

You have 1 free member-only story left this month. [Upgrade for unlimited access.](#)

# How To Automate a Club Challenge with Strava and Google Sheets, for Dummies



Maggie L

Follow

Sep 2, 2020 · 12 min read ★

When my friend suggested a year long mileage challenge for our cycling club I thought, why not automate the data collection? I have some programming experience, I'm a spreadsheet whiz, this shouldn't be too hard. Many Google searches turned up other people asking how to do exactly this but no answers.

I want to credit [@benjikj](#)'s article '[Using Python to Connect to Strava's API and Analyze Your Activities — Dummies Guide](#)' which was exactly what I needed to get the basics of authentication and activity collection so I could create my program. I used the script he provides in that article as a starting point for my code.

This article is intended to guide you through the creation of a python script, using the Strava and Google Sheets APIs, to collect activity data from your club members and write it to a spreadsheet. I was not very familiar with most of this technology before creating this solution but some familiarity with how to modify code is necessary if you want to adapt this to your own club.

The full script is available [here](#). This article is intended to walk you through the authentication process and the functions in my script. This is my first time writing this type of program and I know it is not the most efficient. I hope that it is simple enough and beginner friendly enough that a Strava user with some basic programming skills will be able to use it to run their own club challenge.

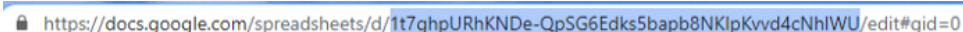
## Setup

Before we get to the code, you will need to create a Google Sheet and register a Strava app.

### Create Google Sheet

Log into your Google account and create a new spreadsheet. Once you are in the new, blank spreadsheet, make a note of the id in the URL. This id will be referred to as the

`google_sheet_id`.

 <https://docs.google.com/spreadsheets/d/1t7ghpURhKNDe-QpSG6Edks5bapb8NKlpKvvd4cNhlWU/edit#gid=0>

Sheet id is highlighted in the URL

If you already know how you want your data setup in the spreadsheet I recommend creating the spreadsheet and leaving the cells that need input blank. This process will help you think about how you need to structure your data.

For my club challenge we were offered 3 options: (1) ride 52 miles a week, (2) ride one 52 mile ride each week, or (3) climb 5200 feet during one ride each week. The challenge would be 13 weeks long and we had some rules related to how many miles/rides you could carry over from week to week.

Name	Strava ID	Mileage Plan	2020-08-31		2020-09-07	
			This Week	Previous Balance	This Week	Previous Balance
Juan	123	1	23	0	0	-29
Sarah	234	1	0	0	0	-52
Alice	345	1	15	0	0	-37
Chris	456	2	1	0	0	0
Tom	567	1	3	0	0	-49
Amal	678	2	0	0	0	-1
AJ	789	1	12	0	0	-40
Dominic	890	2	1	0	0	0
Kiara	901	1	44	0	0	-8

My club's challenge spreadsheet

You will enter all the members of your club who are participating in the challenge, along with their Strava ID. To find their ID, go to their profile page, then look at the URL. It will look like [https://www.strava.com/athletes/\[STRAVA\\_ID\]](https://www.strava.com/athletes/[STRAVA_ID]) .

For this challenge, the logic related to how to carry miles/rides over week to week is coded into the spreadsheet, in the *Previous Balance* columns. I need the python script to get each user's rides for the week and write them to the appropriate *This Week* column.

## Register Strava App

I registered my API account with Strava seven years ago and it looks a lot different now than it did then. Since I didn't have to register an account for this project I found [a guide in Johnson's article](#) which you should check out (see the Create your App/API Connection section).

Once your account is registered, the information that you will need can be found by going to <https://www.strava.com/settings/api> . Here you will find your `client_id` and `client_secret` .

## Authentication

This is the section I had the most trouble with. I had never written a script that required authentication before. Thankfully [Johnson's article provided an easy way to authenticate in the Authentication section](#).

There are different levels of access that your application can have. I recommend using the least permissive controls you can get away with. For a club activity data pull you will need `profile:read_all` and either `activity:read_all` or `activity:read` . If you want to collect all activities, even those set to 'Only You' visibility, choose `activity:read_all` . If you are interested in just publicly visible activities choose `activity:read` .

Once you choose which level of access you need, log into your Strava account, then go to the following URL (after replacing the `client_id` with your application's `client_id`. If you

only need activity:read access, delete the `_all` at the end of the URL.

```
http://www.strava.com/oauth/authorize?client_id=
[CLIENT_ID]&response_type=code&redirect_uri=http://localhost/exchange_token&approval
_prompt=force&scope=profile:read_all,activity:read_all
```



## Authorize 52 52s parser to connect to Strava

Collect data for 52 52s challenge

<http://hiccupingyogi.com>

### 52 52s parser will be able to:

- ☒ View data about your public profile (required)
- ☒ View your complete Strava profile
- ☒ View data about your private activities

Authorize

Cancel

To revoke access to an application, please visit your [settings](#) at any time.

By authorizing an application you continue to operate under our [Terms of Service](#).

You should see something similar to this

After you click the orange 'Authorize' button you will be taken to an error page, or a page that simply fails to load. Copy the URL of this page. Note the `code` that is in the parameters string.

```
http://localhost/exchange_token?state=&code=
.[THIS_IS_THE_CODE_YOU_NEED_TO_COPY]&scope=read,activity:read_all,profile:read_all
```

Now, using the script I pulled from Johnson's article, insert the `client_id`, `client_secret`, and `code`. To make the output more readable, since we are interested in pulling data for multiple users, insert your `strava_id` on the output files. **This needs to be done within 15 minutes of the user requesting the token** so if you're doing this for a club it might be easier to schedule a time for everyone to try authenticating.

```
import requests
import json

# Make Strava auth API call with your
# client_code, client_secret and code
response = requests.post(
    url = 'https://www.strava.com/oauth/token',
```

```

data = {
    'client_id': [INSERT_CLIENT_ID_HERE],
    'client_secret':
'[[INSERT_CLIENT_SECRET_KEY]',
    'code': '[INSERT_CODE_FROM_URL_HERE]',
    'grant_type': 'authorization_code'
}

)

#Save json response as a variable
strava_tokens = response.json()

# Save tokens to file
with open('strava_tokens_[STRAVA_ID].json', 'w') as outfile:
    json.dump(strava_tokens, outfile)

# Open JSON file and print the file contents
# to check it's worked properly
with open('strava_tokens_[STRAVA_ID].json') as check:
    data = json.load(check)
print(data)

```

When you run this, you should get an output that contains identifying information for the user.

DO NOT RUN THIS SCRIPT AGAIN FOR A USER WHO IS ALREADY AUTHENTICATED. I learned that if you run this again you effectively force the token to expire and you will have to do this process again. Asking the user to reauthenticate is an easy solution, but it gets annoying if you are asking members of your club to go through this process multiple times. If this happens, you will get an error message like this.

```

{"message": "Bad Request", "errors": [{"resource": "AuthorizationCode", "field": "", "code": "expired"}]}

```

Error message when token is expired

## Authenticating Other Users

This is one of the last steps to take but it is directly related to the above process so I'm going to include it here. When your program is tested and ready to go, you will need each user participating in your club's challenge to authenticate. I'm sure there are easier ways to do this, but since I wanted to do this quickly, I sent an email to my club mates with the following request.

*In order to automate the data collection I need everyone to authenticate with my developer account. I need read access to your data. I didn't want to spend a ton of time getting fancy so the authentication involves two steps from you. It's all read only and if anyone is worried about it I can remove the permissions once this is over.*

*1. Go to the following URL, make sure that it is your Strava account that is logged in, then click the orange Authorize button*

*[https://www.strava.com/oauth/authorize?client\\_id=\[CLIENT\\_ID\]&response\\_type=code&redirect\\_uri=http://localhost/exchange\\_token&approval\\_prompt=force&scope=profile:read\\_all,activity:read\\_all](https://www.strava.com/oauth/authorize?client_id=[CLIENT_ID]&response_type=code&redirect_uri=http://localhost/exchange_token&approval_prompt=force&scope=profile:read_all,activity:read_all)*

*2. Once you click 'Authorize' you will be taken to a new page which will either be blank or display some type of error message. Please copy the URL of this page and send it to me. This contains all the information I need to get you added to the automated data pull.*

*Once this is done you won't need to worry about adding data on a weekly basis. We'll spot check it to make sure it's collecting properly. This script will pull activities from the previous several weeks so there is no need to worry if you upload something a week after the fact.*

You will run the authorization script once for each user, swapping in their unique `code` and their `strava_id`.

## Get User Activity Data From Strava API

I used the script provided in Johnson's article to get my data just to see what I would be working with. We will be pulling data for multiple users and then writing that data into a spreadsheet so this is where my project diverges from Johnson's. I'm writing this guide for people with some technical background but if you don't need to know the details feel free to skip ahead.

### Data Structures (skip if don't need to know the details)

We are going to iterate over a list of users, over the course of several weeks, so I decided to use a dictionary of dictionaries of dictionaries, keyed on week and `strava_id`.

First I built a dictionary of user information. In my case I had the user `name`, `strava_id`, and `plan` (so I knew which challenge option the user was participating in).

```
def get_user_info(sht):

    # iterate over tracking numbers
    # first cell with data is E3
    n = 4
    name_cell = 'A' + str(n)
    strava_id_cell = 'B' + str(n)
    plan_cell = 'C' + str(n)

    name = sht.acell(name_cell).value
    strava_id = int(sht.acell(strava_id_cell).value)
    plan = int(sht.acell(plan_cell).value)

    users = {}

    while name != '':
        strava_id = int(sht.acell(strava_id_cell).value)
        plan = int(sht.acell(plan_cell).value)
        user = {'name' : name, 'plan' : plan, 'value' : 0, 'row_nbr'
: n}
        users[strava_id] = user

    n = n+1
    name_cell = 'A' + str(n)
    strava_id_cell = 'B' + str(n)
    plan_cell = 'C' + str(n)
    name = sht.acell(name_cell).value

    return users
```

Next, I built a dictionary of dictionaries keyed on the weeks that the challenge would cover. My spreadsheet uses two columns per week (one for the week's mileage and another that adds the previous week's balance to the current week's mileage), so I needed to skip forward two columns to get the next week's date and column letter.

```

def create_weekly_dictionary(start_date, nbr_of_weeks):

    # initialize dictionary
    weekly_totals = {}

    # set weeks
    weekly_date = start_date
    first_date_col_letter = ''
    second_date_col_letter = 'D' # initial date column
    date_col = str(first_date_col_letter) +
    str(second_date_col_letter)

    i = 0
    letters_per_column_name = 1
    while i < nbr_of_weeks:
        weekly_totals[weekly_date] = {'date_col': date_col}
        weekly_date = weekly_date + timedelta(days=7)

    # update second column letter
    if (second_date_col_letter == 'Y' or second_date_col_letter
    == 'Z'):
        second_date_col_letter =
        chr(ord(second_date_col_letter)-24)
        if first_date_col_letter == '':
            first_date_col_letter = 'A'
        else:
            first_date_col_letter =
            chr(ord(first_date_col_letter)-24)
        else:
            second_date_col_letter =
            chr(ord(second_date_col_letter)+2)

    date_col = str(first_date_col_letter) + str(second_date_col_letter)
    i = i+1

    return weekly_totals

```

Finally I create a dictionary of users for each week. This involves iterating over each dictionary keyed by week and adding the user keyed dictionary. It is important that you make a deep copy of this user dictionary, otherwise each week will reference the same user dictionary so any update made in one week will be seen by all weeks. This part is where I was most thankful for my CS education because I don't know how I would have caught this mistake otherwise.

```

def create_weekly_user_dict(weekly_dict, user_dict):

    wk_dict = weekly_dict
    for week in wk_dict:
        for user in user_dict:
            wk_dict[week][user] = copy.deepcopy(user_dict[user])
    return wk_dict

```

## Get Activity Data From Strava

Now that we have a dictionary of users and their ids we can iterate over them to pull their data. In order for this to work I assume that the user has already authenticated, that their token is saved in a file named `strava_tokens_[STRAVA_ID].json`, and that these token files are located in the same directory as your script.

The request needs the access token. The other arguments are not required. The number of results per page, `per_page`, defaults to 30. The maximum number of results per page

is 200, which is what I have set here. The default `page` is 1. Before and after accept epoch timestamps. If either of these parameters is used, the results will be limited to those activities that were created after the timestamp or before the timestamp.

```
r = requests.get(url + '?access_token=' + access_token +
'&per_page=200' + '&page=' + str(page) + '&after=' +
str(start_date_epoch) + '&before=' + str(end_date_epoch))
```

The following function loops over each user. It authenticates the user, requests the user's activities from Strava, stores the results in a data frame, then stores that data frame in a dictionary, keyed on the user's Strava ID.

```
def get_user_activities_from_strava(user_dict, start_date_epoch,
end_date_epoch):

    # create dictionary of user activities
    activities = {}

    # loop over users
    for user in user_dict:

        ## Get the tokens from file to connect to Strava
        with open('strava_tokens_' + str(user) + '.json') as
json_file:
            strava_tokens = json.load(json_file)
            ## If access_token has expired then use the refresh_token to
get the new access_token
            if strava_tokens['expires_at'] < time.time():
                #Make Strava auth API call with current refresh token
                response = requests.post(
                    url =
'https://www.strava.com/oauth/token',
                    data = {
                        'client_id': [CLIENT_ID],
                        'client_secret':
'[CLIENT_SECRET]',
                        'grant_type':
'refresh_token',
                        'refresh_token':
strava_tokens['refresh_token']
                    }
                )
                #Save response as json in new variable
                new_strava_tokens = response.json()
                # Save new tokens to file
                with open('strava_tokens.json', 'w') as outfile:
                    json.dump(new_strava_tokens, outfile)
                #Use new Strava tokens from now
                strava_tokens = new_strava_tokens

        #Loop through all activities
        page = 1
        url = "https://www.strava.com/api/v3/activities"
        access_token = strava_tokens['access_token']
        ## Create the dataframe ready for the API call to store your
activity data
        user_activities = pd.DataFrame(
            columns = [
                "id",
                "name",
                "start_date_local",
                "type",
                "distance",
                "moving_time",
                "elapsed_time",
                "total_elevation_gain"
```

```

    ]
)

while page < 2: # this is to limit the results coming back just in
case a user has hundreds of activities in that window

# get page of activities from Strava
r = requests.get(url + '?access_token=' + access_token +
'&per_page=200' + '&page=' + str(page) + '&after=' +
str(start_date_epoch) + '&before=' + str(end_date_epoch))
r = r.json()

# if no results then exit loop
if not r:
    break

# otherwise add new data to dataframe
for x in range(len(r)):
    user_activities.loc[x + (page-1)*200, 'id'] = r[x]
['id']
    user_activities.loc[x + (page-1)*200, 'name'] = r[x]
['name']
    user_activities.loc[x + (page-
1)*200, 'start_date_local'] = r[x]['start_date_local']
    user_activities.loc[x + (page-1)*200, 'type'] = r[x]
['type']
    user_activities.loc[x + (page-1)*200, 'distance'] =
r[x]['distance']
    user_activities.loc[x + (page-1)*200, 'moving_time'] =
r[x]['moving_time']
    user_activities.loc[x + (page-1)*200, 'elapsed_time']
= r[x]['elapsed_time']
    user_activities.loc[x + (page-
1)*200, 'total_elevation_gain'] = r[x]['total_elevation_gain']

# increment page
page += 1

# add user_activities df to activities dictionary
activities[user] = user_activities

return activities

```

## Parse Activity Data

This method is where the activity data is processed and the weekly user data is updated. This method will reflect the rules of your challenge. For my challenge, the user's value will either be the number of miles they rode during the week ( if plan == 1 ) or the number of rides they completed that are at least 52 miles ( if plan == 2 ) or at least 5200 feet in elevation ( if plan == 3 ).

```

def parse_activity_data(weekly_user_dict, user_dict, activities):

# loop over user ids
for user in user_dict:
    strava_id = user

mileage_total = 0
rides_over_fifty_two_miles = 0
rides_over_fifty_two_hundred_feet = 0

activity = activities[user]

for index, row in activity.iterrows():

    if row['type'] == 'Ride':
        activity_date =
datetime.strptime(row['start_date_local'][:10], '%Y-%m-%d')

```



```

        week_of =
pendulum.instance(activity_date).start_of('week').date()
        activity_mileage = int(float(row['distance'])) # in
meters
        activity_elevation =
int(float(row['total_elevation_gain'])) # in meters

# if activity is from outside the challenge scope, break
        if week_of not in weekly_user_dict:
            break

plan = weekly_user_dict[week_of][strava_id]['plan']

        # get value that will be written to sheet
        if plan == 1: # 52 miles over the course of the week
            weekly_user_dict[week_of][strava_id]['value'] +=
activity_mileage*0.000621371192 # convert to miles
            if plan == 2 and activity_mileage >= 83685: # one 52
mile ride during the week (83685 meters)
                weekly_user_dict[week_of][strava_id]['value'] +=
1
            if plan == 3 and activity_elevation >= 1584: # one
5200 ft ride during the week (1584 meters)
                weekly_user_dict[week_of][strava_id]['value'] +=
1
1

```

## Update the Google Sheet

Now that the data is parsed and our data structure has the users and their values split by week we have everything we need to update the Google Sheet. We will write to cells by referring to them in A1 notation.

We stored the column names (A, B, C, etc) in the weekly dictionary. We stored the user row in the user dictionary. We iterate over each week and write the value in the user's row.

```

def write_to_sheet(sht, weekly_user_dict, user_ct):

    # iterate over weeks
    for week in weekly_user_dict:

        date_col = weekly_user_dict[week]['date_col']

        # iterate over users
        for user in weekly_user_dict[week]:
            if user != 'date_col':

                plan = weekly_user_dict[week][user]['plan']

                # get user's row number
                row_nbr = weekly_user_dict[week][user]['row_nbr']

                # write value to week/user cell
                update_cell = str(date_col) + str(row_nbr)
                value = weekly_user_dict[week][user]['value']

                sht.update(update_cell, value)

```

## How To Run Your Challenge

1. Set up your spreadsheet
2. Ask your users to authenticate
3. Run the script
4. Get motivated to beat your club mates!

You can find the full script [here](#).

I hope that this makes the logistics of running a club challenge easier. Unfortunately this won't help you log extra miles!

If you have any questions or feedback please let me know. This article is intended to be a guide; you will still need to make some changes to get it to work for your club.

Find me on [Strava](#) or [LinkedIn](#)

## Python In Plain English

Did you know that we have three publications and a YouTube channel? Find links to everything at [plainenglish.io](https://plainenglish.io)!

[Strava](#)   [Python](#)   [Automation](#)

[About](#) [Help](#) [Legal](#)

Get the Medium app

