

STRAVA Webscraping Tool Documentation

Mathis S

January 2019

Contents

1	Introduction	3
2	Tutorial	3
2.1	Setup	3
2.1.1	Python	3
2.1.2	Libraries	4
2.1.3	Downloading the Scraper	4
2.1.4	secret.py	4
2.2	Running the crawler	5
2.3	Writing your own code	7
3	Function Documentation	10
3.1	acc.py	10
3.2	activities.py	10
3.3	anon.py	12
3.4	clubs.py	13
3.5	ego.py	14
3.6	expo.py	15
3.7	follows.py	15
3.8	main.py	16
3.9	network.py	16
3.10	plotting.py	18
3.11	sampling.py	18
3.12	secret.py	20
A	Using the Command Line	21
B	Code	23
B.1	Unaltered main.py	23
B.2	“Empty” main.py	24
B.3	Example main.py for Section 2.3	25
B.4	Snippet for importing a network in R	26
B.5	tiesDict Format	26
C	Privacy and Anonymization	27
C.1	Process	27
C.2	Collected Data	29

1 Introduction

This tool can be used to scrape network information from the social networking site STRAVA¹. It consists of 12 python files (.py) and a Readme. This Documentation will consist of 2 parts: Firstly, we will go over a small example of how this tool is supposed to be used. Secondly, we will explain all functions and how you might use them in your scraping or to expand the functionalities.

2 Tutorial

2.1 Setup

Note: You can use a complete IDE like Anaconda², but I will explain how to setup Python and a standalone text editor, since that is what I am using.

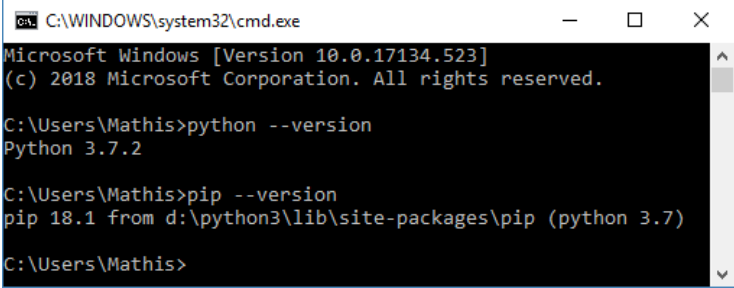
2.1.1 Python

To use this tool, you will have to install some version of Python 3, preferably the latest one³. Make sure to also install pip (It will be installed with python if you do not change the defaults).

We will also be editing part of the code to specify what data we want to collect. For that we need a text editor. I like to use Sublime Text 3⁴, but Notepad++⁵ or any other editor works equally well.

After completing the setup, open a command prompt⁶ and type `python --version` to verify it is working. Also test pip by running `pip --version`.

It should look something like Figure 1



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Mathis>python --version
Python 3.7.2

C:\Users\Mathis>pip --version
pip 18.1 from d:\python3\lib\site-packages\pip (python 3.7)

C:\Users\Mathis>
```

Figure 1: Python and Pip successfully installed.

¹<https://www.strava.com/>

²<https://www.anaconda.com/download/>

³You can get it from here: <https://www.python.org/>

⁴<https://www.sublimetext.com/3>

⁵<https://notepad-plus-plus.org/>

⁶See Appendix A for information on how to open and use a command prompt.

2.1.2 Libraries

We now need to install five libraries; numpy⁷, pandas⁸, matplotlib⁹, networkx¹⁰, pycryptodome¹¹ and robobrowser¹².

Just run

```
pip install numpy
pip install pandas
pip install matplotlib
pip install networkx
pip install pycryptodomex (notice the x)
pip install robobrowser.
```

Side note: If you ever get an error saying:

`ModuleNotFoundError: No module named 'SomeModule'`

it can probably be fixed by running

```
pip install somemodule.
```

2.1.3 Downloading the Scraper

If you are reading this, you should already have the files necessary to run the scraper. If not, get them from <https://gitlab.socsci.ru.nl/M.Sackers/strava-scraper>.

2.1.4 secret.py

The last thing we need to do before getting the data is to fill some secret information into the `secret.py` file.

You need a STRAVA account to access some of the information we are after, so add that information here. I strongly advise you not to use your main STRAVA account, since this account will be doing a large number of requests. For now STRAVA does not seem to mind, but you never know. A VPN might also be a good idea.

Since we are working in code, you might need to escape special characters in your password. Just add a `\` before every `\` (resulting in `\\`) and `'` (`\'`).

The `keyPass` is a 16-byte key to en- and de-crypt the ids. Just enter 16 random characters, you don't have to remember them¹³.

```
1 email = 'user123@gmail.com'
2 password = 'my\'password' # my'password
3 keyPass = b'd2QRL05FzllEaJi0'
4 projectPath = 'unnamedProject' # leave this as is
```

⁷<http://www.numpy.org/>

⁸<https://pandas.pydata.org/>

⁹<https://matplotlib.org/users/installing.html>

¹⁰<https://networkx.github.io/>

¹¹<https://pycryptodome.readthedocs.io/en/latest/src/introduction.html>

¹²<https://robobrowser.readthedocs.io/en/latest/>

¹³Maybe take them from here: <https://www.random.org/strings/?num=1&len=16&digits=on&upperalpha=on&loweralpha=on&unique=on&format=html&rnd=new>

As the comment in line 4 already states, don't touch that line. It's used internally and changed from another file to which we'll get shortly.

You should, of course, never share any of the information of the `secret.py` or the file itself.

That was the whole setup!

2.2 Running the crawler

To run the crawler, open a command prompt inside the folder with all of the webscraper's files¹⁴. Now run `python main.py` and wait a few minutes¹⁵. You should be seeing a graph of a network and notice that a folder called `tutorial` was created. (Also see Figure 2)

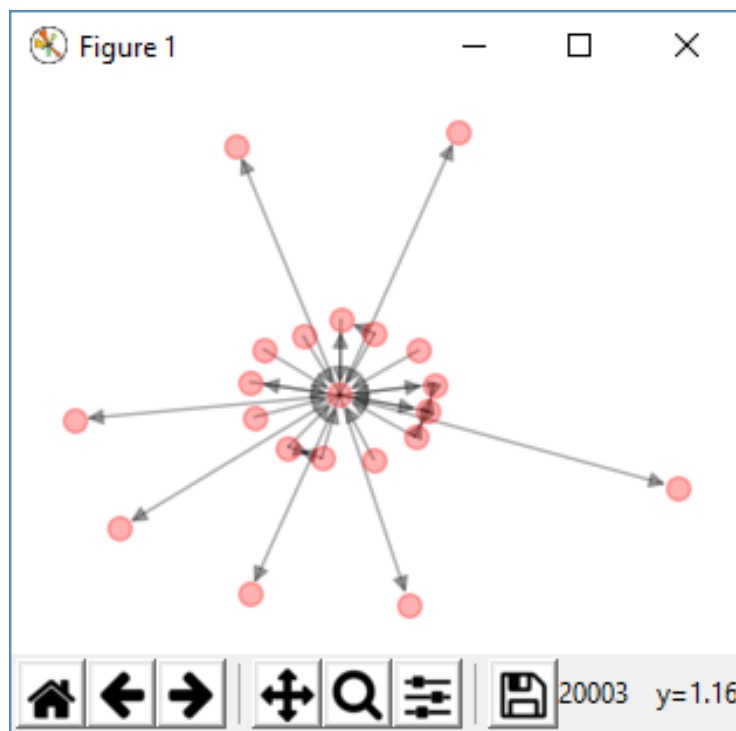


Figure 2: What you should see after running `python main.py`.

Close the graph and open `main.py` in your text editor¹⁶.

This is the only file you will have to change if you just want to use the tool with its current functionality.

Let's look at the code of what we have just witnessed:

¹⁴If you need help with that, check Appendix A

¹⁵Depending on your luck, this might take half an hour.

¹⁶Alternatively the code is also available in its entirety in Appendix B.1

Lines 1 and 2 import two standard libraries, while lines 4 and 7-16 import the other files of the scraper. Line 5 defines the name of the project, which is used as the name of the folder into which all data is saved. Lines 17-20 are a multi-line comment (identified by being enclosed in triple quotes) describing the file. Lines 23-45 define two functions, but how exactly they work is not important.

Line 48 calls a function called `setDelay`, which is part of the `acc.py` file. It passes the number 500 as an argument. What this function does, is tell the scraper that it should wait a certain number of milliseconds between its requests to the STRAVA server. We do this to not flood or overload STRAVA's server. In this case we wait 500 milliseconds (=0.5s).

```
48 acc.setDelay(500)
```

Line 49 is a inline comment (everything after the `#` is ignored). We'll come back to this later.

Finally, line 50 calls the `setup()` function defined earlier. This function makes sure all the necessary folders are there.

Line 53 starts with our first meaningful instruction:

```
53 sampled = sampling.getAthletes(['New York', 'London'], number=1)
```

We define a new variable called `sampled` and assign it the return value of a function. That function is part of the `sampling.py` as indicated by prefixing it with `sampling..` The function itself (`getAthletes`) takes two arguments: a list of search terms and a number of athletes. We provide a list of *New York* and *London*. We also just need one athlete for our demonstration (`number=1`). What this function does behind the scenes is using these search terms on the STRAVA athlete search and picking random athletes until we have at least as many as specified. It also makes sure that these athletes have public profiles, such that we can get their information.

```
55 tiesDict = network.egoFollowsNetwork(sampled[0])
```

In line 55 we use a function from the `network.py` file called `egoFollowsNetwork`. It takes the id of an athlete and returns an ego-centric network for that athlete based on follows¹⁷. We just take the first athlete that we have just sampled by indicating in square brackets the index of the element we want from the list (`sampled[0]`)¹⁸.

```
57 networkAthletes = network.athletesInNetwork(tiesDict)
```

Line 57 uses another function from the `network.py` file: `athletesInNetwork`. It takes as argument a `tiesDict` such as the one we created in line 55 and it returns a list of all athletes that appear in the network. This list we save in a variable called `networkAthletes`.

We then use two functions from the `expo.py` file:

¹⁷An example of how exactly the `tiesDict` is structured can be found in Appendix B.5.

¹⁸Indexing in most programming languages starts with 0 for the first element.

```

59 expo.saveGraphAsMatrix(tiesDict, 'followNet-{}.dat'.format(sampled[0]))
60 expo.saveEgos(networkAthletes, 'egos.dat', labels=True, type=True)

```

`saveGraphAsMatrix` in line 59 does exactly what it says: It saves a graph (= `tiesDict`) as a matrix. The second argument tells it where to save this matrix. Here we have chosen to save it under the name `followNet-123.dat`, where 123 is the id of the ego¹⁹. If you wanted to do a more in-depth analysis of this network, you could import it in R with the snippet presented in Appendix B.4.

Line 60 creates another file called `egos.dat`. It has a row with labels at the top (due to the argument `labels=True`). Then every row lists the id of the athletes in `networkAthletes`, and as a second column their type (due to the argument `type=True`). Type here means, whether they mostly *swim*, *run* or *ride*. If they do two or more it will list them as *mix*.

You can open both `followNet-123.dat` and `egos.dat` (located in the new `tutorial`-folder) with your favorite text editor and have a quick look.

Finally, line 62 uses a function from the `plotting.py` file that created the graph seen in Figure 2. It takes the `tiesDict` as an argument. The `plotting.py` functionality is not very powerful, so I recommend to use the exported `followNet-123.dat` and use e.g. `igraph` in R²⁰ to do your visualization.

Line 64 is a comment informing you, that your own code should not extend beyond this line. Line 65 saves the anonymization dictionary. When you alter the code, just leave this line last and don't worry about it. Read Appendix C if you want more information on the anonymization this tool provides.

2.3 Writing your own code

Let's walk through how you would approach writing your own code. First of all, remove lines 53 until 64, between the start and end comments. You're left with the code seen in Appendix B.2.

Now we can change the project name/path in line 5 such that we do not mix this data with the data we collected in the first part of the tutorial.

```

5 secret.projectPath = 'myFirstCrawl'

```

We can remove the `#` and the space at the beginning of line 49, such that when we run our script all the data we had collected is removed. This is not necessary, and also forces us to ask STRAVA for all the data every time we run our script. Leave it commented out if in doubt.

Now put some empty lines between the start and end comment. On line 53 we should first get a sample of athletes. We can again use the `getAthletes` function from the `sampling.py` file. We could even use it without arguments, as it has some defaults specified. You can see that in its specification in Section 3.11.

¹⁹If you want to know how that `format` works, see <https://docs.python.org/3/library/stdtypes.html#str.format>

²⁰<https://igraph.org/r/>

It states:

```
getAthletes(searches=['Nijmegen'], number=50)
```

This means, if no search list is specified, a list consisting only of *Nijmegen* is used and if no number is specified, 50 is used.

For the purposes of this tutorial it does not matter what parameters you pass to this function, but I would recommend taking a number between 30 and 100, such that you have some samples, but it also doesn't take too long.

As search terms I would generally recommend either countries or cities. This way you will get a good number of results.

Here's what I'm using:

```
53 sample = sampling.getAthletes(['Germany', 'Jim', 'Jane'])
54 print('Number of samples:', len(sample))
```

I also added a line that will print the length of the sample list to the output.

Next, we might want to only have a look at the women in our sample.

```
55 women = sampling.filterGender(sample, 'F')
56 print('Number of women:', len(women))
```

Here we save only the athletes where gender matches 'F' in the list called `women`. Due to the way the gender is scraped this might take a long time, so you might want to use other filtering functions available in `sampling.py`. You can filter depending on the time spent sporting in a given month, the time they started sporting, the last time they logged an activity or the type of sport they predominantly do. Check Section 3.11 for a list of functions available and how to use them.

Next, we might want to filter this list of women down to only women that started sporting in January of any year.

```
57 womenJan = sampling.filterStartDate(women, 1)
58 print('Number of women that started in January:', len(womenJan))
```

We did not provide a third argument for `filterStartDate` which means the argument `year` will stay at its default value `None`.

Let's export the information to a file. We again use `expo.saveEgos()` and specify the information to save.

```
59 expo.saveEgos(sample, 'myExports/samples.dat', gender=True)
60 expo.saveEgos(women, 'myExports/women.dat', gender=True,
61               startDate=True)
62 expo.saveEgos(womenJan, 'myExports/womenJan.dat',
63               sportingTime=[(1,2017),(7,2017),
64                              (1,2018),(7,2018)])
```

Line 59 exports the 50 samples with their gender. The file name is actually a path and a file name: `myExports/` specifies the folder in which to put the file. **If the folder does not exist, this will crash**, so make sure it exists (or don't save into sub-folders). Finally, `samples.dat` is the name of the file.

Line 60 (and 61) exports the women with their gender (which will be F for all of them) and their start dates.

Line 62 (and 63, 64) exports the women that started in January with the time in minutes they spent sporting in January of 2017, July of 2017, January of 2018 and July of 2018.

The information saved in lines 59-61 is information that we already downloaded from STRAVA when filtering, so that will not make unnecessary calls to their website.

Finally, we are interested in how to get network information.

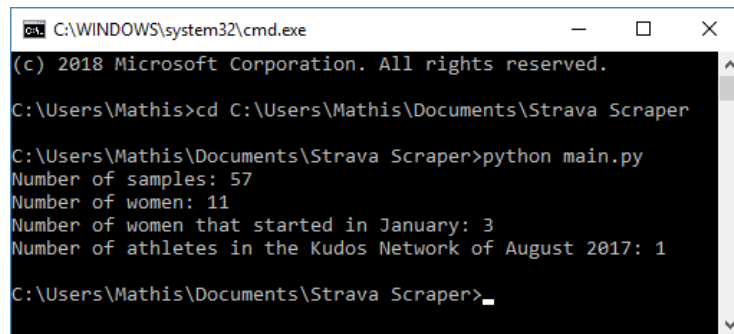
```
65 athleteId = sample[0]
66 if len(womenJan) > 0:
67     athleteId = womenJan[0]
68 elif len(women) > 0:
69     athleteId = women[0]
70 tiesDict = network.egoKudosNetwork(athleteId, 8, 2017)
71 print('Number of athletes in the Kudos Network of August 2017:',
72       len(athletesInNetwork(tiesDict)))
```

We just select the first id of our sample in line 65 as our subject. If we have women who started in January, we select the first one of those to overwrite the first one of the sample. If we do not have women who started in January, but we do have women we select the first of those.

This way we can be sure in line 70 that we do have some `athleteId`. We get the ego-centric network of who gave whom kudos in August of 2017.

We then just print the number of athletes in that network to the console.

Figure 3 shows the output of this code. The complete code can be found in Appendix B.3. This code does, however, take quite a while to complete running (approx. 60 minutes).



```
C:\WINDOWS\system32\cmd.exe
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Mathis>cd C:\Users\Mathis\Documents\Strava Scraper

C:\Users\Mathis\Documents\Strava Scraper>python main.py
Number of samples: 57
Number of women: 11
Number of women that started in January: 3
Number of athletes in the Kudos Network of August 2017: 1

C:\Users\Mathis\Documents\Strava Scraper>_
```

Figure 3: What you should see after running the altered version of `main.py` described above.

3 Function Documentation

For completeness sake all functions are listed and described here. I have marked the ones that I believe are important to a user (as opposed to a developer) with an `*`.

3.1 `acc.py`

Manages the ‘browser’ behind the scenes.

`login(br=RoboBrowser(history=True, parser='html.parser'))`

`br` = the RoboBrowser to use

Logs into STRAVA using the account data provided in `secret.py`. An instance of RoboBrowser that is logged in will be returned.

`getBrowser()`

Returns the RoboBrowser that is logged in or creates one if need be.

`setDelay(delay)`

`delay` = the delay in milliseconds

Sets the delay the scraper will use between requests to STRAVA.

`browse(url)`

`url` = the url to open (you can leave out ‘https://www.strava.com/’)

Opens the requested url in RoboBrowser while respecting the delay defined with `setDelay()`. An instance of RoboBrowser that has opened the url will be returned.

3.2 `activities.py`

Functions that retrieve activities and information about the activities.

`getActivities(fAthleteId, month, year)`

`fAthleteId` = fake id of the athlete

`month` = month of the activities (1 = Jan, 2 = Feb, ..)

`year` = year of the activities to get

Returns a list of fake activity ids that the athlete participated in during the given month.

This information is either retrieved online or, if it was retrieved before already, it is loaded from the `/activities/` folder. We are also getting some information for free here, so we're saving that too: Type, Distance, Pace, etc.

`getKudosForActivity(fActivityId)`

`fActivityId` = fake id of the activity

Returns a list of fake athlete ids that have given kudos for this activity.

This information is either retrieved online or, if it was retrieved before already, it is loaded from the `/kudos/` folder.

`getCommentersForActivity(fActivityId)`

`fActivityId` = fake id of the activity

Returns a list of fake athlete ids that have commented on this activity.

This information is either retrieved online or, if it was retrieved before already, it is loaded from the `/commenters/` folder.

`getRodeWithsForActivity(fActivityId)`

`fActivityId` = fake id of the activity

Returns a list of fake athlete ids that have participated in this activity.

This information is either retrieved online or, if it was retrieved before already, it is loaded from the `/rodeWiths/` folder.

* `getKudos(fAthleteId, month, year)`

`fAthleteId` = fake id of the athlete

`month` = month of the activities (1 = Jan, 2 = Feb, ..)

`year` = year of the activities to get

Returns a list of dicts with two entries each, `'id'` and `'weight'`. `'id'` is the fake id of an athlete and `'weight'` the amount of kudos that athlete has given the ego in this month.

Example return where fake id 6 has given one kudo and user 3 has given 4:

```
[{'id': 6, 'weight': 1}, {'id': 3, 'weight': 4}]
```

`getActivityType(fActivityId)`

`fActivityId` = fake id of the activity

Returns the type of activity. This can be `'run'`, `'swim'`, `'ride'`, `'workout'`, `'virtualride'` or `'club'` (Maybe more, but these are the ones I came across). `'club'` activities are events where someone joined a club.

`getActivityTime(fActivityId)`

`fActivityId` = fake id of the activity

Returns the time in minutes it took the athlete to complete the activity. This can also be `None` if no time is public.

`getActivityDistance(fActivityId)`

`fActivityId` = fake id of the activity

Returns the distance the athlete ran/rode during the activity. It's a messy string though. This can also be `None` if no distance is public.

`getActivityPace(fActivityId)`

`fActivityId` = fake id of the activity

Returns the pace of the activity. It's a messy string though. This can also be `None` if no pace is public.

`getActivityElevationGain(fActivityId)`

`fActivityId` = fake id of the activity

Returns the elevation gain of the activity. It's a messy string though. This can also be `None` if no elevation gain is public.

`getActivityAvgPower(fActivityId)`

`fActivityId` = fake id of the activity

Returns the average power of the activity. It's a messy string though. This can also be `None` if no average power is public.

3.3 anon.py

Functions that convert between real and fake ids.

`loadKey(key=secret.keyPass)`

`key` = 16-byte key that was used to encrypt the keyfile

Returns the decrypted keyfile or an empty one if no file was found.

`saveKey(key=secret.keyPass)`

`key` = 16-byte key that will be used to encrypt the keyfile

Writes the dict to `id.key`.

`*fake2athlete(fAthleteId)`

`fAthleteId` = fake id of the athlete

Returns the real id of the athlete (and the API token [currently unused])

`*athlete2fake(athleteId, APIId=False)`

`athleteId` = real id of the athlete `APIId` = API token of the athlete (optional)

Returns the fake id of the athlete and saves the real one.

`fake2activity(fActivityId)`

`fActivityId` = fake id of the activity

Returns the real id of the activity

`activity2fake(activityId)`

`activityId` = real id of the activity

Returns the fake id of the activity and saves the real one.

`fake2club(fClubId)`

`fClubId` = fake id of the club

Returns the real id of the club

`club2fake(clubId)`

`clubId` = real id of the club

Returns the fake id of the club and saves the real one.

3.4 clubs.py

Get club-related information.

`getClubMembers(fClubId)`

`fClubId` = fake id of the club

Returns the admins of the club and the members of the club.

`getClubMembers(fClubId)`

`fClubId` = fake id of the club

Returns the admins of the club and the members of the club.

`getClubFollows(fClubId, tiesDict={})`

`fClubId` = fake id of the club
`tiesDict` = dictionary to which the information will be added (optional)
Returns a `tiesDict` (see Appendix B.5).

3.5 ego.py

Information about athletes.

`getStartDate(fAthleteId)`

`fAthleteId` = fake id of the athlete

Returns the month and year that the athlete has recorded their first activity.

`getLastDate(fAthleteId)`

`fAthleteId` = fake id of the athlete

Returns the month and year that the athlete has recorded their latest activity.

`getGender(fAthleteId)`

`fAthleteId` = fake id of the athlete

Returns the gender of the athlete ('M', 'F', or None)

`getLocation(fAthleteId)`

`fAthleteId` = fake id of the athlete

Returns the location of the athlete as a messy string.

`getType(fAthleteId)`

`fAthleteId` = fake id of the athlete

Returns the type of activity that the athlete mainly does, or 'mix' or 'none'.

`getSportingTime(fAthleteId, month, year)`

`fAthleteId` = fake id of the athlete

`month` = month of the activities

`year` = year of the activities

Returns the number of minutes the athlete has spent on activities in the given month.

3.6 expo.py

Functions that convert data to .csv files.

`graphToMatrix(tiesDict, labels=True)`

`tiesDict` = the dictionary that will be converted to a matrix. See B.5

`labels` = adds the ids as row/column 0 if True

Returns an np.array `m`, where the `m[a, b]` is the weight of the tie from `a` to `b`.

`*saveGraphAsMatrix(tiesDict, filename, labels=True)`

`tiesDict` = the dictionary that will be saved as a matrix. See Appendix B.5

`filename` = the name of the file

`labels` = adds the ids as row/column 0 if True

Writes the `tiesDict` to a file with name `filename` inside the projectfolder. The `tiesDict` is transformed to a matrix as described in `graphToMatrix(tiesDict, labels=True)` above. The cells in the file are separated by spaces and newlines.

`*saveEgos(egos, filename, labels=True, type=False, gender=False, startDate=False, lastDate=False, sportingTime=[])`

`egos` = list of fake athlete ids

`filename` = the name of the file

`labels` = adds labels as the first row if True (optional)

`type` = adds the type of activity the athlete does if True (optional)

`gender` = adds the gender of the athlete if True (optional)

`startDate` = adds the month and year of the athletes first activity if True (optional)

`lastDate` = adds the month and year of the athletes latest activity if True (optional)

`sportingTime` = a list of (month, year) tuples. Adds the sporting time in minutes for the given months (optional)

Writes the requested information to a file inside the project folder with name `filename`. The cells in the file are separated by spaces and newlines.

3.7 follows.py

Gets information on who follows whom.

`getFollowers(rAthleteId)`

`rAthleteId` = real id of the ego-athlete

Returns a list of fake ids that follow the ego-athlete.

`getFollowing(rAthleteId)`

`rAthleteId` = real id of the ego-athlete

Returns a list of fake ids that follow the ego-athlete.

`*getFollows(fAthleteId)`

`fAthleteId` = fake id of the ego-athlete

Returns a dictionary with two lists of fake ids that follow the ego-athlete. An example where the ego-athlete follows fake ids 1, 2 and 8, while they are being followed by 1, 2 and 3.

```
1 {  
2     "followings": [1, 2, 3],  
3     "followers": [1, 2, 8]  
4 }
```

3.8 main.py

The main file that imports the others. Specify what you want to do in this file. Check the tutorial if you don't know how to use it.

`removeData()`

Removes all data collected.

`setup()`

Creates the folders the webscraper needs, if they do not exist yet.

3.9 network.py

Makes and manages networks.

`*athletesInNetwork(tiesDict)`

`tiesDict` = network dictionary (see Appendix B.5)

Returns a list of all athlete ids that are in the network.

`trim(fAthleteId, tiesDict)`

`fAthleteId` = id that should be removed from `tiesDict`

`tiesDict` = network dictionary (see Appendix B.5)

Returns the `tiesDict` without all references to `fAthleteId`.

`trimEgo(fAthleteId, tiesDict)`

`fAthleteId` = id of the ego

`tiesDict` = network dictionary (see Appendix B.5)

Returns the `tiesDict` without all ids that do not have a direct connection to the ego.

`trimClub(fClubId, tiesDict)`

`fClubId` = fake id of the club

`tiesDict` = network dictionary (see Appendix B.5)

Returns the `tiesDict` without all ids that are not in the club.

`getClubFollows(fClubId)`

`fClubId` = fake id of the club

Returns a `tiesDict` with all follow relations of all club-members (see Appendix B.5). This also includes ties to athletes outside the club.

`getEgoFollows(fAthleteId, depth=0, tiesDict={}, todo=[], done=[])`

`fAthleteId` = fake id of the ego-athlete

`depth` = how deep the network should be (friend of a friend) 0 for only direct ties to ego

`tiesDict` = dictionary where the info is saved into (see Appendix B.5)

`todo` = used internally. List of ids that have to be investigated

`done` = used internally. List of ids that have been investigated

Returns a `tiesDict` with the ego follows network.

`getEgoKudos(fAthleteId, month, year, depth=0, tiesDict=, todo=[], done=[])`

`fAthleteId` = fake id of the ego-athlete

`month` = month from which to collect the kudos

`year` = year from which to collect the kudos

`depth` = how deep the network should be (friend of a friend) 0 for only direct ties to ego

`tiesDict` = dictionary where the info is saved into (see Appendix B.5)

`todo` = used internally. List of ids that have to be investigated

`done` = used internally. List of ids that have been investigated

Returns a `tiesDict` with the ego kudos network of the given month.

`*egoFollowsNetwork(fAthleteId, depth=1)`

`fAthleteId` = fake id of the ego-athlete

`depth` = how deep the network should be (friend of a friend)

0 if you only want direct connections between ego and alters

1 if you also want connections between alters

Returns a `tiesDict` (see Appendix B.5) with the ego follows network.

```
*egoKudosNetwork(fAthleteId, month, year, depth=1)
```

`fAthleteId` = fake id of the ego-athlete

`month` = month from which to collect the kudos

`year` = year from which to collect the kudos

`depth` = how deep the network should be (friend of a friend)

0 if you only want direct connections between ego and alters

1 if you also want connections between alters

Returns a `tiesDict` (see Appendix B.5) with the ego kudos network.

```
*clubFollowsNetwork(fClubId)
```

`fClubId` = fake id of the club

Returns a `tiesDict` (see Appendix B.5) with all follow connections in the club.

3.10 plotting.py

Uses `networkx` to plot a network. This is not very powerful or useful, but just meant as a quick graphical demo.

```
graphToDf(tiesDict)
```

`tiesDict` = network to convert (see Appendix B.5)

Returns a pandas dataframe with a *from* and *to* column.

```
*showGraph(tiesDict, directed=True, labels=False))
```

`tiesDict` = network to convert (see Appendix B.5)

`directed` = whether the edges should be directed

`labels` = whether the vertices should be labelled

Plots the `tiesDict` on screen.

3.11 sampling.py

Finds random athletes and filters athletes.

```
validId(rAthleteId)
```

`rAthleteId` = real id of the athlete

Returns True if the id exists and the profile is public.

`getAthletes(searches=['Nijmegen'], number=50)`

`searches` = list of search terms

`number` = least number of athletes to get

Returns a list of at least `number` fake athlete ids using the search terms provided in `searches`. It is recommended to use cities and/or countries as search terms. We can maximally look at 50 result pages per search term with a maximum of 20 public athletes per page. If your `number` is too high or your search terms don't result in many results, you might get fewer athletes back than you searched.

`filterType(athletes, type='run')`

`athletes` = list of fake athlete ids

`type` = the type an athlete should have

Returns only those athletes that match the type.

`filterStartDate(athletes, month=None, year=None)`

`athletes` = list of fake athlete ids

`month` = the month an athlete should have logged their first activity (or None)

`year` = the year an athlete should have logged their first activity (or None)

Returns only those athletes that started in the given month and year. If either month or year is None, only the year or month respectively will be checked.

`filterGender(athletes, gender='M')`

`athletes` = list of fake athlete ids

`gender` = the gender an athlete should have

Returns only those athletes that have the stated gender.

`filterSportingTime(athletes, month, year, min=None, max=None)`

`athletes` = list of fake athlete ids

`month` = the month to check

`year` = the year to check

`min` = the minimum amount of minutes an athlete should have logged (or None)

`max` = the maximum amount of minutes an athlete should have logged (or None)

Returns only those athletes that have logged a certain number of minutes in the given month and year. If either min or max is None only the other one is checked.

3.12 `secret.py`

Check Section 2.1.4 for information on this file and its contents.

A Using the Command Line

You can open a command line by pressing **Win + R** (Resulting in Figure 4), typing **cmd** and pressing **Enter** (Resulting in Figure5).

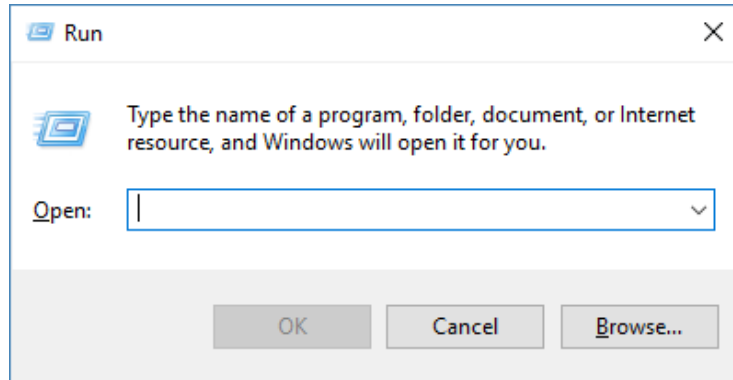


Figure 4: Window after pressing Win + R.

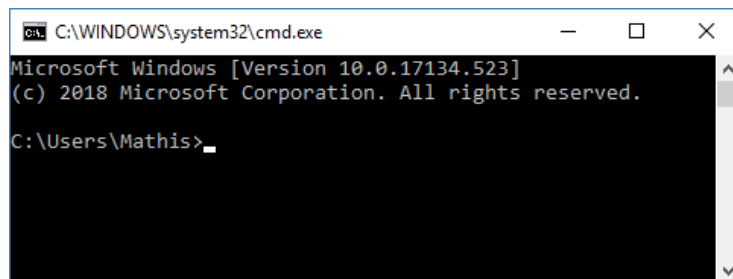


Figure 5: A command prompt.

You will also need to navigate inside the command prompt. The last line in Figure 5 shows that the command prompt is currently in the directory **C:\Users\Mathis**.

But the Scraper is located in **C:\Users\Mathis\Documents\Strava Scraper**. Find out your path by clicking the blank space in the navigation bar in your explorer window, like shown in Figure 6.

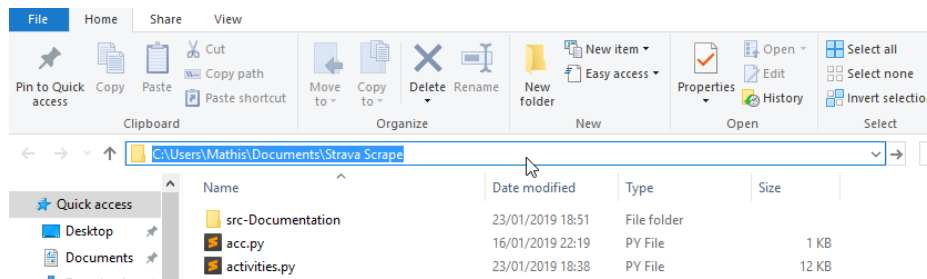


Figure 6: Finding the path to your scraper.

Now use the `cd` (change dir) command to navigate to your python folder by typing `cd C:\Path\To\Your\Folder`. See Figure 7 for how I did it.

If you need to change to another hard drive, because your files are, for example, on D:\, just type D: and press enter and then use `cd` like described above.

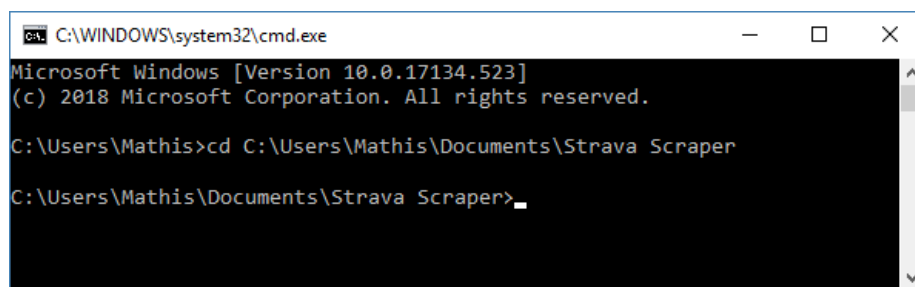


Figure 7: Navigating to a path in cmd.

B Code

B.1 Unaltered main.py

```
1  import os
2  import shutil
3
4  import secret
5  secret.projectPath = 'tutorial'
6
7  import acc
8  import activities
9  import anon
10 import clubs
11 import ego
12 import expo
13 import follows
14 import network
15 import plotting
16 import sampling
17 """
18 This is the main file that imports functions from the other
19 files and runs them
20 """
21
22
23 def removeData():
24     """ remove all the data that was scraped and saved
25     """
26     for path in [x.format(secret.projectPath) for x in
27                 ['./{}/activities/', './{}/clubs/', './{}/commenters/',
28                 './{}/egos/', './{}/followings/', './{}/kudos/',
29                 './{}/rodeWiths/']]:
30         if os.path.exists(path):
31             shutil.rmtree(path)
32     if os.path.isfile('./{}/id.key'.format(secret.projectPath)):
33         os.remove('./{}/id.key'.format(secret.projectPath))
34         anon.loadKey()
35
36
37 def setup():
38     """ Make sure all the folders exist
39     """
40     for path in [x.format(secret.projectPath) for x in
41                 ['./{}/activities/', './{}/clubs/', './{}/commenters/',
42                 './{}/egos/', './{}/followings/', './{}/kudos/',
43                 './{}/rodeWiths/']]:
44         if not os.path.exists(path):
45             os.makedirs(path)
46
47
48 acc.setDelay(500)
49 # removeData()
50 setup()
51
52 # start of own code
53 sampled = sampling.getAthletes(['New York', 'London'], number=1)
54
55 tiesDict = network.egoFollowsNetwork(sampled[0])
56 plotting.showGraph(tiesDict)
57
58 networkAthletes = network.athletesInNetwork(tiesDict)
59
60 expo.saveGraphAsMatrix(tiesDict, 'followNet-{}.dat'.format(sampled[0]))
61 expo.saveEgos(networkAthletes, 'egos.dat', labels=True, type=True)
62
63 # end of own code
64 anon.saveKey()
```

B.2 “Empty” main.py

```
1 import os
2 import shutil
3
4 import secret
5 secret.projectPath = 'empty'
6
7 import acc
8 import activities
9 import anon
10 import clubs
11 import ego
12 import expo
13 import follows
14 import network
15 import plotting
16 import sampling
17 """
18 This is the main file that imports functions from the other
19 files and runs them
20 """
21
22
23 def removeData():
24     """ remove all the data that was scraped and saved
25     """
26     for path in [x.format(secret.projectPath) for x in
27                 ['./{}/activities/', './{}/clubs/', './{}/commenters/',
28                 './{}/egos/', './{}/followings/', './{}/kudos/',
29                 './{}/rodeWiths/']]:
30         if os.path.exists(path):
31             shutil.rmtree(path)
32     if os.path.isfile('./{}/id.key'.format(secret.projectPath)):
33         os.remove('./{}/id.key'.format(secret.projectPath))
34         anon.loadKey()
35
36
37 def setup():
38     """ Make sure all the folders exist
39     """
40     for path in [x.format(secret.projectPath) for x in
41                 ['./{}/activities/', './{}/clubs/', './{}/commenters/',
42                 './{}/egos/', './{}/followings/', './{}/kudos/',
43                 './{}/rodeWiths/']]:
44         if not os.path.exists(path):
45             os.makedirs(path)
46
47
48 acc.setDelay(500)
49 # removeData()
50 setup()
51
52 # start of own code
53 # end of own code
54 anon.saveKey()
```


B.3 Example main.py for Section 2.3

```
1 import os
2 import shutil
3
4 import secret
5 secret.projectPath = 'myFirstCrawl'
6
7 import acc
8 import activities
9 import anon
10 import clubs
11 import ego
12 import expo
13 import follows
14 import network
15 import plotting
16 import sampling
17 """
18 This is the main file that imports functions from the other
19 files and runs them
20 """
21
22
23 def removeData():
24     """ remove all the data that was scraped and saved
25     """
26     for path in [x.format(secret.projectPath) for x in
27                 ['./{}/activities/', './{}/clubs/', './{}/commenters/',
28                 './{}/egos/', './{}/followings/', './{}/kudos/',
29                 './{}/rodeWiths/']]:
30         if os.path.exists(path):
31             shutil.rmtree(path)
32     if os.path.isfile('./{}/id.key'.format(secret.projectPath)):
33         os.remove('./{}/id.key'.format(secret.projectPath))
34         anon.loadKey()
35
36
37 def setup():
38     """ Make sure all the folders exist
39     """
40     for path in [x.format(secret.projectPath) for x in
41                 ['./{}/activities/', './{}/clubs/', './{}/commenters/',
42                 './{}/egos/', './{}/followings/', './{}/kudos/',
43                 './{}/rodeWiths/']]:
44         if not os.path.exists(path):
45             os.makedirs(path)
46
47
48 acc.setDelay(500)
49 removeData()
50 setup()
51
52 # start of own code
53 sample = sampling.getAthletes(['Germany', 'Jim', 'Jane'])
54 print('Number of samples:', len(sample))
55 women = sampling.filterGender(sample, 'F')
56 print('Number of women:', len(women))
57 womenJan = sampling.filterStartDate(women, 1)
58 print('Number of women that started in January:', len(womenJan))
59 expo.saveEgos(sample, 'myExports/samples.dat', gender=True)
60 expo.saveEgos(women, 'myExports/women.dat', gender=True,
61               startDate=True)
62 expo.saveEgos(womenJan, 'myExports/womenJan.dat',
63               sportingTime=[(1,2017),(7,2017),
64                             (1,2018),(7,2018)])
65 athleteId = sample[0]
66 if len(womenJan) > 0:
67     athleteId = womenJan[0]
68 elif len(women) > 0:
69     athleteId = women[0]
70 tiesDict = network.egoKudosNetwork(athleteId, 8, 2017)
71 print('Number of athletes in the Kudos Network of August 2017:',
```

```

72     len(athletesInNetwork(tiesDict)))
73 # end of own code
74 anon.saveKey()

```

B.4 Snippet for importing a network in R

```

myTable <- as.matrix(read.table('network.dat', header= TRUE))
dimnames(myTable)[[2]] <- dimnames(myTable)[[1]]
dimnames(myTable)

```

B.5 tiesDict Format

tiesDict

A python dictionary of ties. The key is the id of the person that initiates the tie. The value is a python list with dictionaries as content. These dictionaries have a field **id** and a field **weight**. The **id** is the the fake athlete id of the person receiving the tie. The **weight** differs depending on the type of tie the **tiesDict** indicates. For Follows it can only be 1, for Kudos, it represents the number of Kudos given.

Example **tiesDict** where fake id 3 is following 1, 2 and 4, while 1 is following 3 and the others follow nobody:

```

1  {"1":
2    [
3      {"id": 3, "weight": 1}
4    ],
5    "2": [],
6    "3": [
7      {"id": 1, "weight": 1},
8      {"id": 2, "weight": 1},
9      {"id": 4, "weight": 1}
10   ],
11   "4": []
12 }

```

C Privacy and Anonymization

C.1 Process

We aim to collect data about athletes on the social sporting network STRAVA and especially about their network relationships. Normally this sort of data is provided by the site via an API. This way, the user has given the app express permission to collect and/or use some specific data. Sadly, STRAVA does not provide API endpoints that deliver the information we would like to collect, like the data on who is following whom.

The information is, however, publicly accessible on the internet. Another tool to collect data from a website is a web scraper. A web scraper simulates the browsing of a website that would normally happen by hand. The advantage is, that the web scraper can navigate the website much faster and automatically.

Because such ‘Robots’ can put a strain on a website by making many more requests in much a shorter time than a human would be able to, there exists a standard protocol called robots.txt to tell robots which sites they may and may not visit. While many companies and robots respect the robots.txt there is no law enforcing it, nor is it a binding contract between user and site owner²¹. In some cases, we are not obeying Strava’s robots.txt and we have indicated that in the appendix where we go over each piece of data we are collecting individually.

Figure 8 indicates the actors and the flow of data that is relevant for this publication. The User transfers their data to Strava: They sign up with some personal information, like name and gender and then starts adding information, like the activities they do and the other athletes they follow or interact with.

Strava internally assigns the user, their activities and every club an id.

The same user (or a different one) can then request that information from Strava and get it back.

Through the same channel as the user our scraper requests information. The only difference is, that it does so automatically.

The scraper, running on a secured system, then anonymizes the data by replacing all ids with fake ones. These are just random numbers that from which it is impossible to guess or infer any information about the original id. The scraper itself keeps an encrypted file that can translate between fake ids and real ones. This is necessary, such that the scraper is able to collect new data and assign it to the correct users it already collected. In addition to anonymization, some data is also aggregated: Instead of exact timestamps for activities, the researcher only receives the month.

This anonymized data is then transferred to the researcher. They can see the real data of and relations between real users without being able to identify the users themselves.

²¹See <http://www.robotstxt.org/faq/legal.html>

The researcher then can do their research with partly aggregated data that has anonymized ids. Before publishing they can choose to further anonymize the data by rounding data points or using categorization.

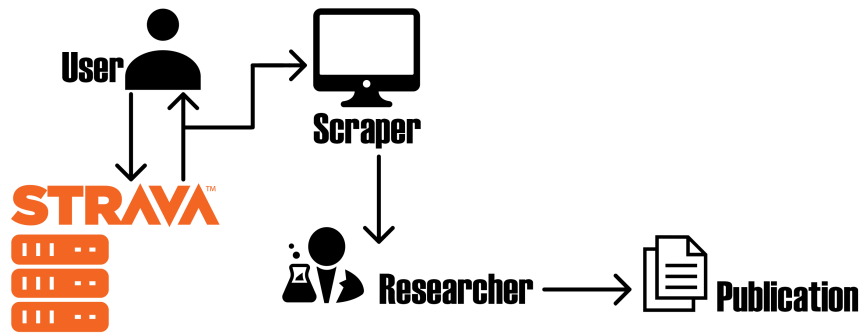


Figure 8: Flow of data between Strava, this tool, the researcher and his publication.

C.2 Collected Data

This is the data that we are collecting. The * symbol represents an id. We start with the information about an athlete.

Data	Where to find it	Protection
Athlete's Gender	Since leader boards can be filtered by gender, the gender can be identified by which leader board the athlete is on /segments/*/leaderboard	This data is accessible to anyone with a STRAVA account
Athlete's Location	Most athletes share city, municipality and country on their profile /athletes/*	This data is publicly accessible
Athlete's Activities	The activities are shown on the athlete's profile, but are actually loaded in via Javascript from a different endpoint /athletes/*/interval_type	This data is accessible to anyone with a STRAVA account
Athlete's preferred type of sport	A summation of the activities is shown on the athlete's profile, but are actually loaded in via Javascript from a different endpoint /athletes/*/profile_sidebar_comparison	This data is accessible to anyone with a STRAVA account
Athlete's Followers	Available on the athlete's profile /athletes/*/follows	This data is accessible to anyone with a STRAVA account, but it is on a URL that is inside the robots.txt
Other athletes that the Athlete follows	Available on the athlete's profile /athletes/*/follows	This data is accessible to anyone with a STRAVA account, but it is on a URL that is inside the robots.txt

Next, information about an activity.

Data	Where to find it	Protection
Activity Type (run, swim, ride)	That information can be found while collecting an Athlete's activities (see table above) /athletes/*/interval_type	This data is accessible to anyone with a STRAVA account
Activity Distance	Sometimes this information is available while collecting an Athlete's activities (see above) If it is not, we are not collecting it /athletes/*/interval_type	This data is accessible to anyone with a STRAVA account
Activity Time	same as above (Activity Distance) /athletes/*/interval_type	This data is accessible to anyone with a STRAVA account
Activity Pace	same as above (Activity Distance) /athletes/*/interval_type	This data is accessible to anyone with a STRAVA account
Activity Elevation Gain	same as above (Activity Distance) /athletes/*/interval_type	This data is accessible to anyone with a STRAVA account
Activity Average Power	same as above (Activity Distance) /athletes/*/interval_type	This data is accessible to anyone with a STRAVA account
Other athletes that gave kudos for the Activity	Loaded from some API-like endpoint /feed/activity/*/kudos	This data is accessible to anyone with a STRAVA account
Other athletes that commented on the Activity	Loaded from some API-like endpoint /feed/activity/*/comments	This data is accessible to anyone with a STRAVA account
Other athletes that partook in the Activity	Loaded from some API-like endpoint /feed/activity/*/group_athletes	This data is accessible to anyone with a STRAVA account
Month that the Activity took place in	Due to the way activities are collected (see table above "Athlete's Activities") we know the month (and year) the activity took place in	This data is accessible to anyone with a STRAVA account

Lastly, club-related information:

Data	Where to find it	Protection
Members of a club	Available on a club's page /clubs/*/members	This data is accessible to anyone with a STRAVA account, but it is on a URL that is inside the robots.txt

To prevent confusion and misunderstanding, here is what we are expressly NOT collecting, even though it would be accessible:

Data	Where to find it	Protection
Exact timestamp of the Activity	Available on the activity's page /activities/*	This data is publicly accessible
Content or timestamp of comments on an Activity	Loaded from some API-like endpoint /feed/activity/*/comments	This data is accessible to anyone with a STRAVA account
Health-related data, like heartrate or cadence	Available on an activity's page, probably loaded from somewhere else /activities/*	This data is accessible to anyone with a STRAVA account
Exact locations of activities	Some activities have a map (with GPS coordinates) of where they happened /activities/*	This data is publicly accessible
Athlete's names	Available on the athlete's profile /athletes/*	This data is publicly accessible
Any images or text	e.g. profile pictures or pictures uploaded to an activity or profile text	Depends on data