GitHub Traffic History

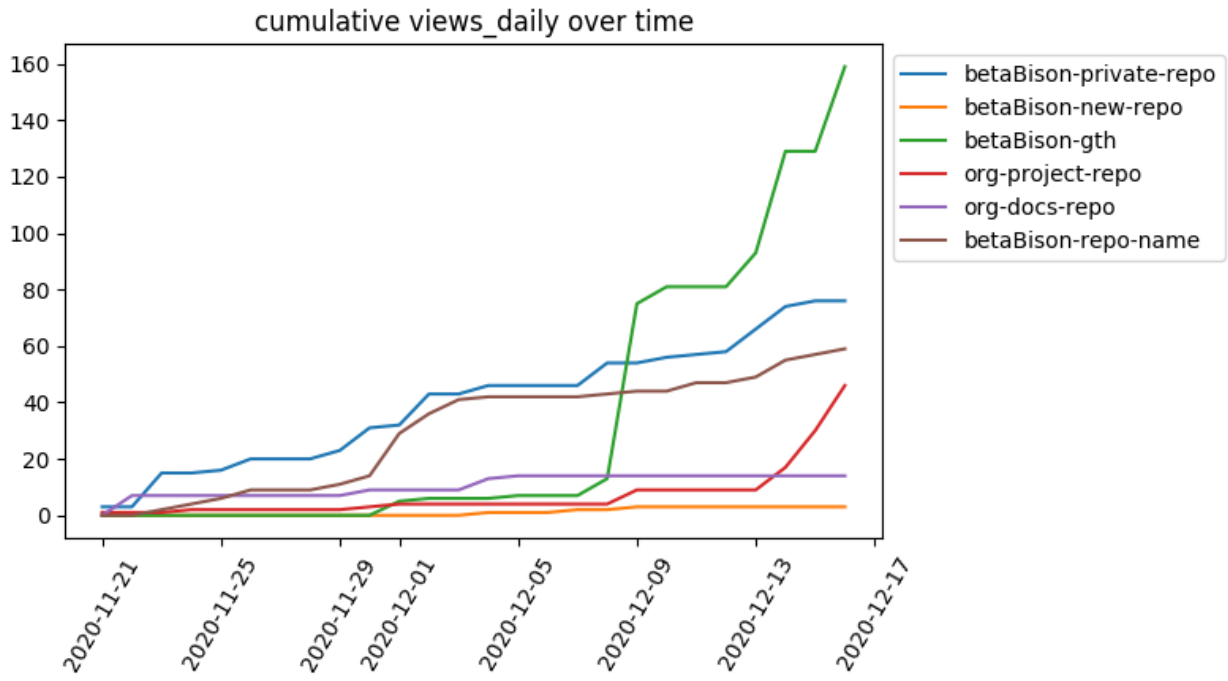
Derek Knowles

Aug 09, 2022

CONTENTS:

This project logs traffic history data for your GitHub repositories and can optionally parse through the data to gain useful insights, plot the data, and send automatic emails with recent trends. This project was inspired by a desire to save long-term traffic history of GitHub repositories to look for patterns that extend beyond the last 14 days (all you can currently see from a respository's Insights page).

This project is broken down into several modules: requesting the traffic data, analyzing the logged traffic data, plotting the logged data, and automatically sending an email with recent history stats. These modules can be run independently. See the *Run Instructions* section for more information on this project's intended modularity.



CHAPTER

ONE

TRAFFIC REQUESTER MODULE

This module uses the GitHub rest API through PyGithub to log traffic data for a user's owner repositories and repositories to which the user has contributed. The output of this module is a csv file with the following traffic information for each repository.

1. stars: number of stars
2. forks: number of forks
3. clones_2weeks: number of clones in the last 14 days
4. clones_uniqeus_2weeks: number of unique clones in the last 14 days
5. views_2weeks: number of views in the last 14 days
6. views_uniques_2weeks: number of views in the last 2 weeks
7. clones_daily: daily clone counts for the last 13 days
8. clones_uniques_daily: daily unique clones for the last 13 days
9. views_daily: daily view counts for the last 13 days
10. views_uniques_daily: daily unique views for the last 13 days
11. referrers_top_10: top referrers to the repository (beta)
12. content_top_10: top content in the repository (beta)

Check out the Setting up the Traffic Requester Module wiki page for more information about installing dependencies, setting up your GitHub authorization key, and stand-alone run instructions.

CHAPTER

TWO

ANALYTICS MODULE

This module parses through the latest raw data from the traffic requester module and concatenates new data to individual repository history logs. The first output of this module is a folder `log/analytics/YYYY-MM-DD/` that contains analytics of the tracked repositories comparing the current metrics to the last time the analytics module was run. The comparative metrics the analytics module logs include:

1. began_tracking: repositories that the user has newly created or to which the user has first contributed
2. ended_tracking: repositories that have been deleted
3. stars_change: additions or deletions of stars to repositories
4. forks_change: additions or deletions of forks of repositories

The second output of this module is the `log/repos/` directory. The analytics module creates a separate folder for each repository and concatenates the metrics from the traffic requester module into individual csv files.

Check out the Setting up the Analytics Module wiki page for more information about installing dependencies and stand-alone run instructions.

CHAPTER

THREE

PLOTTER MODULE

This module contains plotting functions for the analytics data. The plotter has functions for plotting daily metrics or the cummulative summation of metrics over the trackd history period. The plotter has functions for graphing all repositories together (e.g. the top 10 most-viewed repositories) or graphing the metrics for a single repository by itself. Some of the plotter functions also allow you to add a date filter for only plotting historical data after a specied date. Check out the Setting up the Plotter Module wiki page for the list of dependencies and examples of the possible graph options.

CHAPTER

FOUR

EMAIL SENDER MODULE

This module combines the most recently logged analytics metrics and graphs created in the plotter module into an html message. The module then uses the Gmail API to send the html message to a desired receiver. Check out the Setting up the Email Sender Module wiki page for more information about installing dependencies, downloading Gmail authorization credentials, and stand-alone run instructions.

CHAPTER

FIVE

RUN INSTRUCTIONS

This project was intended to be modular; however, the modules do have sequential dependencies on each other. The email sender module depends on metrics created by the analytics module and calls functions from the plotter module. The analytics module depends on traffic data obtained from the traffic requester module. Please go through the wiki page of each module that you would like to use to install needed dependencies or authorizations.

The provided main.py file shows a simple example of running all of the modules consecutively. This file can be run be executing `python3 main.py` in the project directory. You could also only run the traffic requester if you only want the raw data. You could also run the traffic requester weekly, but only run the analytics and email sender once a month. For complete traffic history coverage, the only requirement is that the traffic requester module must be run at least every 13 days (see *Disclaimer #3*).

I suggest implementing a cronjob to automatically run the provided code. Check out the Setting up a cronjob wiki page for examples of how to set up an appropraite cronjob.

If you use all modules, then you should end up with a file structure that looks similar to:

```
├── config/
│   ├── credentials.json              # (opt: email_sender) Gmail credentials file
│   ├── email_token.pickle            # (opt: email_sender) email token once you verify
│   └── settings.ini                  # settings file
├── lib/
│   ├── analytics.py                  # analytics module
│   ├── email_sender.py               # email sender module
│   ├── plotter.py                    # plotter module
│   └── traffic_requester.py          # traffic requester module
├── log/
│   ├── analytics
│       ├── YYYY-MM-DD/
│           ├── YYYY-MM-DD.json       # comparative metrics created by the analytics␣
↪module
│           ├── plot_1.png            # (opt: email_sender) plots created with the␣
↪email sender
│           ├── plot_2.png
│           └── ...
│       ├── YYYY-MM-DD/
│       ├── YYYY-MM-DD/
│       ├── ...
│       ├── plot_1.png                # (opt: plotter) cummulative plots created by␣
↪plotter module
│       ├── plot_2.png
│       └── ...
│   ├── raw/
```

(continues on next page)

(continued from previous page)

```
│       ├── YYYY-MM-DD.csv              # raw traffic history output by the traffic␣
↪requester module
│       ├── YYYY-MM-DD.csv
│       └── ...
│   └── repos/                          # repository metrics separated out by the␣
↪analytics module
│       ├── your_repo_1/
│           ├── clones_2weeks.csv
│           ├── clones_daily.csv
│           ├── clones_uniques_2weeks.csv
│           ├── clones_uniques_daily.csv
│           ├── forks.csv
│           ├── stars.csv
│           ├── views_2weeks.csv
│           ├── views_daily.csv
│           ├── views_uniques_2weeks.csv
│           └── views_uniques_daily.csv
│           ├── plot_1.png              # (opt: plotter) repo plots created with the␣
↪plotter module
│           ├── plot_2.png
│           └── ...
│       ├── your_repo_2/
│       ├── your_repo_3/
│       └── ...
└── main.py                             # main example file
```

CHAPTER

SIX

DISCLAIMERS

1. This project is optimized for readability and not optimized for runtime performance.
2. This project was built and tested with Python3.
3. To obtain continuous data history, run the traffic requester module at least every 13 days. Full clones and visitor information updates hourly, but referring sites and popular content sections only update daily. All traffic data uses UTC+0 timezone no matter where in the world you are [docs]. To avoid saving partial data, the traffic requester throws out the current UTC day's data, hence you're only left with 13 days worth of data instead of the expected 14.
4. If you like the idea of this project but want a nicer front end, check out lukasz-fiszer/github-traffic-stats.
5. If you find bugs or possible improvements, please create an issue or pull request.

6.1 Modules Documentation

6.1.1 Traffic Requester Module

class `lib.traffic_requester.`**`TrafficRequester`**(*config*, *prefix='settings_standard'*, *verbose=False*)

Bases: `object`

traffic requester initialization

Parameters

- **`config`** (*`configparser file`*) – configuration file
- **`prefix`** (*`string`*) – name for log file
- **`verbose`** (*`bool`*) – print verbose debugging statements

`get_history`()

requests traffic history for each repository

Then adds all information to the dataframe

`get_repositories`()

api request for repositories

checks which repositories are owned by the user or to which the user has contributed. Adds all of these repo names to the dataframe.

`log_data`()

save raw data to log file

run()

main run function for traffic requester

6.1.2 Analytics Module

class lib.analytics.**Analytics**(*prefix='settings_standard'*, *verbose=False*)

Bases: `object`

analytics initialization

Parameters

- **prefix** (*string*) – name for log file
- **verbose** (*bool*) – print verbose debugging statements

check_dirs()

check and create directories

create log directories if they don't yet exist and check which raw logs need to be analyzed.

Returns
analytics_needed – the raw logs that do not yet have a corresponding analytics directory

Return type
list

check_forks_change()

Checks forks counts

checks whether the forks count has changed and appends any changes to self.forks_change

check_stars_change()

Checks start counts

checks whether the stars count has changed and appends any changes to self.stars_change

check_tracking_change()

check tracked repositories

checks which repositories are beginning to be tracked or have stopped being tracked.

create_repo_dirs()

create log directories if they don't yet exist

full2dir(*fullname*)

changes full repository name into a directory name

Parameters
fullname (*string*) – full repository name

Returns
dirname – new directory name

Return type
string

load_log()

load_log file into dataframe

log_analytics()

Logs the analytics to a json file

run()

main run function for analytics

sort_raw_data()

Sort through each of the main metrics for each repository

update_daily_metric(*ri*, *col_name*)

update metrics that are daily

this function reads through the old data and only adds new daily values

Parameters

- **ri** (*int*) – row of dataframe to read from
- **col_name** (*string*) – column name and thus file name for the specific metric

update_nondaily_metric(*ri*, *col_name*)

update nondaily metrics

update metrics that are not daily, this function simply appends the newest value to the log file

Parameters

- **ri** (*int*) – row of dataframe to read from
- **col_name** (*string*) – column name and thus file name for the specific metric

6.1.3 Plotter Module

class lib.plotter.**Plotter**(*prefix='settings_standard'*)

Bases: object

Plotter class.

Parameters

prefix (*string*) – name for log file

create_email_plots(*date_cur*, *date_prev=None*)

create and save some plots for use in an email

Parameters

- **date_cur** (*string*) – YYYY-MM-DD, date of current analytics file
- **date_prev** (*string*) – YYYY-MM-DD, date of previous analytics file

Returns

fig_paths – [string,string,...]) : list of strings of the location of where each figure is saved

Return type

list

create_plots(*verbose=False*)

create a bunch of plots as desired

Parameters

verbose (*bool*) – print verbose debugging statements

plot_daily_metrics(*col_name*, *type='daily'*, *top_num=None*, *date_filter=None*)

plot and daily metrics.

The plots get saved to default location if there is no date filter implmented

Parameters

- **col_name** (*string*) – name for filename and column name
- **type** (*string*) – either "cumsum" or "daily". "cumsum" will plot the cumulative sum of the column over time while "daily" will plot the daily change over time
- **top_num** (*int*) – number of top repositories (according to cumulative sum) to show in the graph. Repos with a cumulative value of 0 will still not be plotted
- **date_filter** (*string*) – "YYYY-MM-DD", all data after this date (inclusive) will be plotted. None means all data will be plotted

Returns
fig – new figure

Return type
matplotlib figure

plot_repo_metric(*repo_dir*, *metric_name*, *type*)

plots individual repository metrics and saves the plots

Parameters

- **repo_dir** (*string*) – filepath to the repository logs
- **metric_name** (*string*) – name for metric and column name
- **type** (*string*) – either "cumsum" or "daily". "cumsum" will plot the cumulative sum of the column over time while "daily" will plot the daily change over time

save_and_close(*fig*, *plt_file*)

saves and closes the figure

Parameters

- **fig** (*matplotlib fig*) – figure object
- **plt_file** (*string*) – filepath for the figure

update_repo_plots(*verbose=False*)

update all repo plots.

This function in particular takes a long amount of time. You could not call this function if for some reason you need to run this code faster

Parameters
verbose (*bool*) – print verbose debugging statements

6.1.4 Email Sender Module

class lib.email_sender.**EmailSender**(*config*, *prefix*, *verbose=False*)

Bases: `object`

email sender initialization

Parameters

- **config** (*configparser file*) – configuration file
- **prefix** (*string*) – name for log file
- **verbose** (*bool*) – print verbose debugging statements

build_html_message()

Build HTML message

create the bulk of the html message by combing lots of strings together that include tracked analytics and plots that were created

Returns
msg – long string that contains the html message

Return type
string

build_service()

builds gmail api service.

Code copied with minor edits from https://developers.google.com/gmail/api/quickstart/python

Returns
service – gmail api service

Return type
gmail api

create_mixed_message(*message_html*)

Create a message for an email.

Copied with edits from https://developers.google.com/gmail/api/guides/sending Also see this answer for how to add attachments https://stackoverflow.com/questions/1633109/

Parameters
message_html (*string*) – html text message to be sent

Returns
msg_object – email object

Return type
base64url encoded email object

prep_attachments()

Prepare attachements.

call the plotter function and correlate figure names with the figures that were created

run()

main run function for the email sender

send_message(*service*, *user_id*, *message*)

Send an email message.

Copied with minor edits from https://developers.google.com/gmail/api/guides/sending

Parameters

- **service** (*Gmail API service instance*) – Authorized Gmail API
- **user_id** (*string*) – User's email address. The special value of "me" can be used to indicate the authenticated user.
- **message** (*string*) – Message to be sent.

Returns

message – the sent message

Return type

message object

PYTHON MODULE INDEX

l

INDEX