, *totalChatMessages: int = 0*, *totalUniqueUsers: int = 0*, *overallAvgActivityPerSecond: float = 0*, *overallAvgChatMessagesPerSecond: float = 0*, *overallAvgUniqueUsersPerSecond: float = 0*, *highlights: ~typing.List[~chat_analyzer.dataformat.Highlight] = <factory>*, *highlights_duration: float = 0*, *highlights_duration_text: str = ''*, *highlight_percentile: float = 0*, *highlight_metric: str = ''*, *spikes: ~typing.List[~chat_analyzer.dataformat.Spike] = <factory>*, *_overallUserChats: dict = <factory>*, *_currentSample: ~typing.Optional[~chat_analyzer.dataformat.Sample] = None*)

Bases: `ABC`

Class that contains the results of the chat data analysis/processing.

An instance of a subclass is created and then modified throughout the analysis process. After the processing of the data is complete, the object will contain all relevant results we are looking for.

This class cannot be directly instantiated, see the subclasses YoutubeChatAnalytics & TwitchChatAnalytics. YT and Twitch chats report/record data differently and contain site-specific events, so we centralize common data/fxnality and separate specifics into subclasses.

The object can then be converted to JSON/printed/manipulated as desired to format/output the results as necessary.

—

[Defined when class Initialized]:

duration: float
The total duration (in seconds) of the associated video/media. Message times correspond to the video times.

interval: int
The time interval (in seconds) at which to compress datapoints into samples. i.e. Duration of the samples. The smaller the interval, the more granular the analytics are. At interval=5, each sample contains 5 seconds of cumulative data. *(With the exception of the last sample, which may be shorter than the interval.)* This

is b/c media duration is not necessarily divisible by the interval. #(samples in raw_data) is about (video duration/interval) (+1 if necessary to encompass remaining non-divisible data at end of data).

description: str
A description included to help distinguish it from other analytical data.

program_version: str
The version of the chat analytics program that was used to generate the data. Helps identify outdated/version-specific data formats.

platform: str
Used to store the platform the data came from: 'www.youtube.com', 'www.twitch.tv', 'youtu.be'… While it technically can be determined by the type of subclass, this makes for easier conversion to JSON/output

[Automatically re-defined on post-init]:

duration_text: str
String representation of the media duration time.

interval_text: str
String representation of the interval time.

[Defined w/ default and modified DURING analysis]:

mediaTitle: str
The title of the media associated with the chatlog.

mediaSource: str
The link to the media associated with the chatlog (url that it was origianlly downloaded from or filepath of a chatfile).

samples: List[Sample]
An array of sequential samples, each corresponding to data about a section of chat of 'interval' seconds long. Each sample has specific data corresponding to a time interval of the vid. See the 'Sample' class

totalActivity: int
The total number of messages/things (of any type!) that appeared in chat. (Sum of intervalActivity from all samples) Includes messages,notifications,subscriptions, superchats, … *anything* that appeared in chat

totalChatMessages: int
The total number of chats sent by human (non-system) users (what is traditionally thought of as a chat) NOTE: Difficult to discern bots from humans other than just creating a known list of popular bots and blacklisting, because not all sites (YT/Twitch) provide information on whether chat was sent by a registered bot or not.

highlight_percentile: float
The cutoff percentile that samples must meet to be considered a highlight

highlight_metric: str
The metric to use for engagement analysis to build highlights. NOTE: must be converted into actual Sample field name before use.

[Defined w/ default and modified AFTER analysis]:

totalUniqueUsers: int
The total number of unique users that sent a chat message (human users that sent at least one traditional chat)

overallAvgActivityPerSecond: float
The average activity per second across the whole chatlog. (totalActivity/totalDuration)

overallAvgChatMessagesPerSecond: float
The average number of chat messages per second across the whole chatlog. (totalChatMessages/totalDuration)

overallAvgUniqueUsersPerSecond: float
The average number of unique users chatting per second.

highlights: List[Highlight]
A list of the high engagement sections of the chatlog.

highlights_duration: float
The cumulative duration of the highlights (in seconds)

highlights_duration_text: str
The cumulative duration of the highlights represented in text format (i.e. hh:mm:ss)

spikes: List[Spike]
Not yet implemented TODO A list of the calculated spikes in the chatlog. May contain spikes of different types, identifiable by the spike's type field.

`chatlog_post_process`(*settings: ProcessSettings*)

After we have finished iterating through the chatlog and constructing all of the samples, we call chatlog_post_process() to process the cumulative data points (so we don't have to do this every time we add a sample).

This step is sometimes referred to as "analysis".

Also removes the internal fields that don't need to be output in the JSON object.

Parameters
`settings` (*ProcessSettings*) – Utility class for passing information from the analyzer to the chatlog processor and post-processor

`create_new_sample`()

Post-processes the previous sample, then appends & creates a new sample following the previous sample sequentially. If a previous sample doesn't exist, creates the first sample.

NOTE: If there there are only 2 chats, one at time 0:03, and the other at 5:09:12, there are still a lot of empty samples in between (because we still want to graph/track the silence times with temporal stability)

`get_highlights`(*highlight_metric: str*, *highlight_percentile: float*)

Highlights reference a contiguous period of time where the provided metric remains above the percentile threshold. Find and return a list of highlights referencing the start and end times of samples whose highlight_metric is in the highlight_percentile for contiguous period of time of the referenced samples.

A highlight may reference more than one sample if contiguous samples meet the percentile cutoff.

Samples in the top 'percentile'% of the selected engagement metric will be considered high-engagement samples and included in the highlights output list. The larger the percentile, the greater the metric requirement before being reported. If 'engagement-percentile'=93.0, any sample in the 93rd percentile (top 7.0%%) of the selected metric will be considered an engagement highlight.

These high-engagement portions of the chatlog are stored as highlights, and may last for multiple samples.

This method should only be called after the averages have been calculated, ensuring accurate results when determining periods of high engagement.

Parameters

- `highlight_metric` – The metric samples are compared to determine if they are high-engagement samples. NOTE: Internally converted to the actual field name of a sample field.

- **`highlight_percentile`** – The cutoff percentile that the samples must meet to be included in a highlight

Returns
a list of highlights referencing samples that met the percentile cutoff requirements for the provided metric

Return type
List[*Highlight*]

`get_spikes`(*spike_sensitivity*, *spike_metric*)

A spike is a point in the chatlog where from one sample to the next, there is a sharp increase in the provided metric.

. . . ? Are spikes sustained or..? ?: A spike is a point in the chatlog where the activity is significantly different from the average activity. Activity is significantly different if it is > avg*SPIKE_MULT_THRESHOLD. We detect a spike if the high activity level is maintained for at least SPIKE_SUSTAIN_REQUIREMENT # of samples.

`print_process_progress`(*msg*, *idx*, *finished=False*)

Prints progress of the chat download/process to the console.

If finished is true, normal printing is skipped and the last bar of progress is printed. This is important because we print progress every UPDATE_PROGRESS_INTERVAL messages, and the total number of messages is not usually divisible by this. We therefore have to slightly change the approach to printing progress for this special case.

`process_chatlog`(*chatlog: Chat*, *source: str*, *settings: ProcessSettings*)

Iterates through the whole chatlog and calculates the analytical data (Modifies and stores in a ChatAnalytics object).

Parameters

- **`chatlog`** (*`chat_downloader.sites.common.Chat`*) – The chatlog we have downloaded
- **`source`** (*`str`*) – The source of the media associated w the chatlog. URL of the media we have downloaded the log from, or a filepath
- **`settings`** (*`ProcessSettings`*) – Utility class for passing information from the analyzer to the chatlog processor and post-processor

`process_message`(*msg*)

Given a msg object from chat, update appropriate statistics based on the chat

Sample Data

The main JSON data contains a `sample` field comprised of a list of Sample objects.

`class chat_analyzer.dataformat.`**`Sample`**(*startTime: float*, *endTime: float*, *sampleDuration: float = -1*, *startTime_text: str = ''*, *endTime_text: str = ''*, *activity: int = 0*, *chatMessages: int = 0*, *firstTimeChatters: int = 0*, *uniqueUsers: int = 0*, *avgActivityPerSecond: float = 0*, *avgChatMessagesPerSecond: float = 0*, *avgUniqueUsersPerSecond: float = 0*, *_userChats: dict = <factory>*)

Bases: `object`

Class that contains data of a specific time interval of the chat. Messages will be included in a sample if they are contained within [startTime, endTime)

—

[Defined when class Initialized]:

startTime: float
The start time (inclusive) (in seconds) corresponding to a sample.

endTime: float
The end time (exclusive) (in seconds) corresponding to a sample.

[Automatically Defined on init]:

startTime_text: str
The start time represented in text format (i.e. hh:mm:ss)

endTime_text: str
The end time represented in text format (i.e. hh:mm:ss)

sampleDuration: float
The duration (in seconds) of the sample (end-start) NOTE: Should be == to the selected interval in all except the last sample if the total duration of the chat is not divisible by the interval

[Defined w/ default and modified DURING analysis of sample]:

activity: int
The total number of messages/things (of any type!) that appeared in chat within the start/endTime of this sample. Includes messages,notifications,subscriptions, superchats, ... *anything* that appeared in chat

chatMessages: int
The total number of chats sent by human (non-system) users (what is traditionally thought of as a chat) NOTE: Difficult to discern bots from humans other than just creating a known list of popular bots and blacklisting, because not all sites (YT/Twitch) provide information on whether chat was sent by a registered bot or not.

firstTimeChatters: int
The total number of users who sent their first message of the whole stream during this sample interval

[Defined w/ default and modified AFTER analysis of sample]:

uniqueUsers: int
The total number of unique users that sent a chat message across this sample interval (len(self._userChats))

avgActivityPerSecond: float
The average activity per second across this sample interval. (activity/sampleDuration)

avgChatMessagesPerSecond: float
The average number of chat messages per second across this sample interval. (totalChatMessages/sampleDuration)

avgUniqueUsersPerSecond: float
The average number of unique users that sent a chat across this sample interval. (uniqueUsers/sampleDuration)

`sample_post_process()`

After we have finished adding messages to a particular sample (moving on to the next sample), we call sample_post_process() to process the cumulative data points (so we don't have to do this every time we add a message)

Also removes the internal fields that don't need to be output in the JSON object.

Highlight Data

The main JSON data contains a `highlights` field comprised of a lsit of Highlight objects. Currently, there are no platform-specific fields corresponding to Highlights (i.e. highlight objects look the same for all platforms).

`class chat_analyzer.dataformat.`**`Highlight`**(*startTime: float*, *endTime: float*, *description: str*, *type: str*, *peak: float*, *avg: float*)

Bases: `Section`

Highlights reference a contiguous period of time where the provided metric remains above the percentile threshold.

—

type: str
The engagement metric. i.e. "avgActivityPerSecond", "avgChatMessagesPerSecond", "avgUniqueUsersPerSecond", etc. NOTE: It is stored as its converted value (the name of the actual field), NOT the metric str the user provided in the CLI.

peak: float
The maximum value of the engagement metric throughout the whole Highlight (among the samples in the Highlight).

avg: float
The average value of the engagement metric throughout the whole Highlight (among the samples in the Highlight).

7.3.2 Twitch-specific fields:

Chat Analytics Data (Twitch)

`class chat_analyzer.dataformat.`**`TwitchChatAnalytics`**(*duration: float*, *interval: int*, *description: str*, *program_version: str*, *platform: str*, *duration_text: str = ''*, *interval_text: str = ''*, *mediaTitle: str = 'No Media Title'*, *mediaSource: str = 'No Media Source'*, *samples: ~typing.List[~chat_analyzer.dataformat.Sample] = <factory>*, *totalActivity: int = 0*, *totalChatMessages: int = 0*, *totalUniqueUsers: int = 0*, *overallAvgActivityPerSecond: float = 0*, *overallAvgChatMessagesPerSecond: float = 0*, *overallAvgUniqueUsersPerSecond: float = 0*, *highlights: ~typing.List[~chat_analyzer.dataformat.Highlight] = <factory>*, *highlights_duration: float = 0*, *highlights_duration_text: str = ''*, *highlight_percentile: float = 0*, *highlight_metric: str = ''*, *spikes: ~typing.List[~chat_analyzer.dataformat.Spike] = <factory>*, *_overallUserChats: dict = <factory>*, *_currentSample: ~typing.Optional[~chat_analyzer.dataformat.Sample] = None*, *totalSubscriptions: int = 0*, *totalGiftSubscriptions: int = 0*, *totalUpgradeSubscriptions: int = 0*)

Bases: `ChatAnalytics`

Extension of the ChatAnalytics class, meant to contain data that all chats have and data specific to Twitch chats.

NOTE: Most twitch-specific attributes don't make a lot of sense to continously report a per-second value, so we don't!

—

(See ChatAnalytics class for common fields)

[Defined w/ default and modified DURING analysis]:

totalSubscriptions: int
The total number of subscriptions that appeared in the chat (which people purchased themselves).

totalGiftSubscriptions: int
The total number of gift subscriptions that appeared in the chat.

totalUpgradeSubscriptions: int
The total number of upgraded subscriptions that appeared in the chat.

`chatlog_post_process`(*settings*)

After we have finished iterating through the chatlog and constructing all of the samples, we call chatlog_post_process() to process the cumulative data points (so we don't have to do this every time we add a sample).

This step is sometimes referred to as "analysis".

Also removes the internal fields that don't need to be output in the JSON object.

Parameters
`settings` (*`ProcessSettings`*) – Utility class for passing information from the analyzer to the chatlog processor and post-processor

`process_message`(*msg*)

Given a msg object from chat, update common fields and twitch-specific fields

Sample Data (Twitch)

class `chat_analyzer.dataformat.`**`TwitchSample`**(*startTime: float*, *endTime: float*, *sampleDuration: float = -1*, *startTime_text: str = ''*, *endTime_text: str = ''*, *activity: int = 0*, *chatMessages: int = 0*, *firstTimeChatters: int = 0*, *uniqueUsers: int = 0*, *avgActivityPerSecond: float = 0*, *avgChatMessagesPerSecond: float = 0*, *avgUniqueUsersPerSecond: float = 0*, *_userChats: dict = <factory>*, *subscriptions: int = 0*, *giftSubscriptions: int = 0*, *upgradeSubscriptions: int = 0*)

Bases: `Sample`

Class that contains data specific to Twitch of a specific time interval of the chat.

—

[Defined w/ default and modified DURING analysis of sample]:

subscriptions: int
The total number of subscriptions (that people purhcased themselves) that appeared in chat within the start/endTime of this sample.

giftSubscriptions: int
: The total number of gift subscriptions that appeared in chat within the start/endTime of this sample.

upgradeSubscriptions: int
: The total number of upgraded subscriptions that appeared in chat within the start/endTime of this sample.

7.3.3 YouTube-specific fields:

Chat Analytics Data (YouTube)

class `chat_analyzer.dataformat.`**`YoutubeChatAnalytics`**(*duration: float*, *interval: int*, *description: str*, *program_version: str*, *platform: str*, *duration_text: str = ''*, *interval_text: str = ''*, *mediaTitle: str = 'No Media Title'*, *mediaSource: str = 'No Media Source'*, *samples: ~typing.List[~chat_analyzer.dataformat.Sample] = <factory>*, *totalActivity: int = 0*, *totalChatMessages: int = 0*, *totalUniqueUsers: int = 0*, *overallAvgActivityPerSecond: float = 0*, *overallAvgChatMessagesPerSecond: float = 0*, *overallAvgUniqueUsersPerSecond: float = 0*, *highlights: ~typing.List[~chat_analyzer.dataformat.Highlight] = <factory>*, *highlights_duration: float = 0*, *highlights_duration_text: str = ''*, *highlight_percentile: float = 0*, *highlight_metric: str = ''*, *spikes: ~typing.List[~chat_analyzer.dataformat.Spike] = <factory>*, *_overallUserChats: dict = <factory>*, *_currentSample: ~typing.Optional[~chat_analyzer.dataformat.Sample] = None*, *totalSuperchats: int = 0*, *totalMemberships: int = 0*)

Bases: `ChatAnalytics`

Extension of the ChatAnalytics class, meant to contain data that all chats have and data specific to YouTube chats.

NOTE: Most youtube-specific attributes don't make a lot of sense to continously report a per-second value, so we don't!

—

(See ChatAnalytics class for common fields and descriptions)

[Defined w/ default and modified DURING analysis]:

totalSuperchats: int
: The total number of superchats (regular/ticker) that appeared in the chat. NOTE: A creator doesn't necessarily care what form a superchat takes, so we just combine regular and ticker superchats

totalMemberships: int
: The total number of memberships that appeared in the chat.

`process_message`(*msg*)
: Given a msg object from chat, update common fields and youtube-specific fields

Sample Data (YouTube)

class `chat_analyzer.dataformat.`**`YoutubeChatAnalytics`**(*duration: float*, *interval: int*, *description: str*, *program_version: str*, *platform: str*, *duration_text: str = ''*, *interval_text: str = ''*, *mediaTitle: str = 'No Media Title'*, *mediaSource: str = 'No Media Source'*, *samples: ~typing.List[~chat_analyzer.dataformat.Sample] = <factory>*, *totalActivity: int = 0*, *totalChatMessages: int = 0*, *totalUniqueUsers: int = 0*, *overallAvgActivityPerSecond: float = 0*, *overallAvgChatMessagesPerSecond: float = 0*, *overallAvgUniqueUsersPerSecond: float = 0*, *highlights: ~typing.List[~chat_analyzer.dataformat.Highlight] = <factory>*, *highlights_duration: float = 0*, *highlights_duration_text: str = ''*, *highlight_percentile: float = 0*, *highlight_metric: str = ''*, *spikes: ~typing.List[~chat_analyzer.dataformat.Spike] = <factory>*, *_overallUserChats: dict = <factory>*, *_currentSample: ~typing.Optional[~chat_analyzer.dataformat.Sample] = None*, *totalSuperchats: int = 0*, *totalMemberships: int = 0*)

Bases: `ChatAnalytics`

Extension of the ChatAnalytics class, meant to contain data that all chats have and data specific to YouTube chats.

NOTE: Most youtube-specific attributes don't make a lot of sense to continously report a per-second value, so we don't!

—

(See ChatAnalytics class for common fields and descriptions)

[Defined w/ default and modified DURING analysis]:

totalSuperchats: int
The total number of superchats (regular/ticker) that appeared in the chat. NOTE: A creator doesn't necessarily care what form a superchat takes, so we just combine regular and ticker superchats

totalMemberships: int
The total number of memberships that appeared in the chat.

`process_message`(*msg*)

Given a msg object from chat, update common fields and youtube-specific fields

7.3.4 Example JSON output:

An output JSON file might look something like… (*Note, only generic fields are shown. Platform-specific fields would be included in their respective sections: the main analytics data in the main body of the JSON, and the sample data within each sample.*)

```
{
    "duration": 7386.016,
    "interval": 5,
    "description": "description ",
    "program_version": "1.0.0b5",
    "platform": "www.....com",
    "duration_text": "2:03:06",
    "interval_text": "0:05",
    "mediaTitle": "The title of the VOD",
    "mediaSource": "https://www...",
    "samples": [
        {
        "startTime": 0,
        "endTime": 5,
        "sampleDuration": 5,
        "startTime_text": "0:00",
        "endTime_text": "0:05",
        "activity": 10,
        "chatMessages": 9,
        "firstTimeChatters": 9,
        "uniqueUsers": 9,
        "avgActivityPerSecond": 2.0,
        "avgChatMessagesPerSecond": 1.8,
        "avgUniqueUsersPerSecond": 1.8,
        "_userChats": {},
        },
        ...
    ],
    "totalActivity": 42547,
    "totalChatMessages": 42034,
    "totalUniqueUsers": 12533,
    "overallAvgActivityPerSecond": 5.760480345561126,
    "overallAvgChatMessagesPerSecond": 5.691024768968819,
    "overallAvgUniqueUsersPerSecond": 5.66955345060893,
    "highlights": [
        {
        "startTime": 4405,
        "endTime": 4420,
        "description": "avgUniqueUsersPerSecond sustained at or above [8.6]",
        "type": "avgUniqueUsersPerSecond",
        "peak": 11.2,
        "avg": 9.866666666666665,
        "duration": 15,
        "duration_text": "0:15",
        "startTime_text": "1:13:25",
        "endTime_text": "1:13:40"
        },
```

(continues on next page)

(continued from previous page)

```
        ...
    ],
    "highlights_duration": 540,
    "highlights_duration_text": "9:00",
    "highlight_percentile": 93.0,
    "highlight_metric": "usersPSec",
    "spikes": [],
    "_overallUserChats": {},
    "_currentSample": null,
}
```

7.4 Contributing Guide

Thank you so much for considering contributing to the project! Below you'll find the basic steps required to contribute to the project, from setup to pull request.

7.4.1 Add a feature, fix a bug, or write documentation

1. Fork this repository on GitHub: Make your own copy of the repository. Once you've made changes to your own version, you will create a pull request to merge your changes into the main repository.
2. Clone the forked repository: Clone your forked version of the repository to your local machine so that you can make edits to it.

   ```
   git clone git@github.com:YOUR_GITHUB_USERNAME/chat-analyzer.git
   ```

3. Create a new branch: Create a new branch for each different type of feature/fix/edit you would like to make in your forked repository. Each pull request you eventually make should correspond to a single feature, fix, or edit. This allows for an easier review process & improves the overall quality of the commit history.

   ```
   cd chat-analyzer
   git checkout -b <branch_name>
   ```

   The `<branch_name>` should be a short, descriptive name for the feature being worked on. i.e: "add-foo-field", "fix-issue-24", "refactor-help-cmd", etc…
4. Set up your enviornment by installing the developer dependencies: Certain dependencies are used for development that are not required for standard usage. These dependencies allow you to do things such as run tests, build documentation, lint your code, etc…

   ```
   pip install -e ".[dev]"
   ```

5. **Make your changes**: You are now ready to make all of the changes/additions you want to contribute to the project!
6. *[SKIP FOR NOW]:* Testing w/ `pytest` is not yet implemented, so you can ignore this step for now. It will eventually happen at this step.
7. *[SKIP FOR NOW]:* Linting w/ `flake8` is not yet implemented, so you can ignore this step for now. It will eventually happen at this step.

8. As you make your changes, add the new/modified files and commit the files, adding a commit a message with `-m "<message>"` that abides by the Conventional Commits specification. *Do not commit files until they have been tested and linted (see previous two steps).* After fully completing the change you wish to add to the main codebase, push the commits.

   ```
   git add path/to/code.py
   git commit -m 'message'
   git push origin <branch_name>
   ```

   If the change you are making is large or has slightly different components, consider chunking the changes into separate commits. For example:

   ```
   git add chat_analyzer/cli.py
   git add chat_analyzer/analyzer.py
   git commit -m 'feat: added foo command'
   git add docs/
   git commit -m 'docs: guide covers foo command'
   git push origin add-foo-command
   ```

9. Create a pull request: Once pushing your final changes to the branch in your forked repository, create a pull request from your forked repository on GitHub to the main branch of the Chat Analyzer repository.

All done! - Your changes will be reviewed and merged into the main repository by the maintainers. Thanks again for contributing to the project :)

7.5 Changelog

(SEE TODO.md in root directory to see whats up next!)

7.5.1 Releases

LATEST RELEASE:

`1.0.6b3` (8/15/2022)

- When not given an output filename, the automatically generated filepath will use a cleansed version of the media title (to help avoid issues with invalid filenames)

`1.0.6b2` (8/9/2022)

- `highlight_percentile` and `highlight_metric` now included in output.
- README example image fix

`1.0.5b1` (8/3/2022)

- Added support for *youtu.be* links (youtube shortlinks generated from "share" button on YT)

`v1.0.1b1` (7/31/2022)

- Version 1 beta fxnality!
- [Re-release w/ Doc overhaul]

INDEX