

Saros Fit Strava App Implemented in AWS

By Sheraz Choudhary

Full Saros Fit (Strava App) Implemented in AWS YouTube Video (Long [17:22]):

- https://www.youtube.com/watch?v=hJLA_NPaJw

This project will live in GitHub at:

- <https://github.com/sherazch00/strava-app>

Getting all the data into AWS and providing the ability to get new Strava activity data is the main goal of this project.

I extensively use Strava to record all my workouts. Unfortunately, Strava website does not allow me to run all the analysis I would like to on my activities or download all my activities easily. Fortunately, Strava does provide an API to access all Strava data.

The initial step of this project was to create a Strava App that can access all of a user's data using the Strava API. The application first downloads an overview of all the Strava activities. This activity overview data provides a lot of the basic information needed for each activity. To get rich data for every second of a workout the application then downloads the detailed activity streams for each new activity in the activity overview data. This part of the project required learning to use and authenticate with the Strava using the REST API.

The python script was configured to run on AWS EC2 instance(s) and to use S3 to store the two csv files (activities_overview and activities_details). Strava limits the number of requests a Strava App can make via the API so each call will have to be made efficiently with wait timers.

Initially, I followed the usual steps and created an EC2 instance, setup security groups, SSH'd, created a python environment and used a text-based editor such as nano. This approach worked but was cumbersome and not very useful when debugging. The Cloud9 IDE really simplified the process while provided a lot more tools that were extremely helpful when coding and debugging the python scripts!

Future steps:

- AWS RDS to query the activities overview dataframe using SQL.
- AWS EMR to analyze the activities details dataframe (*1.8 million rows only for my personal Strava data*).
- AWS SageMaker/ML to find patterns in the activities details dataframe
- AWS Lambda for creating calculated fields from the data
- AWS SNS for notifications based on rules
- Amazon Alexa for daily summaries and refreshing data
- Host application on AWS Cloud
- Integrate with Golden Cheetah (open-source fitness analysis app)

STEP 1: CREATING STRAVA APP AND LEARNING STRAVA API

<https://developers.strava.com/>

<https://towardsdatascience.com/using-the-strava-api-and-pandas-to-explore-your-activity-data-d94901d9bfde>

Follow the steps in the 'Getting Started Guide' to create a Strava App. The first tricky part was to get the temporary refresh token that had read all access to Strava data.

Once that step is accomplished the next challenge is to craft calls to the API that provide the data desired from Strava.

Create Strava API Application:

The screenshot shows the Strava API Application creation interface. On the left is a sidebar with links: My Profile, My Account, My Performance, Display Preferences, Privacy Controls, Data Permissions, Email Notifications, My Gear, My Apps, Partner Integrations, My Badges, and **My API Application** (which is highlighted with a red box). The main area has a title 'My API Application' and a sub-section 'Create an App'. It includes fields for 'Application Name' (Saros Fit), 'Category' (Performance Analysis), 'Club' (a dropdown menu), 'Website' (https://www.sarosai.com), 'Application Description' (Analysis on Strava data to predict fitness.), 'Authorization Callback Domain' (local host), and a checkbox for 'I've read and agree with Strava's API Agreement'. A large orange 'Create' button with a hand cursor icon is at the bottom.

Details about Strava API Application:

The screenshot shows the Strava API Application details page. On the left is a sidebar with links: My Profile, My Account, My Performance, Display Preferences, Privacy Controls, Data Permissions, Email Notifications, My Gear, My Apps, Partner Integrations, My Badges, and My API Application (which is highlighted with a red box). The main content area has a title "My API Application". It shows the following information:

- Category:** Performance Analysis
- Club:**
- Client ID:** 63 (with a cursor arrow pointing to it)
- Client Secret:** ***** [show](#)
- Your Access Token (?)**: scope: read, expires at 2021-04-02T09:57:30+00:00
- Your Refresh Token (?)**: scope: read
- Rate Limits (?)**: 100 requests every 15 minutes, 1000 daily

At the bottom, there is a link [Daily requests \(?\)](#).

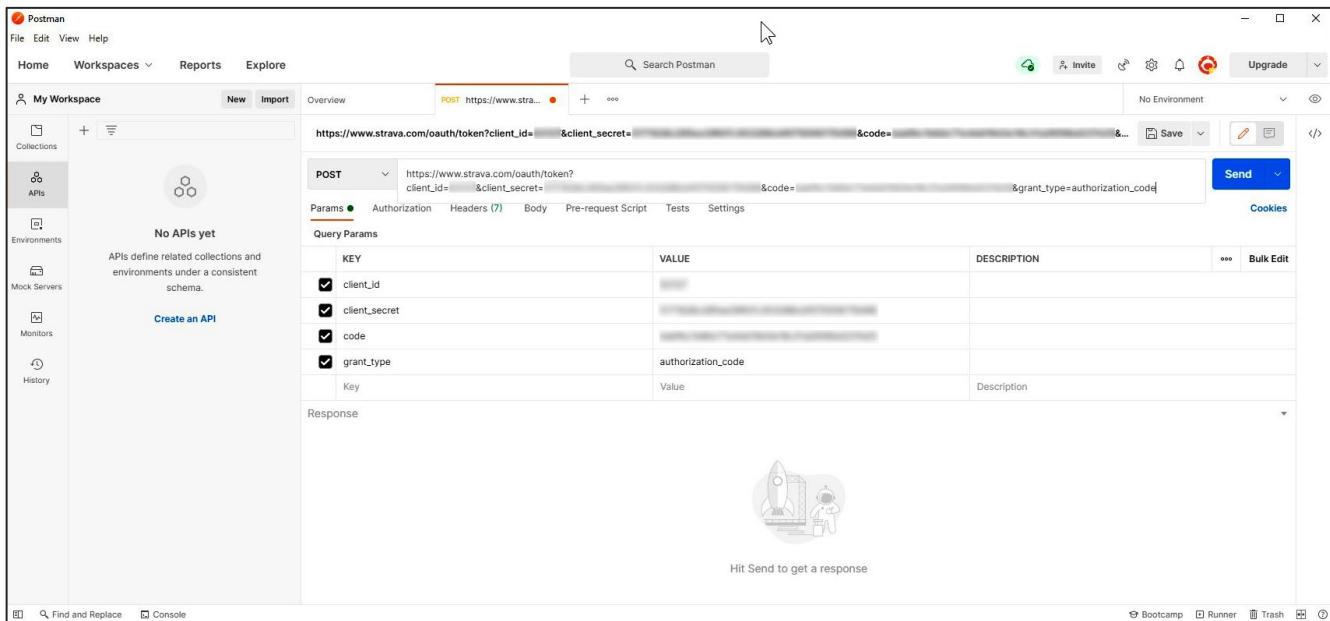
Access Application from Strava User and Authorize It (*detailed instructions in guide*):

The screenshot shows the Strava authorization consent screen. It features the Saros Fit logo and the text "Authorize Saros Fit to connect to Strava". Below that, it says "Analysis on Strava data to predict fitness." and provides the URL <https://www.sarosai.com>. A section titled "Saros Fit will be able to:" lists two permissions with checkboxes:

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

 There are two buttons at the bottom: "Authorize" (highlighted with a cursor arrow) and "Cancel". At the bottom of the screen, there is a note: "To revoke access to an application, please visit your [settings](#) at any time." and "By authorizing an application you continue to operate under our [Terms of Service](#)".

Use Strava API documentation to get the syntax for the calls to get the temporary ‘Refresh Token’ which is needed to access Strava data:



The Strava API documentation was especially hard to figure out when trying to get activity streams from Strava. It took a long time to figure out the exact call to make to successfully get an activity stream.

Initially, I was getting streams one at a time for each of the 11 streams individually for each activity. So, I could only download around 10 activities before I hit the 100 requests per 15 minutes limit! There was no example I found that tried to get multiple streams in one request but I tried it and it worked!!!

JSON Request:

```
requests.get('https://www.strava.com/api/v3/activities/<activity_id>/streams?access_token=<access_token>&keys=time,distance,latlng,altitude,velocity_smooth,heartrate,cadence,watts,temp,moving,grade_smooth&key_by_type=true')
```

Python Code:

```
def activity_streams(id):
    a_dict = {}
    a_df = pd.DataFrame()
    a_url = "https://www.strava.com/api/v3/activities/"

    streams_list = ['time','distance','latlng','altitude','velocity_smooth','heartrate','cadence','watts','temp',
                    'moving','grade_smooth']
    streams_text = 'time,distance,latlng,altitude,velocity_smooth,heartrate,cadence,watts,temp,moving,grade_smooth'

    a_json = pd.json_normalize(requests.get(a_url + str(id) + '/streams?access_token=' + access_token +
                                            '&keys=time,distance,latlng,altitude,velocity_smooth,heartrate,cadence,watts,temp,moving,grade_smooth' +
                                            '&key_by_type=true').json())

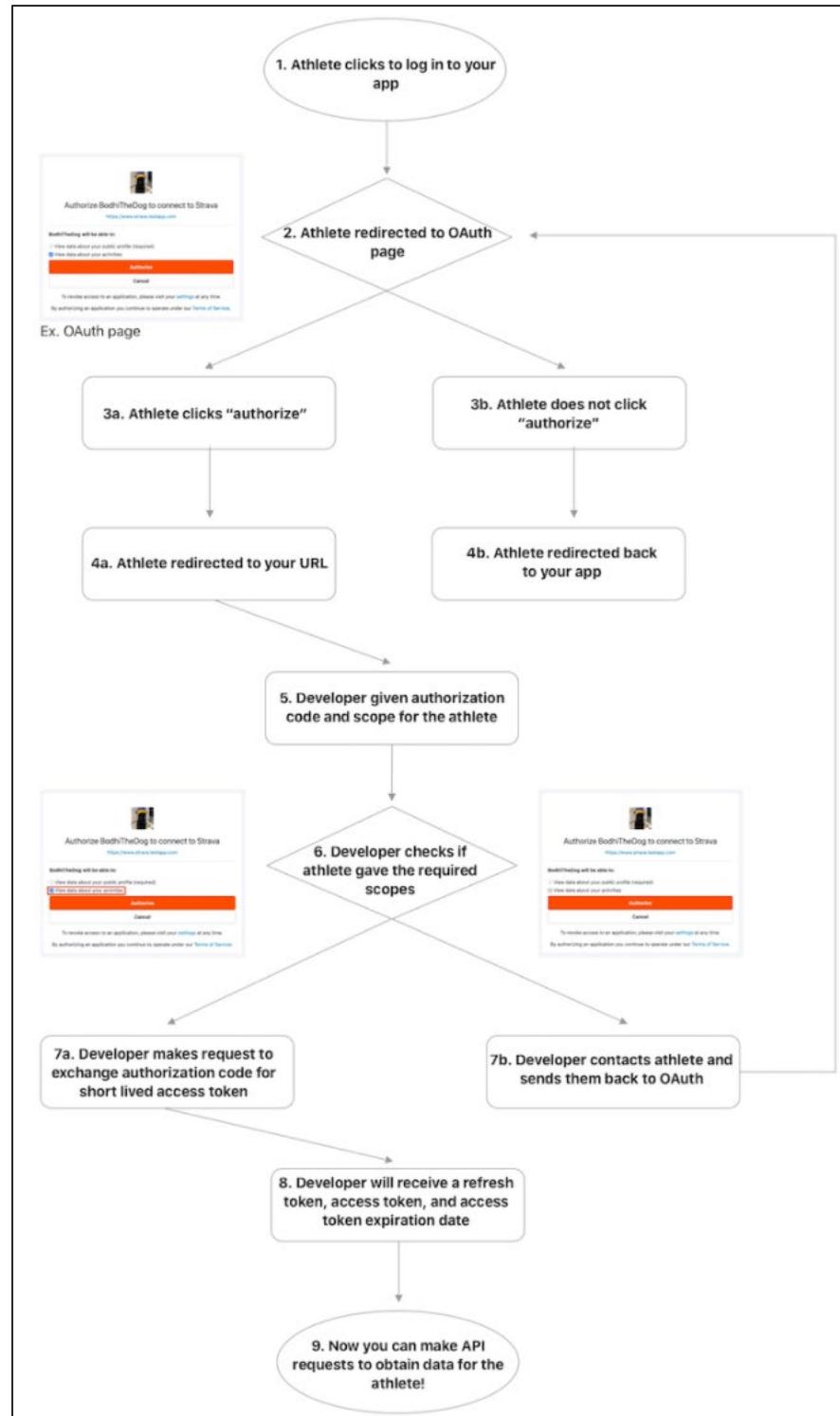
    for a in range(0,len(streams_list)):
        try:
            a_df[streams_list[a]] = a_json[str(streams_list[a]) + '.data'][0]
        except:
            a_df[streams_list[a]] = np.nan

    a_df['id'] = id
    idx = activities_overview.index[activities_overview['id'] == id].tolist()[0]

    a_df['date'] = activities_overview['start_date_local'][idx]
    a_df['name'] = activities_overview['name'][idx]
    a_df['type'] = activities_overview['type'][idx]

    return a_df
```

Strava Flow Chart of How to Authorize a Strava App to Get User Data:



STEP 2: WRITING PYTHON SCRIPTS TO DOWNLOAD STRAVA DATA

The python script was initially written locally in Jupyter Notebooks. The script was then migrated to AWS with the following changes:

- boto3 was imported to read and write from S3 bucket
 - AWS Secret Manager function was added
 - AWS Secret Manager was used to get the tokens rather than a local environmental file
 - All files are written to S3 in addition to the local folder
 - Code was added to resolve <NA> error when calling to_pickle

Full Saros Fit Strava App in AWS Python Script (saros-fit.py)

Yellow-highlighted sections are specific to AWS and will fail on a local machine

```

# # Download Activity Summary and Details from Strava
#
# Strava provides basic information on how to get access to the API at https://developers.strava.com/docs/getting-started/.
# The information from the Strava API page combined with
# https://towardsdatascience.com/using-the-strava-api-and-pandas-to-explore-your-activity-data-d94901d9bfde and other websites
# allowed me to write the code necessary to connect to Strava.
#
# With some additional research I was able to download all my activities and their summary data. Though, in all my online
# searching I could not find any good examples of how to download all the detailed activity streams associated with each
# Strava activity. The code to download activity details was mostly written from scratch.
#
# **The value of this notebook is:**
#   - example of code to authenticate and connect to the Strava API (follow Strava instructions to create an app)
#   - download all activities and their associated summaries to a dataframe
#   - download all 11 detailed activity streams for each activity to a dataframe
#   - single notebook that authenticates to Strava, downloads activity summaries and downloads activity details
#
# **Additional useful features:**
#   - controls the number of requests so they don't exceed Strava's default limits (100 requests/15 min)
#   - stores all download activity details in pkl and csv format
#   - loads previously downloaded activity details and then only downloads details for new activities
#
# ##### NEXT STEPS:
#   - Create a separate notebook that loads the pkl file with activity details
#     - run basic analysis and create visualizations on the activity details
#     - run machine learning models on the activity details to find patterns in heartrate and wattage numbers
#   ---
#
# **Written by: Sheraz Choudhary**
# **Date: November 2021**
# ---

# import libraries
import numpy as np
import pandas as pd

import os
import sys
import subprocess
import pathlib
import requests
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

import codecs
from codecs import open
from datetime import date
import time

import boto3
import base64
from botocore.exceptions import ClientError

cli = boto3.client('s3')

# AWS Secret Keeper Function
def get_secret(secret_name, region_name = "us-east-1"):
    # Create a Secrets Manager client
    session = boto3.session.Session()
    client = session.client(
        service_name='secretsmanager',
        region_name=region_name
    )

```

```

# In this sample we only handle the specific exceptions for the 'GetSecretValue' API.
# See https://docs.aws.amazon.com/secretsmanager/latest/apireference/API_GetSecretValue.html
# We rethrow the exception by default.

try:
    get_secret_value_response = client.get_secret_value(SecretId=secret_name)
except ClientError as e:
    if e.response['Error']['Code'] == 'DecryptionFailureException':
        # Secrets Manager can't decrypt the protected secret text using the provided KMS key.
        # Deal with the exception here, and/or rethrow at your discretion.
        raise e
    elif e.response['Error']['Code'] == 'InternalServiceErrorException':
        # An error occurred on the server side.
        # Deal with the exception here, and/or rethrow at your discretion.
        raise e
    elif e.response['Error']['Code'] == 'InvalidParameterException':
        # You provided an invalid value for a parameter.
        # Deal with the exception here, and/or rethrow at your discretion.
        raise e
    elif e.response['Error']['Code'] == 'InvalidRequestException':
        # You provided a parameter value that is not valid for the current state of the resource.
        # Deal with the exception here, and/or rethrow at your discretion.
        raise e
    elif e.response['Error']['Code'] == 'ResourceNotFoundException':
        # We can't find the resource that you asked for.
        # Deal with the exception here, and/or rethrow at your discretion.
        raise e
else:
    # Decrypts secret using the associated KMS CMK.
    # Depending on whether the secret is a string or binary, one of these fields will be populated.
    if 'SecretString' in get_secret_value_response:
        secret = get_secret_value_response['SecretString']
    else:
        secret = base64.b64decode(get_secret_value_response['SecretBinary'])

return secret

# ## Connect to Strava -- Get Current Access Token
# (https://towardsdatascience.com/how-i-manage-credentials-in-python-using-aws-secrets-manager-1bd1bf5da598)
client_id = get_secret("<arn for client_id>")
client_secret = get_secret("<arn for client_secret>")
refresh_token = get_secret("<arn for refresh_token>")

# (https://github.com/franchyze923/Code_From_Tutorials/blob/master/Strava_Api/strava_api.py)
auth_url = "https://www.strava.com/oauth/token"

payload = {
    'client_id': client_id,
    'client_secret': client_secret,
    'refresh_token': refresh_token,
    'grant_type': "refresh_token",
    'f': 'json'
}

print("\nRequesting Token...")
res = requests.post(auth_url, data=payload, verify=False)
access_token = res.json()['access_token']
print("Access Token Received!")
print("")

# ## Create Dataframe with Summary Info for All Activities *(Strava API)*
# (http://www.hainke.ca/index.php/2018/08/23/using-the-strava-api-to-retrieve-activity-data/)
# Initialize the dataframe
activities_overview = pd.DataFrame()

url = "https://www.strava.com/api/v3/athlete/activities"
page = 1

while True:
    # get page of activities from Strava
    page_json = requests.get(url + '?access_token=' + access_token + '&per_page=200' + '&page=' + str(page)).json()

    for a in range(len(page_json)):
        # (https://stackoverflow.com/questions/21104592/)
        activity_json = pd.json_normalize(page_json[a])
        activities_overview = activities_overview.append(activity_json, ignore_index=True)

    # if no results then exit loop
    if (not page_json):
        break

    # increment page
    page += 1

# makes sense since new added on bottom
activities_overview = activities_overview.sort_values(by='id', ascending=True)

# activities_overview.tail(5)

```

```

print("Number of Strava Activities Found: ", activities_overview.shape)
print("")

activities_overview.to_csv('activities_overview.csv', header=True)
print("OVERVIEW CSV FILE UPDATED\n")

# (https://faun.pub/write-files-from-ec2-to-s3-in-aws-programmatically-716d1a4ef639)
cli.put_object(
    Body='activities_overview.csv',
    Bucket='sarosfit',
    Key='data/activities_overview.csv')

print("OVERVIEW FILE UPDATED IN S3 BUCKET\n")

# ## Create Dataframe with DETAILED ACTIVITY DATA Streams for New Activities *(Strava API)*
# Streams Available via Strava API (https://developers.strava.com/docs/reference/#api-models-StreamSet)
# time.....TimeStream An instance of TimeStream.
# distance.....DistanceStream An instance of DistanceStream.
# latlng.....LatLangStream An instance of LatLangStream.
# altitude.....AltitudeStream An instance of AltitudeStream.
# velocity_smooth.....SmoothVelocityStream An instance of SmoothVelocityStream.
# heartrate.....HeartrateStream An instance of HeartrateStream.
# cadence.....CadenceStream An instance of CadenceStream.
# watts.....PowerStream An instance of PowerStream.
# temp.....TemperatureStream An instance of TemperatureStream.
# moving.....MovingStream An instance of MovingStream.
# grade_smooth.....SmoothGradeStream An instance of SmoothGradeStream.

# https://www.strava.com/api/v3/activities/4998708851streams?access_token=#####&keys=moving&key_by_type=true

def activity_streams(id):
    a_dict = {}
    a_df = pd.DataFrame()
    a_url = "https://www.strava.com/api/v3/activities/"

    streams_list = ['time','distance','latlng','altitude','velocity_smooth','heartrate','cadence','watts','temp',
                    'moving','grade_smooth']
    streams_text = 'time,distance,latlng,altitude,velocity_smooth,heartrate,cadence,watts,temp,moving,grade_smooth'

    a_json = pd.json_normalize(requests.get(a_url + str(id) + '/streams?access_token=' + access_token +
                                           '&keys=time,distance,latlng,altitude,velocity_smooth,heartrate,cadence,watts,temp,moving,grade_smooth' +
                                           '&key_by_type=true').json())

    for a in range(0,len(streams_list)):
        try:
            a_df[streams_list[a]] = a_json[str(streams_list[a]) + '.data'][0]
        except:
            a_df[streams_list[a]] = np.nan

    a_df['id'] = id
    idx = activities_overview.index[activities_overview['id'] == id].tolist()[0]

    a_df['date'] = activities_overview['start_date_local'][idx]
    a_df['name'] = activities_overview['name'][idx]
    a_df['type'] = activities_overview['type'][idx]

    return a_df

# ### Load Already Downloaded Activity Details if Present
try:
    # Check to see if there is a local file
    activities_details = pd.read_pickle('activities_details.pkl')
except:
    try:
        # Check the s3 bucket to see if there is a file
        cli.download_file(
            Bucket='sarosfit',
            Key='data/activities_details.pkl',
            Filename='activities_details.pkl')
    except:
        # Create a new empty dataframe (usually because first run)
        activities_details = pd.DataFrame()

# activities_details.info()
# activities_details.tail()

# ### Download only Details for New Activities
# (https://thispointer.com/pandas-check-if-a-value-exists-in-a-dataframe-using-in-not-in-operator-isin/)

a_details_to_import = []
try:
    a_already_downloaded = activities_details['id'].unique()
except:
    a_already_downloaded = []

```

```

for a in activities_overview['id']:
    # faster searching in only one column
    try:
        if a in a_already_downloaded:
            pass
        else:
            a_details_to_import.append(a)

    # if empty dataframe searching in 'id' column will fail
    except:
        if a in activities_details.values:
            pass
        else:
            a_details_to_import.append(a)

print("Number of Activities to Import: " + str(len(a_details_to_import)))

# Activities that have no details will always show up because they will never have any details added
# a_details_to_import

a_range_l = 0
a_range_h = 89
a_number = len(a_details_to_import)

while a_range_l < a_number:
    if a_range_l > 0:
        print('Waiting...')
        # wait a little over 16m40s to be safe (100 requests per 15min limit)
        time.sleep(1000)

    print('Downloading activities ' + str(a_range_l) + ' to ' + str(a_range_h) + ' ...')

    for a in a_details_to_import[a_range_l:a_range_h]:
        print('Downloading activity ', a)
        a_df_curr = activity_streams(a)
        activities_details = activities_details.append(a_df_curr, ignore_index=True)

    # 90 rather than 100 to be safe
    a_range_l = a_range_l + 90
    a_range_h = a_range_h + 90

print('Done getting details for all new activities.\n')

# print(activities_details.head(2))
# print(activities_details.tail(2))

print("Number of Rows for all Activities Found: ", activities_details.shape)
print("Sum of Moving Time: ", activities_overview['moving_time'].sum())
print("Sum of Elapsed Time: ", activities_overview['elapsed_time'].sum())

# resolve Error: _pickle.PicklingError: Can't pickle <NA>; it's not the same object as pandas._libs.missing.NA
activities_details['latlng'] = activities_details['latlng'].astype("string")
activities_details = activities_details.replace('<NA>', '')

activities_details.to_pickle('activities_details.pkl')
activities_details.to_csv('activities_details.csv', header=True)

print("\nDETAILED CSV and PKL FILES UPDATED\n")

cli.put_object(
    Body='activities_details.csv',
    Bucket='sarosfit',
    Key='data/activities_details.csv')

cli.put_object(
    Body='activities_details.pkl',
    Bucket='sarosfit',
    Key='data/activities_details.pkl')

print("DETAILED FILES UPDATED IN S3 BUCKET\n")

print("EXITING SAROS FIT\n")

```

STEP 3: CREATE S3 BUCKET

Create S3 bucket and create a 'data' folder to store the files created by the python script

Create bucket Info

Buckets are containers for data stored in S3. [Learn more](#)

General configuration

Bucket name
 Bucket name must be unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

AWS Region

Copy settings from existing bucket - *optional*
Only the bucket settings in the following configuration are copied.
[Choose bucket](#)

Create folder Info

Use folders to group objects in buckets. When you create a folder, S3 creates an object using the name that you specify followed by a slash (/). This object then appears as folder on the console. [Learn more](#)

Your bucket policy might block folder creation
If your bucket policy prevents uploading objects without specific tags, metadata, or access control list (ACL) grantees, you will not be able to create a folder using this configuration. Instead, you can use the [upload configuration](#) to upload an empty folder and specify the appropriate settings.

Folder

Folder name
 /

Folder names can't contain "/". [See rules for naming](#)

STEP 4: AMAZON EC2, SSH, CLI

Launch EC2 instance with SSH access from Local IP address

Instance summary for i-02ea57d86acf8d5e4 (Strava-Python-Updater)		Info	C	Connect	Instance state	Actions
Updated less than a minute ago						
Instance ID	(Strava-Python-Updater)	Public IPv4 address	18.208.182.29 open address	Private IPv4 addresses	172.31.85.14	
IPv6 address	-	Instance state	Running	Public IPv4 DNS	ec2-18-208-182-29.compute-1.amazonaws.com open address	
Hostname type	IP name: ip-172-31-85-14.ec2.internal	Private IP DNS name (IPv4 only)	ip-172-31-85-14.ec2.internal	Answer private resource DNS name	IPv4 (A)	
Instance type	t2.micro	Elastic IP addresses	-	VPC ID		
AWS Compute Optimizer finding	Opt-in to AWS Compute Optimizer for recommendations. Learn more	IAM Role	-	Subnet ID		

Connect via SSH to the instance and run the following commands

- aws configure (enter Key ID, Secret Access Key, Region (*same as instance and bucket*))
- python3 -m venv my_app/env
- source ~/my_app/env/bin/activate
- sudo yum -y update
- pip install pip --upgrade
- pip install boto3
- pip install numpy
- pip install pandas
- pip install requests
- aws secretsmanager create-secret --name sarosfit_client_id --secret-string xxx
- aws secretsmanager create-secret --name sarosfit_client_secret --secret-string xxx
(Update python scripts with ARNs for each secret)
- Upload following local python files using SCP and move to correct directory:
scp -i "US-East-2_Sheraz_E90.pem" saros-fit.py ec2-user@172-31-85-14:~/.
-or-
Use nano editor to create the following python scripts
 - saros-fit.py
 - (optional) test-credentials.py, test-s3.py, test-summary.py, test-pickle.py
- python saros-fit.py

Test all scripts to ensure that they work as expected.

- test-credentials.py should print 'Access Token Received!'
- test-s3.py should put a activities_overview_test.csv file in s3 and then get the same file from s3
- test-summary.py should get all summary activity data from Strava and write a activities_overview.csv
- test-pickle.py should not result in an error and exit cleanly

One of the issues I ran into when running the full script was the process getting killed. AWS CLI provided absolutely no input on what was happening.

```
(env) [ec2-user@ip-172-31-85-14 ~]$ python saros-fit.py

Requesting Token...
Access Token Received!

Number of Strava Activities Found: (534, 60)

OVERVIEW CSV FILE UPDATED

OVERVIEW FILE UPDATED IN S3 BUCKET

Killed

(env) [ec2-user@ip-172-31-85-14 ~]$
```

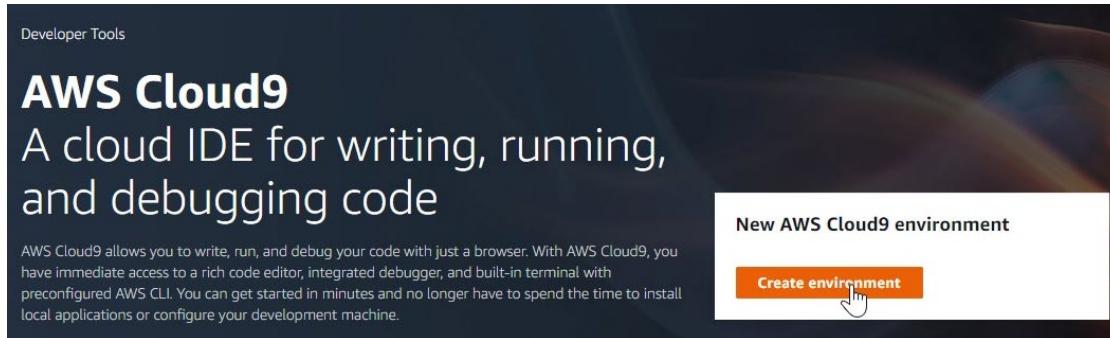
A lot of internet research led me to checking the last few entries in the log files (/var/log/messages) of the EC2 instance. The mention of out of memory made me realize that I needed a bigger size EC2 instance (moved from t2.micro to t2.medium)

```
Nov 30 06:54:04 ip-172-31-85-14 kernel: dump_header+0x4a/0x1f0
Nov 30 06:54:04 ip-172-31-85-14 kernel: oom_kill_process.cold+0xb/0x10
Nov 30 06:54:04 ip-172-31-85-14 kernel: out_of_memory.part_0+0xbd/0x260
Nov 30 06:54:04 ip-172-31-85-14 kernel: out_of_memory+0x3d/0x80
Nov 30 06:54:04 ip-172-31-85-14 kernel: __alloc_pages_slowpath.constrprop.0+0x8a8/0x960
Nov 30 06:54:04 ip-172-31-85-14 kernel: __alloc_pages_nodemask+0x2d6/0x300
Nov 30 06:54:04 ip-172-31-85-14 kernel: alloc_pages_vma+0x80/0x260
Nov 30 06:54:04 ip-172-31-85-14 kernel: do_anonymous_page+0xd3/0x510
Nov 30 06:54:04 ip-172-31-85-14 kernel: handle_mm_fault+0x333/0x640
Nov 30 06:54:04 ip-172-31-85-14 kernel: handle_mm_fault+0xbe/0x290
Nov 30 06:54:04 ip-172-31-85-14 kernel: do_user_addr_fault+0xb3/0x3e0
Nov 30 06:54:04 ip-172-31-85-14 kernel: exc_page_fault+0x68/0x120
Nov 30 06:54:04 ip-172-31-85-14 kernel: ?asm_exc_page_fault+0x8/0x30
Nov 30 06:54:04 ip-172-31-85-14 kernel: asm_exc_page_fault+0xe1/0x30
Nov 30 06:54:04 ip-172-31-85-14 kernel: RIP: 0033:0x7fc737c2ae3d
Nov 30 06:54:04 ip-172-31-85-14 kernel: Code: 01 00 00 48 83 fa 40 77 77 c5 fe 7f 44 17 e0 c5 fe 7f 07 c5 f8 77 c3 66 0f 1f 44 00 00 c5 f8 77 48 89 d1 40 0f b6 c6 48 89 fa <f3> aa 48 89 d0 c3 0f
1f 00 66 0f if 84 00 00 00 48 39 d1
Nov 30 06:54:04 ip-172-31-85-14 kernel: RSP: 002b:00007fecb0edb8 EFLAGS: 0010202
Nov 30 06:54:04 ip-172-31-85-14 kernel: RAX: 0000000000000000 RBX: 0000000000000000 RCX: 000000000015cef0
Nov 30 06:54:04 ip-172-31-85-14 kernel: RDX: 000007fc671b1010 RSI: 0000000000000000 RDI: 00007fcf8410000
Nov 30 06:54:04 ip-172-31-85-14 kernel: RBP: 00007ffecbdedbf0 R08: 0000000000000001 R09: 0000000000000000
Nov 30 06:54:04 ip-172-31-85-14 kernel: R10: 0000000000000002 R11: 0000000000000002 R12: 0000000000000000 R13: 00007ffec7217e6390 R14: 00007ffecb0edc4 R15: 00007ffecb0edb8
Nov 30 06:54:04 ip-172-31-85-14 kernel: Mem-Info:
active_anon:68 isolated_anon:22071 isolated_file:2 inactive_file:2 isolated_file:0#012 unevictable:0 dirty:0 writeback:0#012 slab_reclaimable:4196 slab_unreclaimable:4548#012 mapped:5 pageables:1579 bounces:0#012 free:1224 free_pc:1224 free_cma:0
Nov 30 06:54:04 ip-172-31-85-14 kernel: 9 active_anon/2kB inactive_anon/888284KB active_file:8KB inative_file:8KB isolated(anon):0KB isolated(file):0KB mapped:20KB dirty:0KB mapped:0KB shmem:4096 shmem_pinned:0KB shmem_thp:0KB shmem_pmdmapped:0KB shmem_thp:0KB writeback_tmpl:0KB kernel_stack:2064KB all unreclaimable? yes
Nov 30 06:54:04 ip-172-31-85-14 kernel: Node 0 DMA 0 free:4450kB min:15904kB max:736KB low:920kB high:104KB reserved highatomic:0KB active_anon:11276kB active_file:0KB inactive_file:0KB unevictable:0KB writepending:0KB present:15988kB managed:0KB 15904kB blocked:0KB pageables:1240KB bounce:0KB free_pc:0KB local_pc:0KB free_cma:0KB
Nov 30 06:54:04 ip-172-31-85-14 kernel: Node 0 DMA32: free:44196kB min:55392kB high:66468kB reserved highatomic:0KB active_anon:2728kB inactive_anon:876996kB active_file:8KB inactive_file:8KB unevictable:0KB writepending:0KB present:1032392kB managed:973236kB blocked:0KB pageables:6292kB bounce:0KB free_pc:488kB local_pc:488kB free_cma:0KB
Nov 30 06:54:04 ip-172-31-85-14 kernel: lowmem_reserve[], 0 92 92 92
Nov 30 06:54:04 ip-172-31-85-14 kernel: Node 0 DMA32: free:44196kB min:44316kB low:55392kB high:66468kB reserved highatomic:0KB active_anon:2728kB inactive_anon:876996kB active_file:8KB inactive_file:8KB unevictable:0KB writepending:0KB present:15988kB managed:0KB 15904kB blocked:0KB pageables:1240KB bounce:0KB free_pc:0KB local_pc:0KB free_cma:0KB
Nov 30 06:54:04 ip-172-31-85-14 kernel: lowmem_reserve[], 0 0 0 0
Nov 30 06:54:04 ip-172-31-85-14 kernel: Node 0 DMA: 1*4KB (0M) 1*8KB (0M) 2*16KB (0M) 3*32KB (0M) 2*512KB (0M) 2*1024KB (0M) 0*2048KB 0*4096KB
Nov 30 06:54:04 ip-172-31-85-14 kernel: Node 0 DMA32: 729*4KB (UME) 566*8KB (UME) 423*16KB (UME) 273*32KB (UME) 148*64KB (UME) 50*128KB (UME) 15*512KB (UME) 0*1024KB 0*2048KB 0*4096KB
KB = 44196kB
Nov 30 06:54:04 ip-172-31-85-14 kernel: Node 0 hugepages_total=0 hugepages_free=0 hugepages_surp=0 hugepages_size=2048kB
Nov 30 06:54:04 ip-172-31-85-14 kernel: 192 total pagecache pages
Nov 30 06:54:04 ip-172-31-85-14 kernel: 0 pages in swap cache
Nov 30 06:54:04 ip-172-31-85-14 kernel: Swap cache stats: add 0, delete 0, find 0/0
Nov 30 06:54:04 ip-172-31-85-14 kernel: Free swap = 0KB
Nov 30 06:54:04 ip-172-31-85-14 kernel: Total swap = 0KB
Nov 30 06:54:04 ip-172-31-85-14 kernel: 262045 pages RAM
Nov 30 06:54:04 ip-172-31-85-14 kernel: 0 pages HighMem/MovableOnly
Nov 30 06:54:04 ip-172-31-85-14 kernel: 14760 pages reserved
Nov 30 06:54:04 ip-172-31-85-14 kernel: 0 pages hwpoisoned
Nov 30 06:54:04 ip-172-31-85-14 kernel: Tasks state (memory values in pages):
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ pid ] uid tgid total_vm rss pgtables_bytes swapents oom_score_adj name
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 1705] 0 1705 9741 86 122880 0 0 systemd-journal
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 1737] 0 1737 29701 66 126976 0 0 lvmetad
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 1738] 0 1738 9344 190 122880 0 -1000 systemd-udevd
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 2521] 0 2521 14935 184 131072 0 -1000 auditd
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 2552] 0 2552 3156 42 69632 0 0 lsmd
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 2554] 0 2554 7116 90 98304 0 0 systemd-logind
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 2558] 81 2558 15089 119 155648 0 -900 dbus-daemon
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 2562] 32 2562 17338 137 176128 0 0 rpcbind
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 2564] 0 2564 25399 118 155648 0 0 gssproxy
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 2593] 998 2593 23512 212 221184 0 0 rngd
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 2595] 996 2595 38597 127 147456 0 0 chrony
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 2796] 0 2796 25171 517 225288 0 0 dhclient
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 2803] 0 2803 25171 511 208656 0 0 dhclient
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 2808] 0 2808 23777 363 192512 0 0 master
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 2901] 8 2901 22618 256 200704 0 0 qmgr
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 3047] 8 3047 54650 229 188416 0 0 rsyslogd
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 3050] 8 3050 179174 1748 122880 0 0 amazon-ssm-agen
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 3058] 8 3058 6171 158 94208 0 0 crond
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 3070] 8 3070 6972 51 98304 0 0
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 3100] 8 3100 2638 31 69632 0 0
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 3101] 8 3101 30326 31 73728 0 0
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 3209] 8 3209 182280 3276 159744 0 0
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 3225] 8 3225 28218 256 258048 0 -1000 sshd
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 3252] 8 3252 1066 26 53248 0 0 acpid
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 3281] 8 3281 37636 330 335872 0 0 sshd
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 3299] 1000 3299 37676 380 331776 0 0 sshd
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 3300] 1000 3300 31678 254 77824 0 0 bash
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 3301] 89 1053 22599 255 208896 0 0 pickup
Nov 30 06:54:04 ip-172-31-85-14 kernel: [ 1054] 1008 1054 305733 211756 2187264 0 0 python
Nov 30 06:54:04 ip-172-31-85-14 kernel:oom-kill:constraint=CONSTRAINT_NONE,nodemask=(null),cpuset=,global_oom,task_mmcg=/,task=python,pid=1054,uid=1000
Nov 30 06:54:04 ip-172-31-85-14 kernel: Out of memory: Killed process 1054 (python) total-vm:1222932kB, anon-rss:847016kB, file-rss:8kB, shmem-rss:0kB, UID:1000 pgtables:2136kB oom_score_adj:0
Nov 30 06:54:04 ip-172-31-85-14 kernel:oom_reaper:reaped process 1054 (python), now anon-rss:0kB, file-rss:0kB, shmem-rss:0kB
Nov 30 06:54:57 ip-172-31-85-14 dhclient[2843]: XMT: Solicit on eth0, interval 111460ms.
Nov 30 06:56:49 ip-172-31-85-14 dhclient[2843]: XMT: Solicit on eth0, interval 125740ms.
Nov 30 06:58:54 ip-172-31-85-14 dhclient[2843]: XMT: Solicit on eth0, interval 114010ms.
(env) [ec2-user@ip-172-31-85-14 log]$
```

STEP 5: CLOUD9 IDE ENVIRONMENT

After using the standard process outlined in Step 4, I decided to try the AWS Cloud9 IDE. Cloud9 greatly streamlines the process in Step 4 while being a lot more user-friendly and powerful!

Simply select 'Create environment'



Give the environment a Name and optionally a Description

A screenshot of the 'Name environment' step in the AWS Cloud9 'Create environment' wizard. The left sidebar shows steps: Step 1 'Name environment' (selected), Step 2 'Configure settings', and Step 3 'Review'. The main area has a title 'Name environment' and a section 'Environment name and description'. It contains two input fields: 'Name' (with 'SarosFit' entered) and 'Description - Optional' (with 'Connect to Strava and download overview and detailed activity data' entered). At the bottom right are 'Cancel' and 'Next step' buttons, with a hand cursor icon pointing at 'Next step'.

All the default options are appropriate except that for the `saros-fit.py` script we require a `t2.medium` instance type.

Note: Cloud9 EC2 instances even stop themselves automatically when not in use to save money.

Step 1
Name environment

Step 2
Configure settings

Step 3
Review

Configure settings

Environment settings

Environment type [Info](#)
Run your environment in a new EC2 instance or an existing server. With EC2 instances, you can connect directly through Secure Shell (SSH) or connect via AWS Systems Manager (without opening inbound ports).

Create a new EC2 instance for environment (direct access)
Launch a new instance in this region that your environment can access directly via SSH.

Create a new no-ingress EC2 instance for environment (access via Systems Manager)
Launch a new instance in this region that your environment can access through Systems Manager.

Create and run in remote server (SSH connection)
Configure the secure connection to the remote server for your environment.

Instance type

t2.micro (1 GiB RAM + 1 vCPU)
Free-tier eligible. Ideal for educational users and exploration.

t3.small (2 GiB RAM + 2 vCPU)
Recommended for small-sized web projects.

m5.large (8 GiB RAM + 2 vCPU)
Recommended for production and general-purpose development.

Other instance type
Select an instance type.

t2.medium ▾

Platform

Amazon Linux 2 (recommended)

Amazon Linux AMI

Ubuntu Server 18.04 LTS

Cost-saving setting
Choose a predetermined amount of time to auto-hibernate your environment and prevent unnecessary charges. We recommend a hibernation settings of half an hour of no activity to maximize savings.

After 30 minutes (default) ▾

IAM role
AWS Cloud9 creates a service-linked role for you. This allows AWS Cloud9 to call other AWS services on your behalf. You can delete the role from the AWS IAM console once you no longer have any AWS Cloud9 environments.[Learn more](#)

AWSServiceRoleForAWSCloud9

▶ **Network settings (advanced)**

No tags associated with the resource.

Add new tag

You can add 50 more tags.

Cancel Previous step **Next step** 

That is everything required and in this step we just review the setting and click ‘Create environment’.

The screenshot shows the 'Review' step of a Cloud9 environment creation wizard. On the left, a sidebar lists 'Step 1 Name environment', 'Step 2 Configure settings', and 'Step 3 Review'. The main area is titled 'Environment name and settings' and contains the following configuration details:

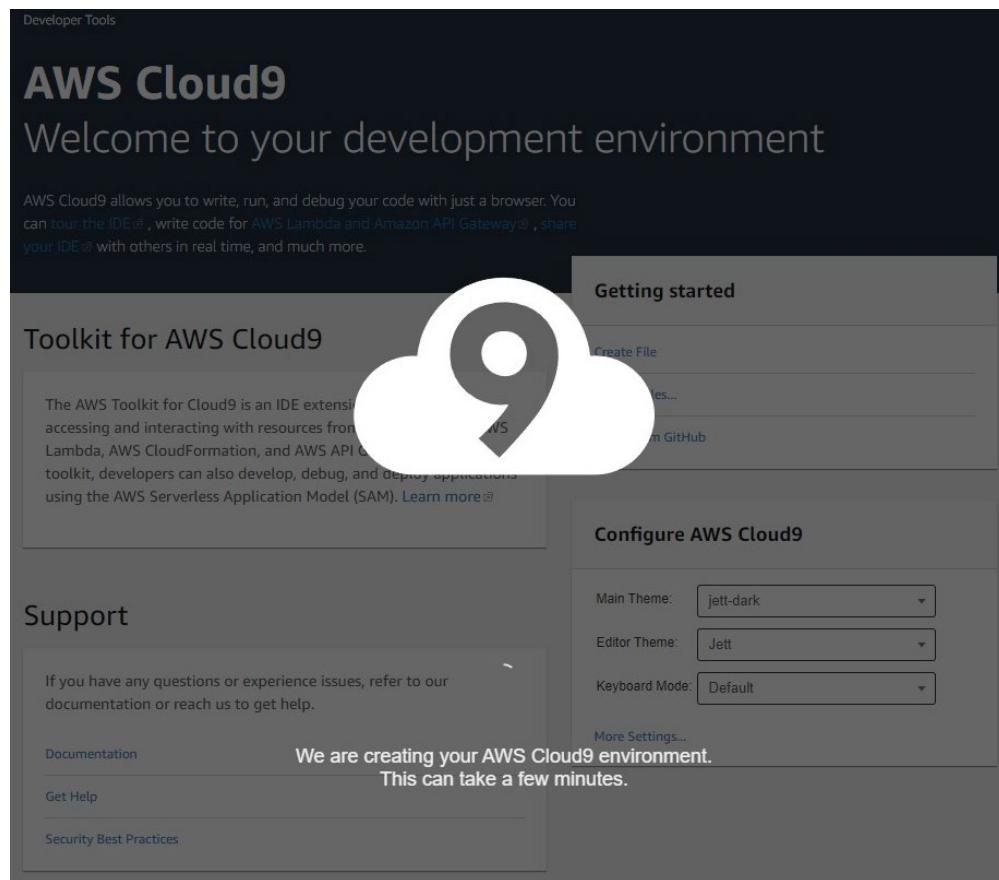
- Name: SarosFit
- Description: Connect to Strava and download all activities for a user (overview and detailed data).
- Environment type: EC2
- Instance type: t2.micro
- Subnet: (not explicitly named)
- Platform: Amazon Linux 2 (recommended)
- Cost-saving settings: After 30 minutes (default)
- IAM role: AWSServiceRoleForAWSCloud9 (generated)

A callout box provides best practices for using the environment:

- Use **source control** and **backup** your environment frequently. AWS Cloud9 does not perform automatic backups.
- Perform regular **updates of software** on your environment. AWS Cloud9 does not perform automatic updates on your behalf.
- Turn on **AWS CloudTrail** in your **AWS account** to track activity in your environment. [Learn more](#)
- Only share your environment with **trusted users**. Sharing your environment may put your AWS access credentials at risk. [Learn more](#)

At the bottom, there are three buttons: 'Cancel', 'Previous step', and a large orange 'Create environment' button with a hand cursor icon over it.

In 2-3 minutes we have an EC2 instance with python already installed an a user-friendly IDE environment



```

277     print("Downloading activities " + str(a_range_i) + ' to ' + str(a_range_h) + ' ...')
278
279     for a in a_details_to_import[a_range_i:a_range_h]:
280         print("Downloaded activity " + str(a))
281         a_df = pd.read_csv(str(a))
282         a_df['activity_streams'] = a
283         activities_details['activities_details'].append(a_df, ignore_index=True)
284
285     # 90 credits less than 100 to be safe
286     a_range_l = a_range_i + 90
287     a_range_h = a_range_h + 90
288
289     print('Done getting details for all new activities.\n')
290
291     # display(activities_details.head(2))
292     # display(activities_details.tail(2))
293
294     print('Number of Rows for all Activities Found: ', activities_overview['activity'].shape)
295     print('Sum of Moving Time: ', activities_overview['moving_time'].sum())
296     print('Sum of Cessed Time: ', activities_overview['elapsed_time'].sum())
297
298     activities_details.to_csv('activities_details.csv', header=True)
299     activities_details.to_pickle('activities_details.pkl')
300
301     print("\nDETAILED CSV AND PKL FILES UPDATED\n")
302
303     cli.put_object(
304         Bucket="sarosfit",
305         Key="data/activities_details.pkl",
306         Body="activities_details.pkl",
307         Bucket="sarosfit",
308         Key="data/activities_details.pkl"
309     )
310     cli.put_object(
311         Body="activities_details.pkl",
312         Bucket="sarosfit",
313         Key="data/activities_details.pkl"
314     )
315     print("DETAILLED FILES UPATED IN S3 BUCKET\n")
316
317     print("EXITING SAROS FIT\n")

```

Requesting Token...
Access Token Received!
Number of Strava Activities Found: (538, 66)
OVERVIEW CSV FILE UPDATED
OVERVIEW FILE UPDATED IN S3 BUCKET
Number of Activities to Import: 26
Downloading activities 0 to 89 ...
Downloading activity 325167893
Downloading activity 4595615123
Downloading activity 4595618160
Downloading activity 4611236404
Downloading activity 4611236409
Downloading activity 4686429811

We can view all our environments and their associated details

AWS Cloud9 > Your environments

Your environments (1)

SarosFit

Type: EC2 Permissions: Owner

Description:
Connect to Strava and download all activities for a user (overview and detailed data).

Owner Arn:
arn:aws:iam::355134978792:user/e90_learner

Open IDE

AWS Cloud9 > Environments > SarosFit

SarosFit

Environment details

Name SarosFit	EC2 instance type t2.micro	Security groups [Redacted]	Environment ARN arn:aws:cloud9:us-east-[Redacted]
Description Connect to Strava and download all activities for a user (overview and detailed data).	Memory 1 GiB	VPC [Redacted]	Number of members 1
Type EC2	vCPU 1	Subnet [Redacted]	Lifecycle State CREATED
Owner ARN [Redacted]	Storage EBS only	EC2 Instance Go To Instance	Environment path /home/ec2-user/environment

Clicking on the EC2 Instance link we can view that EC2 instance that Cloud9 automatically created

Instances (1/2) [Info](#)

Filter instances

Name	Instance ID	Instance state	Instance type	Status check	Alarm st.
Strava-Python-Updater	[Redacted]	Running	t2.micro	2/2 checks p.	No alarm
aws-cloud9-SarosFit-[Redacted]	[Redacted]	Running	t2.micro	2/2 checks p.	No alarm

EC2 > Instances > i-006090d9897e4415a

Instance summary for i- [Info]

Updated less than a minute ago

C Connect Instance state Actions

Instance ID	Public IPv4 address	Private IPv4 addresses
	54.81.200.219 open address	172.31.45.174
IPv6 address	Instance state	Public IPv4 DNS
-	Running	ec2-54-81-200-219.compute-1.amazonaws.com open address
Hostname type	Private IP DNS name (IPv4 only)	Answer private resource DNS name
IP name: ip-172-31-45-174.ec2.internal	ip-172-31-45-174.ec2.internal	-
Instance type	Elastic IP addresses	VPC ID
t2.micro	-	[REDACTED]
AWS Compute Optimizer finding	IAM Role	Subnet ID
Opt-in to AWS Compute Optimizer for recommendations. Learn more	-	[REDACTED]

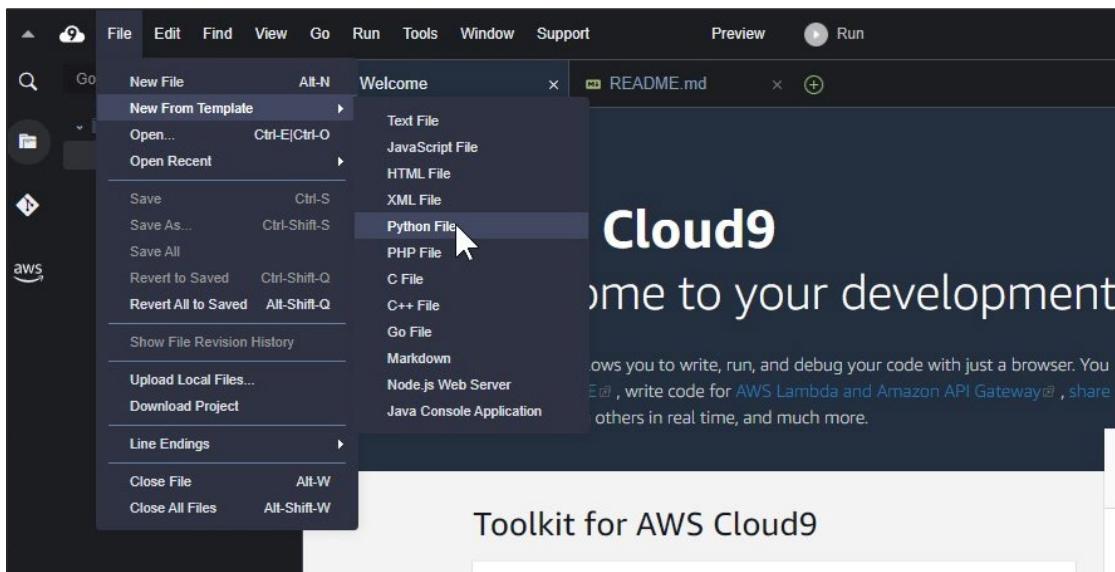
Details Security Networking Storage Status checks Monitoring Tags

Tags

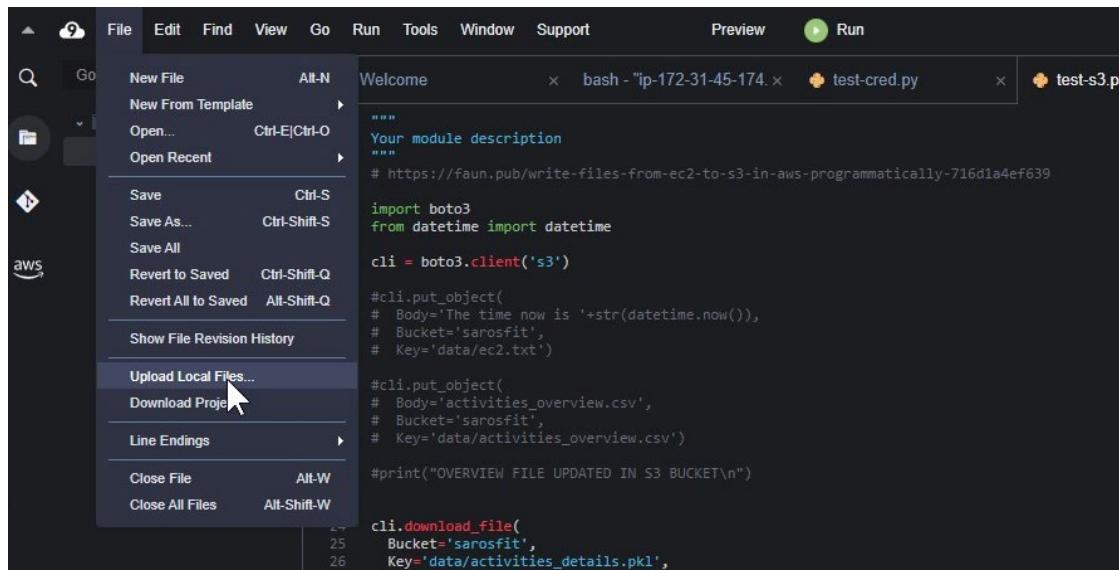
Manage tags

Key	Value
Name	
aws:cloudformation:stack-name	
aws:cloud9:environment	
aws:cloudformation:stack-id	
aws:cloud9:owner	
aws:cloudformation:logical-id	

Creating a new file can easily be accomplished from the 'File' menu



Uploading local files is as simple as clicking and dragging versus using scp from the command line



```

File Edit Find View Go Run Tools Window Support Preview Run
New File Alt-N
New From Template
Open... Ctrl-E|Ctrl-O
Open Recent
Save Ctrl-S
Save As... Ctrl-Shift-S
Save All
Revert to Saved Ctrl-Shift-Q
Revert All to Saved Alt-Shift-Q
Show File Revision History
Upload Local Files...
Download Project
Line Endings
Close File Alt-W
Close All Files Alt-Shift-W

```

```

Welcome bash - "ip-172-31-45-174.x" test-cred.py test-s3.py
"""
Your module description
"""

# https://faun.pub/write-files-from-ec2-to-s3-in-aws-programmatically-716d1a4ef639

import boto3
from datetime import datetime

cli = boto3.client('s3')

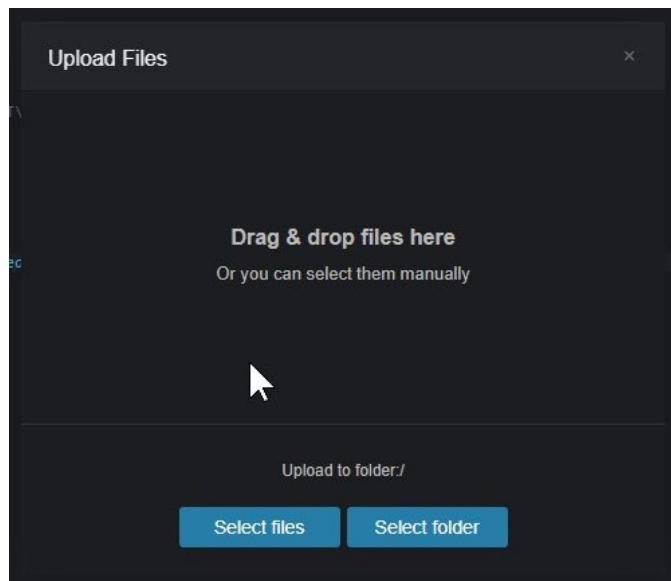
#cli.put_object(
#    Body='The time now is '+str(datetime.now()),
#    Bucket='sarosfit',
#    Key='data/ec2.txt')

#cli.put_object(
#    Body='activities_overview.csv',
#    Bucket='sarosfit',
#    Key='data/activities_overview.csv')

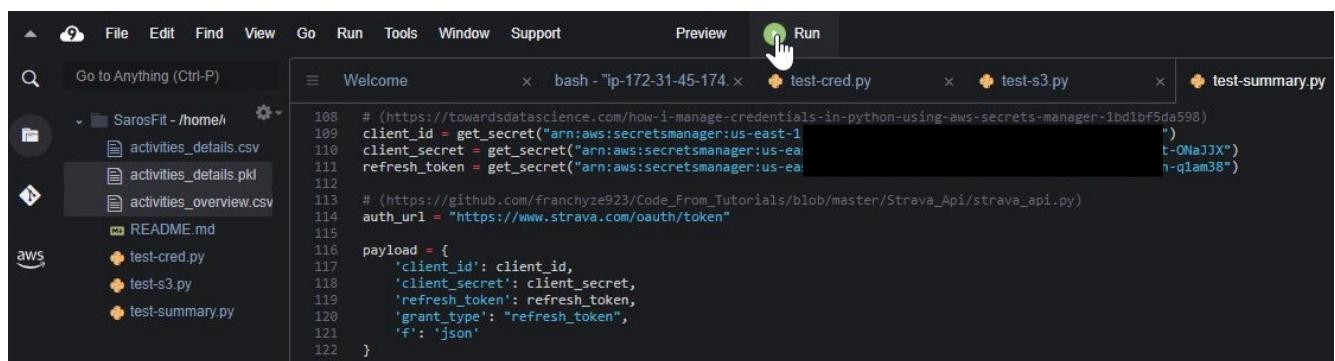
#print("OVERVIEW FILE UPDATED IN S3 BUCKET\n")

cli.download_file(
    Bucket='sarosfit',
    Key='data/activities_details.pkl',
    Filename='activities_details.pkl')

```



To execute a script click the 'Run' button and the output of the code is displayed in the bottom pane



```

File Edit Find View Go Run Tools Window Support Preview Run
Go to Anything (Ctrl-P)
activities_details.csv
activities_overview.csv
README.md
test-cred.py
test-s3.py
test-summary.py

```

```

>Welcome bash - "ip-172-31-45-174.x" test-cred.py test-s3.py test-summary.py
108 # (https://towardsdatascience.com/how-i-manage-credentials-in-python-using-aws-secrets-manager-1bd1bf5da598)
109 client_id = get_secret("arn:aws:secretsmanager:us-east-1")
110 client_secret = get_secret("arn:aws:secretsmanager:us-east-1")
111 refresh_token = get_secret("arn:aws:secretsmanager:us-east-1")
112
113 # (https://github.com/franchize923/Code_From_Tutorials/blob/master/Strava_Api/strava_api.py)
114 auth_url = "https://www.strava.com/oauth/token"
115
116 payload = {
117     'client_id': client_id,
118     'client_secret': client_secret,
119     'refresh_token': refresh_token,
120     'grant_type': "refresh_token",
121     'f': 'json'
122 }

```

```

bash - "ip-172-31-45-174.x" Immediate x test-cred.py - Stopped x test-summary.py - Stopped x +
Run Command: test-summary.py

Requesting Token...
Access Token Received!

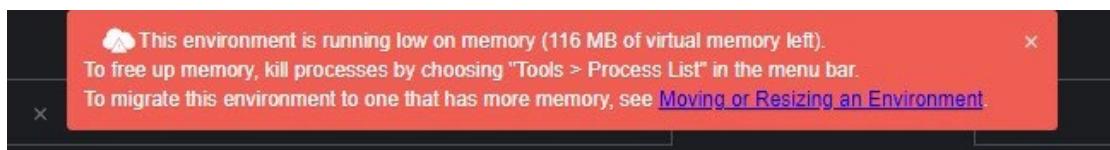
Number of Strava Activities Found: (536, 60)

OVERVIEW CSV FILE UPDATED

Process exited with code: 0

```

Unlike AWS CLI, when the instance is running low on memory it displays a clear message to help quickly figure out why the script is failing



But it is still important to test all scripts in AWS even if they were tested locally. In AWS, <NA> values stored in the latlng column caused an error when trying to pickle the dataframe. Locally, the script ran fine without any errors.

```

Traceback (most recent call last):
  File "/home/ec2-user/environment/saros-fit.py", line 288, in <module>
    activities_details.to_pickle('activities_details.pkl')
  File "/home/ec2-user/.local/lib/python3.7/site-packages/pandas/core/generic.py", line 2962, in to_pickle
    storage_options=storage_options,
  File "/home/ec2-user/.local/lib/python3.7/site-packages/pandas/io/pickle.py", line 119, in to_pickle
    protocol=protocol,
_pickle.PicklingError: Can't pickle <NA>: it's not the same object as pandas._libs.missing.NA

```

time	395.0
distance	1599.0
latlng	<NA>
altitude	0.0
velocity_smooth	3.7
heartrate	157.0
cadence	23.0
watts	207.0
temp	0.0
moving	1.0
grade_smooth	0.0
id	5287226391.0
date	2021-05-12 16:57:49+00:00
name	500m Intervals x 4 (2,2,4 min rest)
type	Rowing
Name:	970253, dtype: object

STEP 6: AWS SECRET MANAGER

For storing the Strava Access Tokens securely I used AWS Secret Manager. Passwords can be stored either via the command line or via the website.

From Command Line:

- aws secretsmanager create-secret --name sarosfit_client_id --secret-string xxx
- aws secretsmanager create-secret --name sarosfit_client_secret --secret-string xxx

Note: The python script should be updated with the ARN's returned

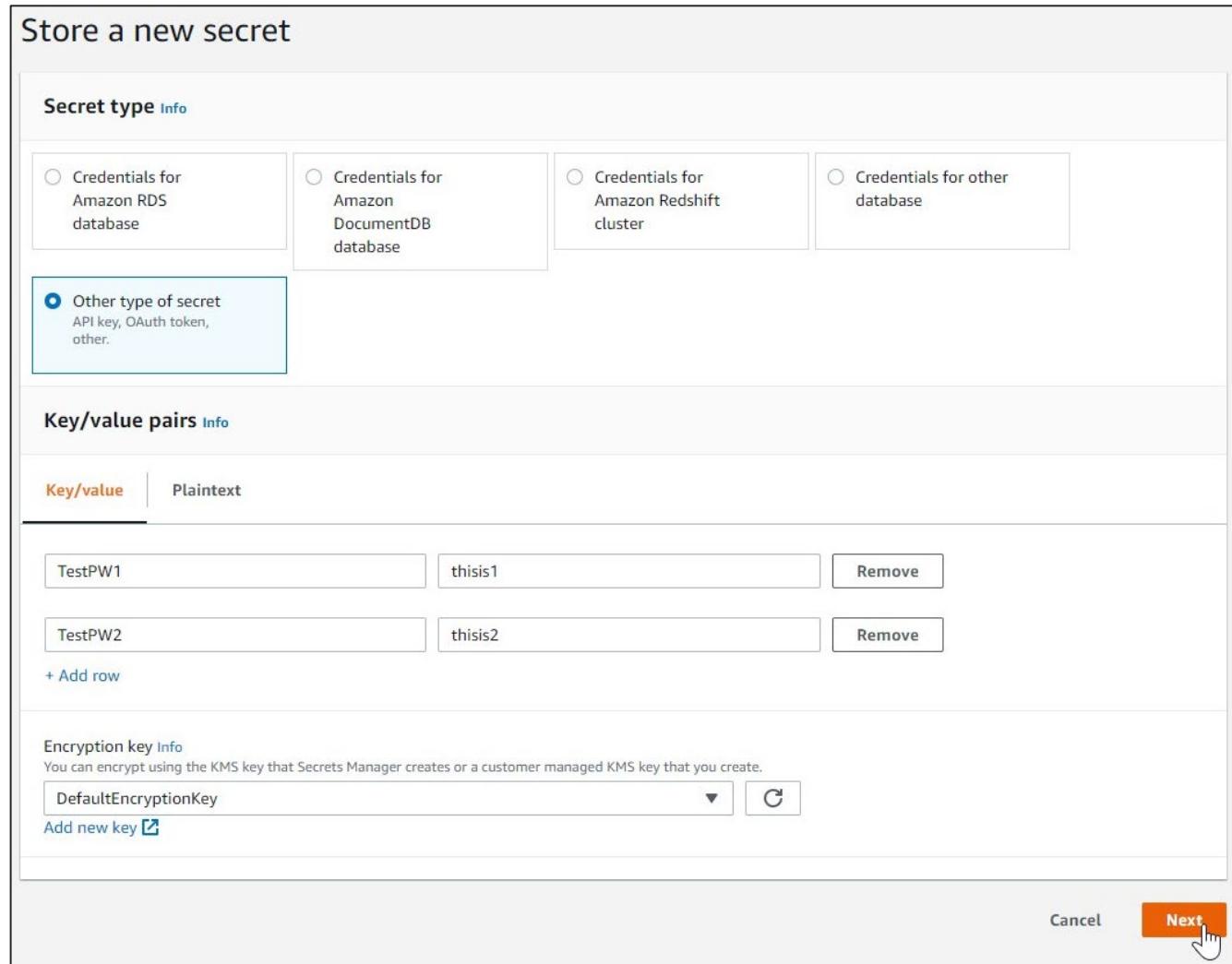
From the Website:

Click on 'Store a new secret' and click on secret type desired.

In our case we want to store a token so we pick 'Other type of secret'

Key/value pairs can be entered simply

To store plain text, click on Plaintext and then clear the pre-entered text from the textbox and then enter the value



Store a new secret

Secret type Info

Credentials for Amazon RDS database Credentials for Amazon DocumentDB database Credentials for Amazon Redshift cluster Credentials for other database

Other type of secret API key, OAuth token, other.

Key/value pairs Info

Key	Value	Action
TestPW1	thisis1	Remove
TestPW2	thisis2	Remove
+ Add row		

Encryption key Info
You can encrypt using the KMS key that Secrets Manager creates or a customer managed KMS key that you create.

DefaultEncryptionKey

Add new key

Cancel **Next Step**

Secret value Info
Retrieve and view the secret value.

Key/value | **Plaintext**

Secret key	Secret value
TestPW1	thisis1
TestPW2	thisis2

Secret value Info
Retrieve and view the secret value.

Key/value | **Plaintext**

```
{
  "TestPW1": "thisis1",
  "TestPW2": "thisis2"
}
```

Give the secret a name, a description (*optional*) and type of secret rotation (*optional*)

AWS Secrets Manager > Secrets > Store a new secret

Store a new secret

Secret name and description Info

Secret name
A descriptive name that helps you find your secret later.
TestPW

Secret name must contain only alphanumeric characters and the characters /_+=,.@-

Description - optional
A test only
Maximum 250 characters.

Store a new secret

If you turn on automatic rotation, the first rotation will happen immediately when you store this secret. See [Rotation](#) in the Secrets Manager User Guide.

Secret rotation - optional Info
Configure Secrets Manager to rotate this secret automatically. See [Rotation](#) in the Secrets Manager User Guide.

Disable automatic rotation
Secrets Manager will not rotate your secret.

Enable automatic rotation
We recommend that you rotate your secrets every 30 days.

Rotation interval Info
The number of days between rotations of this secret.
30 days

Must be a value from 1 to 365 days.

Lambda rotation function Info
Choose a Lambda function that can rotate this secret.
Create function

Cancel Previous

Copy python script to access secret.

Modify the script to return the secret value

Test to ensure the python code works using Cloud9 IDE

Sample code

Use these code samples to retrieve the secret in your application.

[Java](#) | [JavaV2](#) | [JavaScript](#) | [C#](#) | [Python3](#) **Python3** | [Ruby](#) | [Go](#)

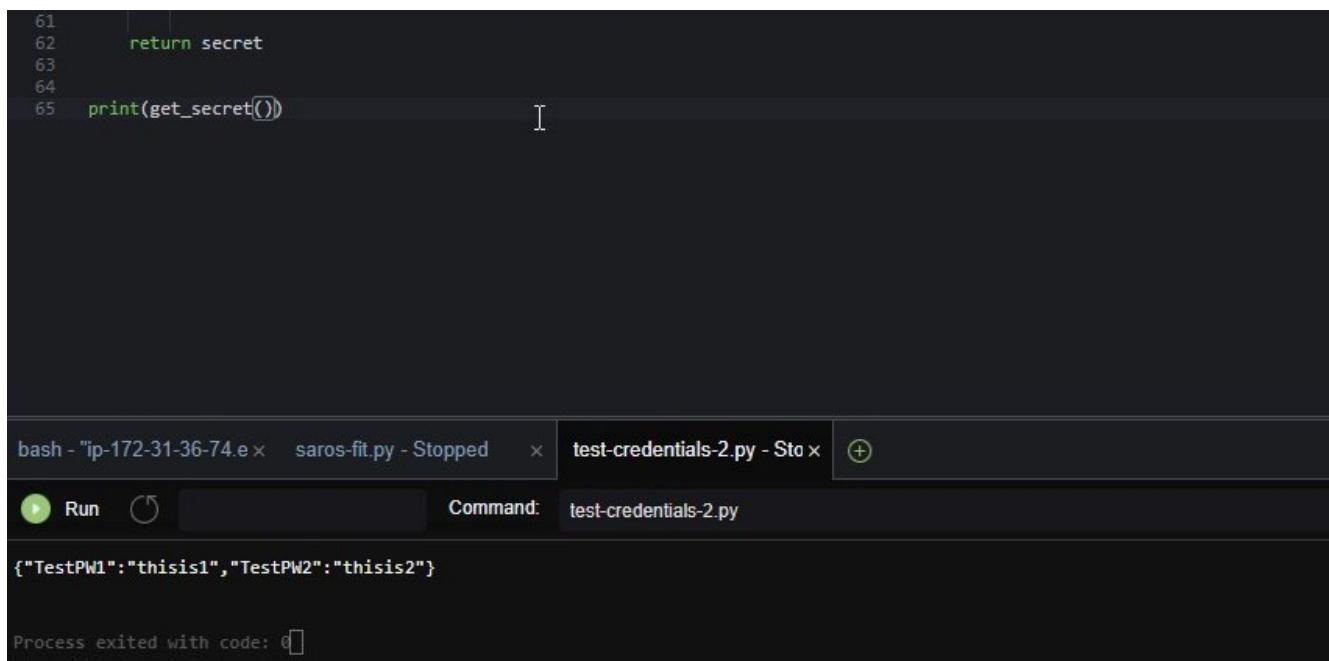
```

1  # Use this code snippet in your app.
2  # If you need more information about configurations or implementing the sample code, visit the AWS docs:
3  # https://aws.amazon.com/developers/getting-started/python/
4
5  import boto3
6  import base64
7  from botocore.exceptions import ClientError
8
9
10 def get_secret():
11     secret_name = "TestPW"
12     region_name = "us-east-1"
13
14     # Create a Secrets Manager client
15     session = boto3.session.Session()
16     client = session.client(
17         service_name='secretsmanager',
18         region_name=region_name
19     )
20

```

[Download AWS SDK for Python](#)

[Cancel](#) [Previous](#) **Store**



```

61
62     return secret
63
64
65 print(get_secret())

```

bash - "ip-172-31-36-74.e.x" saros-fit.py - Stopped x test-credentials-2.py - Stopped x (+)

Run Command: test-credentials-2.py

```

["TestPW1":"thisis1","TestPW2":"thisis2"]

Process exited with code: 0

```

OUTPUT OF SCRIPT

Files in S3 Bucket

- activities_overview.csv contains all a user's activities along with 59 columns of summary data
- activities_details.csv contains 11 datapoints for every second of each activity
- activities_details.pkl file contains same info as activities_details.csv (*only used by script*)

Amazon S3 > sarosfit > data/

data/

Objects | Properties

Objects (3)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant

C **Copy S3 URI** **Copy URL** **Download** **Open** **Delete** **Actions** **Create folder** **Upload**

Find objects by prefix

<input type="checkbox"/>	Name	Type	Last modified
<input type="checkbox"/>	activities_details.csv	csv	December 3, 2021, 17:21:15 (UTC-05:00)
<input type="checkbox"/>	activities_details.pkl	pkl	December 3, 2021, 17:21:15 (UTC-05:00)
<input type="checkbox"/>	activities_overview.csv	csv	December 3, 2021, 17:20:33 (UTC-05:00)

The activities overview file is perfect for getting a summary overview of all Strava activities.

The activities details file is perfect for running detailed analysis on every second of every activity.

Examples of data from these two files are provided below.

Example of data from one activity in activities_overview.csv

Example of data from a sample activity in activities_details.csv

(my personal 548 strava activities have 1.8 million rows in the activities_details file)

time	distance	latng	altitude	velocity_smooth	heartrate	cadence	watts	temp	moving	grade_smooth	id	date	name	type
1000	6324.500	[42.399068, -71.144309]	12.800	7.800	161	87	195	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1001	6332.400	[42.399135, -71.144341]	12.800	7.800	161	87	236	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1002	6340.300	[42.399197, -71.144382]	12.800	7.900	162	87	236	19	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1003	6348.000	[42.39925, -71.144436]	12.800	7.900	162	88	225	19	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1004	6355.800	[42.399291, -71.144505]	13.000	7.800	162	85	167	19	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1005	6363.700	[42.399334, -71.144583]	13.000	7.800	163	88	148	19	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1006	6371.600	[42.399383, -71.144649]	13.000	7.800	163	88	172	19	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1007	6379.500	[42.399442, -71.144698]	13.000	7.800	164	87	157	19	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1008	6387.500	[42.399506, -71.14474]	13.200	7.900	164	88	165	19	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1009	6395.500	[42.39957, -71.144784]	13.200	7.900	164	89	167	19	1	1.300	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1010	6403.500	[42.399637, -71.144821]	13.200	8.000	165	88	172	19	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1011	6411.400	[42.399706, -71.144849]	13.400	8.000	165	88	127	19	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1012	6419.300	[42.39977, -71.144869]	13.400	8.000	165	88	110	19	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1013	6426.900	[42.399852, -71.144871]	13.400	7.900	165	83	137	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1014	6434.500	[42.399926, -71.144853]	13.400	7.800	165	87	105	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1015	6440.800	[42.399954, -71.144819]	13.400	7.500	165	85	69	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1016	6446.100	[42.400051, -71.144774]	13.400	6.900	165	85	0	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1017	6450.300	[42.400095, -71.144732]	13.400	6.200	165	0	0	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1018	6455.500	[42.400131, -71.144702]	13.400	5.700	165	0	0	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1019	6461.400	[42.400166, -71.144682]	13.400	5.400	164	26	72	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1020	6468.100	[42.400208, -71.144671]	13.400	5.500	164	70	439	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1021	6475.300	[42.400206, -71.144673]	13.400	5.800	164	70	312	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1022	6482.600	[42.40032, -71.144469]	13.400	6.500	164	79	312	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1023	6490.000	[42.400381, -71.144714]	13.400	6.900	165	81	148	19	1	0.700	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1024	6497.500	[42.400443, -71.144742]	13.400	7.200	165	81	128	19	1	0.700	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1025	6504.900	[42.400505, -71.144772]	13.600	7.400	164	82	107	19	1	0.700	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1026	6512.300	[42.400567, -71.144801]	13.600	7.400	164	86	114	19	1	0.700	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1027	6519.600	[42.400663, -71.144831]	13.600	7.400	164	86	0	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1028	6526.800	[42.400669, -71.144865]	13.600	7.400	164	0	0	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1029	6533.500	[42.400745, -71.144908]	13.600	7.200	164	20	0	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1030	6540.700	[42.400788, -71.14496]	13.600	7.200	164	77	302	19	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1031	6548.000	[42.400883, -71.14502]	13.600	7.100	164	81	296	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1032	6555.600	[42.400879, -71.145071]	13.600	7.200	164	83	198	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1033	6563.400	[42.400936, -71.14511]	13.600	7.300	164	84	134	20	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1034	6571.200	[42.400996, -71.145146]	13.600	7.500	164	86	150	20	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1035	6579.000	[42.401057, -71.145186]	13.800	7.700	164	86	162	20	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1036	6586.800	[42.40112, -71.145225]	13.800	7.800	164	87	142	20	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1037	6594.800	[42.401184, -71.145265]	13.800	7.800	164	87	157	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1038	6602.600	[42.401215, -71.145305]	13.800	7.800	164	88	140	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1039	6610.500	[42.401316, -71.145344]	13.800	7.900	164	84	174	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1040	6618.300	[42.401383, -71.145381]	13.800	7.900	164	87	164	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1041	6626.300	[42.401415, -71.145416]	13.800	7.900	164	87	187	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1042	6634.200	[42.401516, -71.14545]	13.800	7.900	164	88	191	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1043	6642.100	[42.401582, -71.145486]	13.800	7.900	164	89	161	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1044	6650.600	[42.401647, -71.145522]	13.800	7.900	163	87	142	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1045	6658.000	[42.401712, -71.145557]	13.800	7.900	163	88	198	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1046	6666.100	[42.401779, -71.145592]	13.800	8.000	163	90	182	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1047	6674.000	[42.401849, -71.145626]	13.800	8.000	163	90	138	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1048	6682.000	[42.401913, -71.145661]	13.800	8.000	163	88	138	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1049	6689.800	[42.401976, -71.145695]	13.800	8.000	163	87	158	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1050	6697.600	[42.402041, -71.145727]	13.800	7.900	163	87	168	20	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1051	6705.500	[42.402103, -71.145757]	13.800	7.900	163	87	198	20	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1052	6713.300	[42.402165, -71.145784]	14.000	7.900	163	85	185	20	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1053	6721.100	[42.402277, -71.145814]	14.000	7.800	163	88	173	20	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1054	6728.900	[42.402288, -71.145842]	14.000	7.800	163	86	174	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1055	6736.600	[42.402352, -71.145873]	14.000	7.800	163	86	178	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1056	6744.400	[42.402413, -71.145903]	14.000	7.800	163	85	191	20	1	0.000	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1057	6752.100	[42.402478, -71.145943]	14.000	7.700	163	85	162	20	1	0.600	5010668618	2021-03-25 15:50:09+00:00	Bike Path to Bedford (so humid) ☀️	Ride
1058	6759.800	[42.402504, -71.145966]	14.000	7.700	163	86	193	20	1	0.000	5010668618	2021-03		

APPENDIX A: PYTHON SCRIPT TO RUN ON LOCAL MACHINE

Full Saros Fit Strava App Python Script for use on Local Machine (saros-fit-local.py)

```

# ---
# # Download Activity Summary and Details from Strava
#
# Strava provides basic information on how to get access to the API at https://developers.strava.com/docs/getting-started/.
# The information from the Strava API page combined with
# https://towardsdatascience.com/using-the-strava-api-and-pandas-to-explore-your-activity-data-d94901d9bfde and other websites
# allowed me to write the code necessary to connect to Strava.
#
# With some additional research I was able to download all my activities and their summary data. Though, in all my online
# searching I could not find any good examples of how to download all the detailed activity streams associated with each
# Strava activity. The code to download activity details was mostly written from scratch.
#
# **The value of this notebook is:**
#   - example of code to authenticate and connect to the Strava API (follow Strava instructions to create an app)
#   - download all activities and their associated summaries to a dataframe
#   - download all 11 detailed activity streams for each activity to a dataframe
#   - single notebook that authenticates to Strava, downloads activity summaries and downloads activity details
#
# **Additional useful features:**
#   - controls the number of requests so they don't exceed Strava's default limits (100 requests/15 min)
#   - stores all download activity details in pkl and csv format
#   - loads previously downloaded activity details and then only downloads details for new activities
#
# ##### NEXT STEPS:
#   - Create a separate notebook that loads the pkl file with activity details
#     - run basic analysis and create visualizations on the activity details
#     - run machine learning models on the activity details to find patterns in heartrate and wattage numbers
# ---
#
# **Written by: Sheraz Choudhary**
# **Date: November 2021**
# ---

# import libraries
import requests
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
import warnings

import numpy as np
import pandas as pd

import os
from os.path import join, dirname
import sys
import subprocess
import pathlib
from dotenv import load_dotenv

import codecs
from codecs import open
from datetime import date
import time

# ## Connect to Strava -- Get Current Access Token
# (https://www.realpythonproject.com/3-ways-to-store-and-read-credentials-locally-in-python/)
credential_file = join(os.getcwd(), 'strava-credentials.env')

load_dotenv(credential_file)
client_id = os.environ.get('client_id')
client_secret = os.environ.get('client_secret')
refresh_token = os.environ.get('refresh_token')

# (https://github.com/franchyze923/Code_From_Tutorials/blob/master/Strava_Api/strava_api.py)
auth_url = "https://www.strava.com/oauth/token"

payload = {
    'client_id': client_id,
    'client_secret': client_secret,
    'refresh_token': refresh_token,
    'grant_type': "refresh_token",
    'f': 'json'
}

print("\nRequesting Token...")
res = requests.post(auth_url, data=payload, verify=False)
access_token = res.json()['access_token']
print("Access Token Received!")
print("")

# ## Create Dataframe with Summary Info for All Activities *(Strava API)*
# (http://www.hainke.ca/index.php/2018/08/23/using-the-strava-api-to-retrieve-activity-data/)
# Initialize the dataframe

```

```

activities_overview = pd.DataFrame()

url = "https://www.strava.com/api/v3/athlete/activities"
page = 1

while True:
    # get page of activities from Strava
    page_json = requests.get(url + '?access_token=' + access_token + '&per_page=200' + '&page=' + str(page)).json()

    for a in range(len(page_json)):
        # (https://stackoverflow.com/questions/21104592/)
        activity_json = pd.json_normalize(page_json[a])
        activities_overview = activities_overview.append(activity_json, ignore_index=True)

    # if no results then exit loop
    if (not page_json):
        break

    # increment page
    page += 1

# makes sense since new added on bottom
activities_overview = activities_overview.sort_values(by='id', ascending=True)

# activities_overview.tail(5)

print("Number of Strava Activities Found: ", activities_overview.shape)
print("")

activities_overview.to_csv('activities_overview.csv', header=True)
print("OVERVIEW CSV FILE UPDATED\n")

# ## Create Dataframe with DETAILED ACTIVITY DATA Streams for New Activities *(Strava API)*
# Streams Available via Strava API (https://developers.strava.com/docs/reference/#api-models-StreamSet)
# time.....TimeStream An instance of TimeStream.
# distance.....DistanceStream An instance of DistanceStream.
# latlng.....LatLngStream An instance of LatLngStream.
# altitude.....AltitudeStream An instance of AltitudeStream.
# velocity_smooth.....SmoothVelocityStream An instance of SmoothVelocityStream.
# heartrate.....HeartrateStream An instance of HeartrateStream.
# cadence.....CadenceStream An instance of CadenceStream.
# watts.....PowerStream An instance of PowerStream.
# temp.....TemperatureStream An instance of TemperatureStream.
# moving.....MovingStream An instance of MovingStream.
# grade_smooth.....SmoothGradeStream An instance of SmoothGradeStream.

# https://www.strava.com/api/v3/activities/4998708851streams?access_token#####&keys=moving&key_by_type=true

def activity_streams(id):
    a_dict = {}
    a_df = pd.DataFrame()
    a_url = "https://www.strava.com/api/v3/activities/"

    streams_list = ['time','distance','latlng','altitude','velocity_smooth','heartrate','cadence','watts','temp',
                    'moving','grade_smooth']
    streams_text = 'time,distance,latlng,altitude,velocity_smooth,heartrate,cadence,watts,temp,moving,grade_smooth'

    a_json = pd.json_normalize(requests.get(a_url + str(id) + '/streams?access_token=' + access_token +
                                            '&keys=time,distance,latlng,altitude,velocity_smooth,heartrate,cadence,watts,temp,moving,grade_smooth' +
                                            '&key_by_type=true').json())

    for a in range(0,len(streams_list)):
        try:
            a_df[streams_list[a]] = a_json[str(streams_list[a]) + '.data'][0]
        except:
            a_df[streams_list[a]] = np.nan

    a_df['id'] = id
    idx = activities_overview.index[activities_overview['id'] == id].tolist()[0]

    a_df['date'] = activities_overview['start_date_local'][idx]
    a_df['name'] = activities_overview['name'][idx]
    a_df['type'] = activities_overview['type'][idx]

    return a_df

# ### Load Already Downloaded Activity Details if Present
try:
    activities_details = pd.read_pickle('activities_details.pkl')
except:
    activities_details = pd.DataFrame()

# activities_details.info()
# activities_details.tail()

# ### Download only Details for New Activities
# (https://thispointer.com/pandas-check-if-a-value-exists-in-a-dataframe-using-in-not-in-operator-isin/)

```

```

a_details_to_import = []
try:
    a_already_downloaded = activities_details['id'].unique()
except:
    a_already_downloaded = []

for a in activities_overview['id']:
    # faster searching in only one column
    try:
        if a in a_already_downloaded:
            pass
        else:
            a_details_to_import.append(a)

    # if empty dataframe searching in 'id' column will fail
    except:
        if a in activities_details.values:
            pass
        else:
            a_details_to_import.append(a)

print("Number of Activities to Import: " + str(len(a_details_to_import)))

# Activities that have no details will always show up because they will never have any details added
# a_details_to_import

a_range_l = 0
a_range_h = 89
a_number = len(a_details_to_import)

while a_range_l < a_number:
    if a_range_l > 0:
        print('Waiting...')
        # wait a little over 16m40s to be safe (100 requests per 15min limit)
        time.sleep(1000)

    print('Downloading activities ' + str(a_range_l) + ' to ' + str(a_range_h) + ' ...')

    for a in a_details_to_import[a_range_l:a_range_h]:
        print('Downloading activity ', a)
        a_df_curr = activity_streams(a)
        activities_details = activities_details.append(a_df_curr, ignore_index=True)

    # 90 rather than 100 to be safe
    a_range_l = a_range_l + 90
    a_range_h = a_range_h + 90

print('Done getting details for all new activities.\n')

# print(activities_details.head(2))
# print(activities_details.tail(2))

print("Number of Rows for all Activities Found: ", activities_details.shape)
print("Sum of Moving Time: ", activities_overview['moving_time'].sum())
print("Sum of Elapsed Time: ", activities_overview['elapsed_time'].sum())

activities_details.to_pickle('activities_details.pkl')
activities_details.to_csv('activities_details.csv', header=True)

print("\nDETAILED CSV and PKL FILES UPDATED")
print("")
print("EXITING SAROS FIT")

```

APPENDIX B: PYTHON TEST SCRIPTS

Acquire Refresh Token from Strava Python Script (test-credentials.py)

```

# import libraries
import numpy as np
import pandas as pd
import os
import sys
import subprocess
import pathlib
import requests
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

import codecs
from codecs import open
from datetime import date
import time

# AWS Secret Keeper Function
def get_secret(secret_name, region_name = "us-east-1"):
    # Create a Secrets Manager client
    session = boto3.session.Session()
    client = session.client(
        service_name='secretsmanager',
        region_name=region_name
    )

    # In this sample we only handle the specific exceptions for the 'GetSecretValue' API.
    # See https://docs.aws.amazon.com/secretsmanager/latest/apireference/API_GetSecretValue.html
    # We rethrow the exception by default.

    try:
        get_secret_value_response = client.get_secret_value(SecretId=secret_name)
    except ClientError as e:
        if e.response['Error']['Code'] == 'DecryptionFailureException':
            # Secrets Manager can't decrypt the protected secret text using the provided KMS key.
            # Deal with the exception here, and/or rethrow at your discretion.
            raise e
        elif e.response['Error']['Code'] == 'InternalServiceErrorException':
            # An error occurred on the server side.
            # Deal with the exception here, and/or rethrow at your discretion.
            raise e
        elif e.response['Error']['Code'] == 'InvalidParameterException':
            # You provided an invalid value for a parameter.
            # Deal with the exception here, and/or rethrow at your discretion.
            raise e
        elif e.response['Error']['Code'] == 'InvalidRequestException':
            # You provided a parameter value that is not valid for the current state of the resource.
            # Deal with the exception here, and/or rethrow at your discretion.
            raise e
        elif e.response['Error']['Code'] == 'ResourceNotFoundException':
            # We can't find the resource that you asked for.
            # Deal with the exception here, and/or rethrow at your discretion.
            raise e
    else:
        # Decrypts secret using the associated KMS CMK.
        # Depending on whether the secret is a string or binary, one of these fields will be populated.
        if 'SecretString' in get_secret_value_response:
            secret = get_secret_value_response['SecretString']
        else:
            secret = base64.b64decode(get_secret_value_response['SecretBinary'])
    return secret

# ## Connect to Strava -- Get Current Access Token
# (https://towardsdatascience.com/how-i-manage-credentials-in-python-using-aws-secrets-manager-1bd1bf5da598)
client_id = get_secret("<arn for client_id>")
client_secret = get_secret("<arn for client_secret>")
refresh_token = get_secret("<arn for refresh_token>")

# (https://github.com/franchyze923/Code_From_Tutorials/blob/master/Strava_Api/strava_api.py)
auth_url = "https://www.strava.com/oauth/token"

payload = {
    'client_id': client_id,
    'client_secret': client_secret,
    'refresh_token': refresh_token,
    'grant_type': "refresh_token",
    'f': 'json'
}

print("\nRequesting Token...\n")
res = requests.post(auth_url, data=payload, verify=False)
access_token = res.json()['access_token']
print("Access Token Received\n")
print("EXITING SAROS FIT\n")

```

Test S3 Upload and Download Python Script (test-s3.py)

```
# https://faun.pub/write-files-from-ec2-to-s3-in-aws-programmatically-716d1a4ef639

# import libraries
import boto3
from datetime import datetime

cli = boto3.client('s3')

cli.put_object(
    Bucket='sarosfit',
    Body='activities_overview.csv',
    Key='data/activities_overview_test.csv')

print("TEST OVERVIEW FILE UPDATED IN S3 BUCKET\n")

cli.download_file(
    Bucket='sarosfit',
    Key='data/activities_overview_test.csv',
    Filename='activities_overview_test.csv')

print("TEST OVERVIEW FILE DOWNLOADED FROM S3 BUCKET\n")

print("EXITING SAROS FIT\n")
```

Debug AWS Pickle Errors Python Script (test-pickle.py)

```
# import libraries
import numpy as np
import pandas as pd

import os
import sys
import subprocess
import pathlib
import requests
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

import codecs
from codecs import open
from datetime import date
import time

import boto3
import base64
from botocore.exceptions import ClientError

cli = boto3.client('s3')

# ### Load Already Downloaded Activity Details if Present
try:
    # Check to see if there is a local file
    activities_details = pd.read_pickle('activities_details.pkl')
except:
    try:
        # Check the s3 bucket to see if there is a file
        cli.download_file(
            Bucket='sarosfit',
            Key='data/activities_details.pkl',
            Filename='activities_details.pkl')

        activities_details = pd.read_pickle('activities_details.pkl')
    except:
        # Create a new empty dataframe (usually because first run)
        activities_details = pd.DataFrame()

# resolve Error: _pickle.PicklingError: Can't pickle <NA>; it's not the same object as pandas._libs.missing.NA
activities_details['latlng'] = activities_details['latlng'].astype("string")
activities_details = activities_details.replace('<NA>', '')

print(activities_details.iloc[970253])

activities_details.to_pickle('activities_details_test.pkl')

print("EXITING SAROS FIT\n")
```

APPENDIX C: PYTHON SCRIPT FOR GETTING MAX OVER TIME DURATION

Calculate Rolling Means and Maxes (rolling-maxes.py) ... (*potentially use AWS Lambda*)

```
# Rolling Maxes

# import libraries
import numpy as np
import pandas as pd

# prepare test activity dataframe
df_activity = pd.read_csv('../Golden-Cheetah/2021_02_05_15_59_07.csv') #Time Trial

df_activity = df_activity.drop(['nm', 'headwind', 'slope', 'temp', 'interval', 'lrbalance', 'lte', 'rte',
                                'lps', 'rps', 'smo2', 'thb', 'o2hb', 'hhb'], axis=1)

# add rolling means columns to dataframe
def rolling_means(df, col, durations):
    for d in durations:
        df[col+str(d)] = df[col].rolling(d, min_periods=d).mean()

durations = [1,5,15,30,60,120,300,600,1200,1800,3600,5400,7200]

rolling_means(df_activity, 'watts', durations)
rolling_means(df_activity, 'hr', durations)
rolling_means(df_activity, 'kph', durations)
rolling_means(df_activity, 'cad', durations)

# get max value in each rolling means columns
def rolling_max(df, col, durations):
    maxes = [['Field', 'Duration', 'Max', 'Start', 'End']]
    for d in durations:
        maxes.append([col, d, df[col+str(d)].max(), (df[col+str(d)].idxmax()-(d-1)), df[col+str(d)].idxmax()])
    return maxes

max_watts = rolling_max(df_activity, 'watts', durations)
max_hr = rolling_max(df_activity, 'hr', durations)
max_kph = rolling_max(df_activity, 'kph', durations)
max_cad = rolling_max(df_activity, 'cad', durations)

# print max watts, HR, speed, cadence over different time durations
print(max_watts[1:])
print(max_hr[1:])
print(max_kph[1:])
print(max_cad[1:])
```