Daten aus Social Media automatisiert herunterladen

Das Automatisieren von Downloads spielt bei der Auswertung von Daten eine große Rolle. Oftmals liegen riesige Datenmengen vor, die allerdings auf mehrere Dateien aufgeteilt wurden. Das betrifft vor allem Daten, die von diversen Diensten wie einem Fahrradverleih zur Verfügung gestellt werden. Andererseits sind Daten von sozialen Medien gefragt, anhand derer sich beispielsweise erkennen lässt, wie beliebt ein Post, Tweet oder Video ist.

Für etliche Anbieter von sozialen Medien existieren bereits Bibliotheken, die Zugriff auf das API eines sozialen Netzwerks ermöglichen [1]. Bei diversen Downloadproblemen ist allerdings die Verwendung der allgemeineren Requests-Bibliothek erforderlich [2]. Requests beschäftigt sich nämlich mit den *POST*– und *GET*-Anfragen, die an HTTP-Server übermittelt werden. Anschließend kann die gewünschte Seite oder Datei mittels geeigneter Funktion heruntergeladen werden (**Abb. 1**).

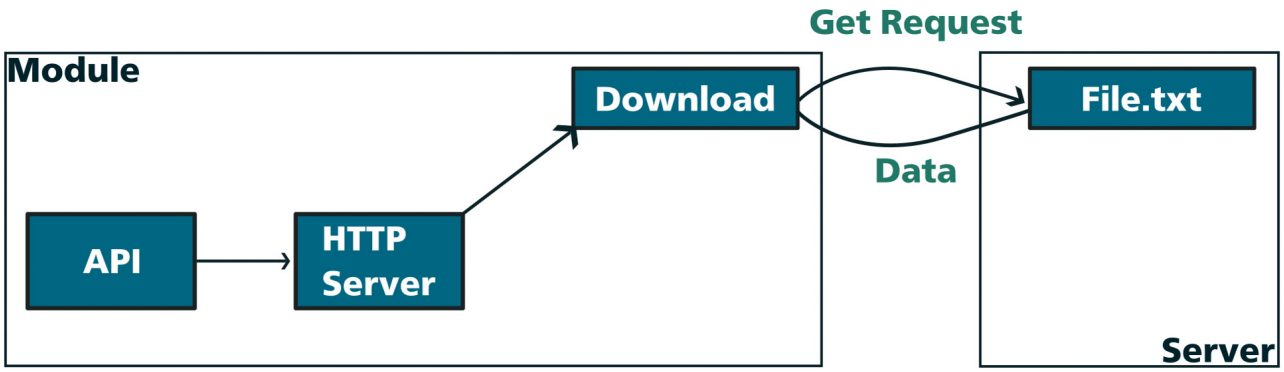

Abb. 1: Requests-Bibliothek

Alles in allem belegen diverse Statistiken, dass das Datenvolumen exponentiell wächst, womit es für Anwender unausweichlich wird, das Herunterladen von Daten beziehungsweise den Zugriff auf Daten zu automatisieren [3].

Soziale Medien

Üblicherweise platzieren Unternehmen beziehungsweise Organisationen Links zu den sozialen Netzwerken auf der eigenen Webseite, was vor allem auf die Kontaktseite zutrifft. So befinden sich die Icons der bekannten Anbieter von sozialen Medien eingebettet auf der Kontaktseite. Andererseits ist es üblich, die Icons im Footer einer x-beliebigen Seite der Organisation zu platzieren. Der dafür erforderliche HTML-Code, um das Icon eines Anbieters für soziale Medien auszugeben, könnte wie folgt aussehen:

```
<a                                        target="_blank"
href="https://www.facebook.com/GradeSaverLLC">
                <img        class="socialMedia__image"
src="/assets/footer/facebook-
beb8875d903a5a5d32d7d55667361d763ad2f0fb1c533b86afa19056eb4cbb
f8.png">
</a>
```

Sobald ein Anwender auf eines dieser Icons klickt, leitet der Browser den Anwender auf die Social-Media-Seite der Organisation weiter. Die Reichweite des Links umfasst dabei das komplette Social-Media-Icon.

Um Links aus Webseiten extrahieren zu können, eignet sich das Paket „Extract Social Media“ in der Version 0.4.0. Zusätzlich ist der Einsatz des Requests-Pakets erforderlich. Mit pip lassen sich die Pakete wie folgt installieren [4]:

```
pip install extract-social-media
pip install requests
```

Wird nun ein URL der selbstdefinierten Methode *get_links* übergeben, parst sie die Webseite nach möglichen Links und gibt diese aus (Listing 1).

Listing 1

```
import requests
```

```
from extract_social_media import find_links_tree
from html_to_etree import parse_html_bytes

def get_links(url):
  res = requests.get(url)
          tree     =     parse_html_bytes(res.content,
res.headers.get('content-type'))
  link_set = set(find_links_tree(tree))
  for s in link_set:
    print(s)

get_links('https://www.gradesaver.com/contact')
```

In der Methode *get_links* wird zunächst eine *GET*-Anfrage *requests.get* an die Webseite gesendet. Der daraufhin übermittelte Inhalt der Webseite wird in der Variable *res* gespeichert. Anschließend fischt die Methode *find_links_tree* die Links heraus und speichert sie in einem Set ab (**Abb. 2**).

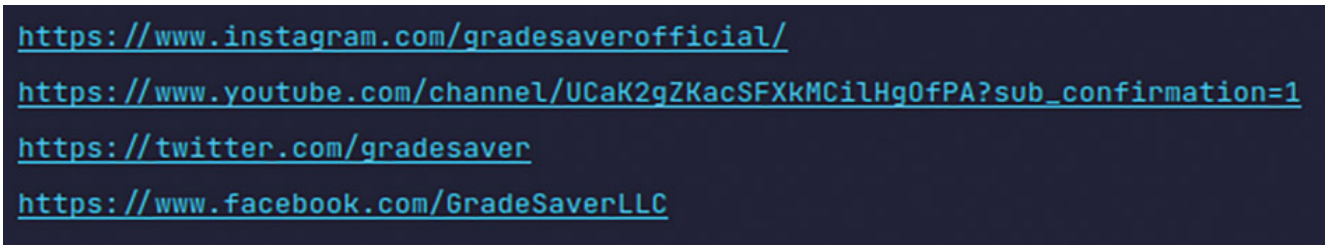

Abb. 2: Extrahierte Links einer Webseite

Dateidownload

Das Herunterladen von Dateien lässt sich unter Python in mehreren Schritten bewerkstelligen, wobei das Python-Skript davon ausgeht, dass die Links zu den Dateien bereits vorhanden sind. Vor allem die Auswertung von Dateien wird durch den automatisierten Download erleichtert, da sich heruntergeladene Text- oder CSV-Dateien im Anschluss in einen Dataframe importieren lassen. Für den Download mit Python werden die Importe aus Listing 2 gebraucht.

Listing 2

```
import numpy as np
```

```
import pandas as pd
import requests
import zipfile
import os
import glob
```

Das Python-Skript legt zunächst einen Ordner an, um die Dateien dort abzuspeichern. Der Ordner wird lediglich dann erstellt, wenn er noch nicht existiert:

```
def make_dir(folder):
  if not os.path.exists(folder):
    os.makedirs(folder)
```

Beim anschließenden Herunterladen der Dateien wird durch eine Liste mit Links *links* iteriert, wobei erneut von der Requests-Bibliothek in Version 2.28.2 Gebrauch gemacht wird (Listing 3). Um den Fortschritt des Downloads anzuzeigen, wird auf Python-Bordmittel zurückgegriffen. So wird der Index des aktuellen Links ermittelt, wobei die Anzeige zusätzlich die Gesamtzahl der Links zusammen mit der Antwort des Servers beinhaltet. In der Produktion sollte die *GET*-Anfrage *requests.get* unter Berücksichtigung eines Timeout ausgegeben werden. Dadurch wartet das Python-Skript lediglich für eine bestimmte Zeit in Sekunden auf eine Antwort des Servers, um anschließend weiterzumachen. Ansonsten besteht die Gefahr, dass sich das Programm aufhängt [5]. Mit den letzten zwei Zeilen werden die Dateien im Ordner abgespeichert (**Abb. 3**).

Listing 3

```
# downloads all zip files that are mentioned in the list:
def download(links, folder):
  size = len(links)
  for l in links:
    i = links.index(l)
    msg = '( '+str(i+1)+'/'+str(size)+' ) '+'downloading file
from: '+l
    response = requests.get(l,timeout=30)
    print(msg)
    print(response)
```

```
    print(response.elapsed)

      with open(os.path.join(folder, l.split('/')[-1]),
mode='wb') as file:
      file.write(response.content)
```

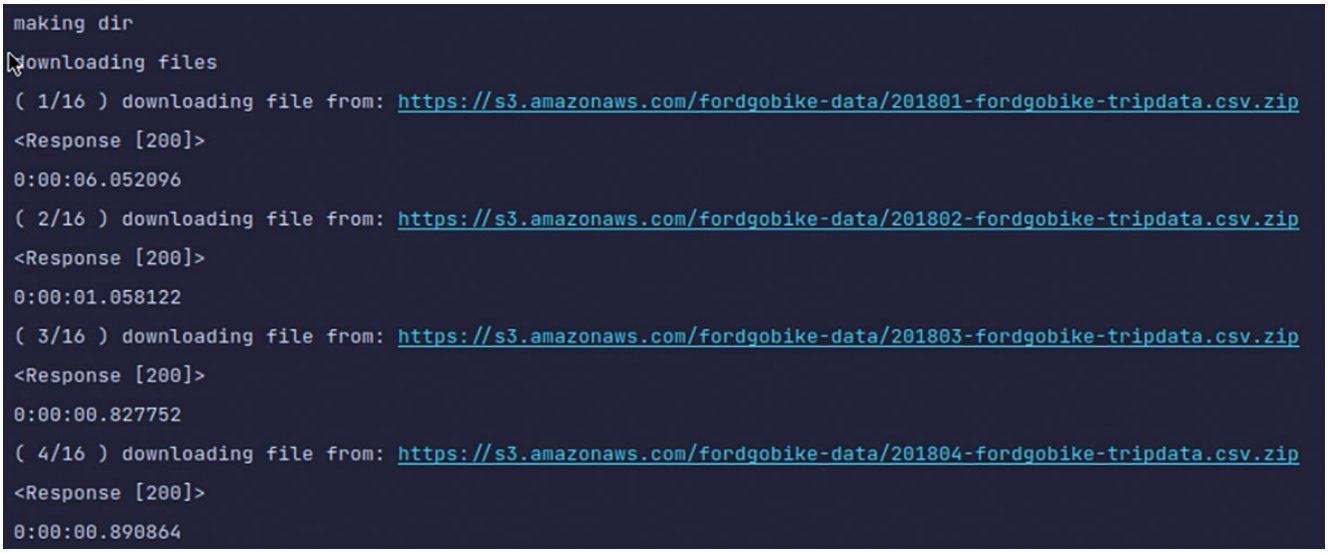

Abb. 3: Fortschrittsanzeige der Downloads

Das Programm könnte mit Listing 3 vorbei sein. Allerdings lässt sich das Programm noch weiter ausbauen, indem beispielsweise heruntergeladene ZIP-Archive automatisch entpackt werden (Listing 4). Die Methode *extract* erledigt genau das, indem sie zusätzlich die entpackten Dateien an einem beliebigen Ort auf der Festplatte ablegt.

Listing 4

```
# Extracts all contents from zip file
def extract(folder):
  all_files = glob.glob(folder + "/*.zip")
  archive = folder + '/' + working_path
  for f in all_files:
    with zipfile.ZipFile(f, 'r') as myzip:
      myzip.extractall(path=folder)
```

Handelt es sich bei den heruntergeladenen Dateien um Datenreihen, die beispielsweise als Textdatei oder im CSV-Format vorliegen, dann bietet es sich an, diese Daten in einem Dataframe zu importieren (Listing 5). So iteriert die Methode

merge durch alle Dateien vom Typ CSV. Danach werden die Datenreihen schrittweise in einen einzigen Dataframe geladen.

Listing 5

```
# merges all csv files into one dataframe
def merge(folder):
  all_files = glob.glob(folder + "/*.csv")
  li = []
  for filename in all_files:
    df = pd.read_csv(filename, index_col=None, header=0)
    li.append(df)
  fordgobike = pd.concat(li, axis=0, ignore_index=True)
  # displays the columns and their datatypes
  fordgobike.info()
```

Listing 6 beherbergt die Testdaten. So können Sie dort entnehmen, wie die Liste mit den Links definiert worden ist und wie sich die einzelnen Methoden aufrufen lassen (vgl. *test_files_download*).

Listing 6

```
folder_name = 'fordgobike'
working_path = 'archive'
# urls of zip files
urls                                                    =
['https://s3.amazonaws.com/fordgobike-data/201801-fordgobike-t
ripdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201802-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201803-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201804-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201805-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201806-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201807-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201808-fordgobike-tr
```

```
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201809-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201810-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201811-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201812-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201901-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201902-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201903-fordgobike-tr
ipdata.csv.zip',
'https://s3.amazonaws.com/fordgobike-data/201904-fordgobike-tr
ipdata.csv.zip']
def test_files_download():
  print('making dir')
  make_dir(folder_name)
  print('downloading files')
  download(urls, folder_name)
  print('extracting files')
  extract(folder_name)
  print('merging files into to dataframe')
  merge(folder_name)
```

Vimeo-Bot

Vimeo ist ein beliebter Videodienst. Die gute Nachricht ist, dass es bereits eine Bibliothek gibt, die das Vimeo-API implementiert [6]. In Version 0.4.1 lässt sich die Bibliothek Vimeo Downloader wie folgt installieren:

```
pip install vimeo-downloader
```

Zusätzlich braucht Ihr Programm den folgenden Import, um auf die Methoden und Attribute des Vimeo-API zuzugreifen:

```
from vimeo_downloader import Vimeo
```

Um mehr über ein Video zu erfahren, können Sie zunächst die Metadaten eines Videos abrufen (Listing 7). Sobald ein Objekt vom Typ *Vimeo* durch Übergabe eines Links initialisiert wird, lassen sich der Titel sowie die Zahl der Likes und Views abrufen. Zusätzlich können Sie alle Kategorien des Metaobjekts anzeigen lassen (**Abb. 4**).

Listing 7

```
def print_info(url):
  v = Vimeo(url)
  meta = v.metadata
  print('Title:', meta.title)
  print('Number of likes: ', meta.likes)
  print('Number of views: ', meta.views)
  print(meta._fields)
  return v
```

```
Title: The Science Catalyst Live Show:Chemistry Experiment
Number of likes:  1
Number of views:  1242
('id', 'title', 'description', 'url', 'upload_date', 'thumbnail_small', 'thumbnail_medium', 'thumbnail_large', 'user_id', 'user_name', 'user_url', 'user_portrait_small', 'user_portrait_medium', 'user_portrait_large', 'user_por
```

Abb. 4: Metadaten eines Vimeo-Videos

Damit Sie nun einen Vimeo-Bot generieren können, ist es lohnenswert, den Bot auf eine Liste von Links anzuwenden. So initialisiert die Schleife jedes Mal das *Vimeo*-Objekt bei Übergabe des aktuellen Links. Dabei können Sie gegebenenfalls den Titel anpassen. So lassen sich beispielsweise alle Sonderzeichen sowie Leerzeichen mit geeigneten Methoden entfernen. Den Titel für einen Link rufen Sie zunächst über die Metadaten ab, indem Sie der Methode *get_title* das zuvor initialisierte Vimeo-Objekt übergeben:

```
def get_title(v):
  meta = v.metadata
  return meta.title
```

Danach können Sie den Titel eines Videos mittels der Methode *set_filename* anpassen (Listing 8). Bei dieser Methode werden alle Sonderzeichen einschließlich der Leerzeichen entfernt. Anschließend prüft die Methode die Länge des Strings und kürzt

ihn gegebenenfalls [7].

Listing 8

```
def set_filename(s):
  max_filename = 255
  normal_string = "".join(ch for ch in s if ch.isalnum())
  stripped_s = normal_string.strip()
  split_string = stripped_s[:max_filename]
  return split_string
```

Das eigentliche Herunterladen des Videos findet in der Methode *download* unter Übermittlung des *Vimeo*-Objekts, des *Download*-Ordners sowie eines neuen Dateinamens statt. Normalerweise beherbergt der Aufruf von *vimeo.streams* eine Liste voller Streams, die für den jeweiligen Titel verfügbar sind. Allerdings handelt es sich beim letzten Listenelement um den Stream mit der höchsten Auflösung:

```
def download(vim,path,filename):
  stream = vim.streams
  best_stream = stream[-1]
        best_stream.download(download_directory=path,
filename=filename)
```

Alternativ lässt sich über die Liste verfügbarer Streams iterieren, um den gewünschten Stream herauszufiltern. Sofern Sie beispielsweise einen Stream mit einer Auflösung von 720 herunterladen wollen, können Sie wie in Listing 9 vorgehen [8].

Listing 9

```
for s in stream:
  if s.quality == '720p':
              s.download(download_directory='video',
filename=v.metadata.title)
    break
  else:
    print('quality not found')
```

Die Methode *download_all* lädt schließlich alle Videos

herunter, indem auf die zuvor erwähnten Hilfsmethoden zurückgegriffen wird (Listing 10). Abgesehen davon, beherbergt diese Methode eine Fortschrittsanzeige, die den aktuellen Download samt Downloadraten ausgibt (**Abb. 5**).

Listing 10

```
vimeo_path = '/home/Videos/vimeo'

v_1 = 'https://vimeo.com/136491689'
v_2 = 'https://vimeo.com/509738789'
v_3 = 'https://vimeo.com/546042556'
v_4 = 'https://vimeo.com/58326464'
v_5 = 'https://vimeo.com/396053468'
v_6 = 'https://vimeo.com/560619626'
v_7 = 'https://vimeo.com/4510860'

videos = [v_1, v_2, v_3, v_4, v_5, v_6]

def download_all(path,urls):
  size = len(urls)
  for url in urls:
    v = Vimeo(url)
    title = get_title(v)
    new_title = set_filename(title)
    i = urls.index(url)
    print('Downloading video ('+str(i+1)+'/'+str(size)+')')
    download(v,path,new_title)
```

Abb. 5: Download von Vimeo-Videos

YouTube-Bot

Genauso wie für Vimeo existiert eine Bibliothek, die Zugriff auf das YouTube-API ermöglicht. Aktuell liegt pytube in der Version 12.1.2 vor und lässt sich wie folgt installieren [9]:

```
pip install pytube
```

Bei Bots können Sie abgesehen von Listen Dateien einlesen und durch diese zeilenweise iterieren, um aus dem aktuellen Link ein *youtube*-Objekt zu generieren. In der Regel verfügt ein YouTube-Link über etliche Streams, sodass es empfehlenswert ist, die Auswahl weiter einzuschränken. So lässt sich angeben, dass beispielsweise lediglich Streams mit der Dateiendung *.mp4* heruntergeladen werden. Die Einstellung *progressive=true* sorgt dafür, dass lediglich Streams ausgewählt werden, die sowohl eine Audio- als auch Videospur beinhalten. Die Anweisung *streams.order_by(‚resolution‘)* wählt schließlich unter allen noch verfügbaren Streams das Video mit der höchsten Auflösung aus. Abgesehen davon beinhaltet die *filter*-Methode weitere Parameter, die in Listing 11 jedoch nicht erscheinen [10].

Mit der Downloadmethode *stream.download()* aus dem YouTube-API wird der ausgewählte Stream heruntergeladen [11]. Dabei lässt sich der Download weiter einstellen, indem auf geeignete Parameter zurückgegriffen wird. So kann zusätzlich der Pfad angegeben werden. Der *Timeout*-Parameter hingegen gibt an, wie lange auf eine Antwort vom Server gewartet wird. Daneben lässt sich die Zahl der Downloadversuche weiter eingrenzen (**Abb. 6**).

Listing 11

```
def download_yt():
    link  =  open('/home/Videos/chemistry/links_file.txt',
mode='r')
  SAVE_PATH = '/home/Videos/chemistry'
  for i in link:
    try:
      youtubeObject = YouTube(i)
      d_video = youtubeObject.streams.filter(progressive=True,
file_extension='mp4').order_by('resolution').desc().first()
      print ('Stream: '+d_video)
      print('Downloading video: ' + d_video.title)
      d_video.download(SAVE_PATH, timeout=30, max_retries=3)
    except:
      print("An error has occurred")
  print("Download is completed successfully")
```

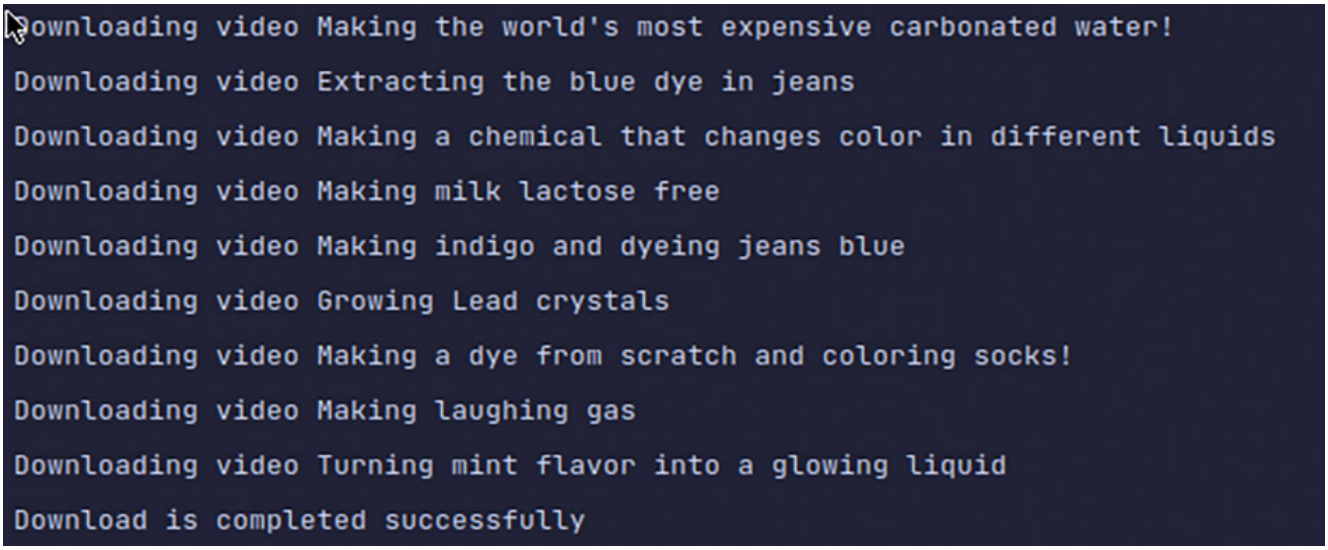

Abb. 6: Download von YouTube-Videos ohne Fortschrittsanzeige

Twitter-Bot

Twitter-Seiten erhalten eine Vielzahl an Informationen, angefangen von Tweets, Retweets, Followern, bis hin zu eingebetteten Videos etc. Dabei ermöglicht die Tweepy-Bibliothek 4.13.0 Zugriff auf das Twitter-API, sodass sich damit Etliches an Daten herunterladen lässt [12]. Um Zugriff auf das Twitter-API zu erhalten, brauchen Sie allerdings einen Entwickleraccount von Twitter. Anhand der folgenden Schritte können Sie einen Entwickleraccount bei Twitter beantragen:

- normales Benutzerkonto auf der Twitter-Seite erstellen [13]
- Benutzerkonto für Entwickler einrichten und sich für den erweiterten Zugang bewerben [14]
- Tokens, Secrets sowie Schlüssel generieren

Als Nächstes installieren Sie die Tweepy-Bibliothek:

```
pip install tweepy
```

Wenn Twitter Ihren Antrag genehmigt hat, sollten Sie über diverse Tokens, Schlüssel sowie Secrets verfügen. Sie können dafür sorgen, dass die Accountdaten griffbereit sind, indem Sie Ihre Zugangsdaten in einer Konfigurationsdatei speichern.

Diese Datei enthält Schlüssel-Wert-Paare und ist unter Python wie in **Abbildung 7** aufgebaut. Anschließend speichern Sie die Konfigurationsdatei mit der Dateiendung *.cfg* ab. Angenommen der Dateiname lautet *config.cfg*, dann lässt sich die Konfigurationsdatei wie in Listing 12 einlesen. Die Schlüssel-Wert-Paare eines Abschnitts (*section*) werden dabei in einem Dictionary abgelegt [15].

Listing 12

```
import configparser

def get_config_dict(section):
  config = configparser.RawConfigParser()
  config.read(r'config.cfg')
  if not hasattr(get_config_dict, 'config_dict'):
    get_config_dict.config_dict = dict(config.items(section))
  return get_config_dict.config_dict
```

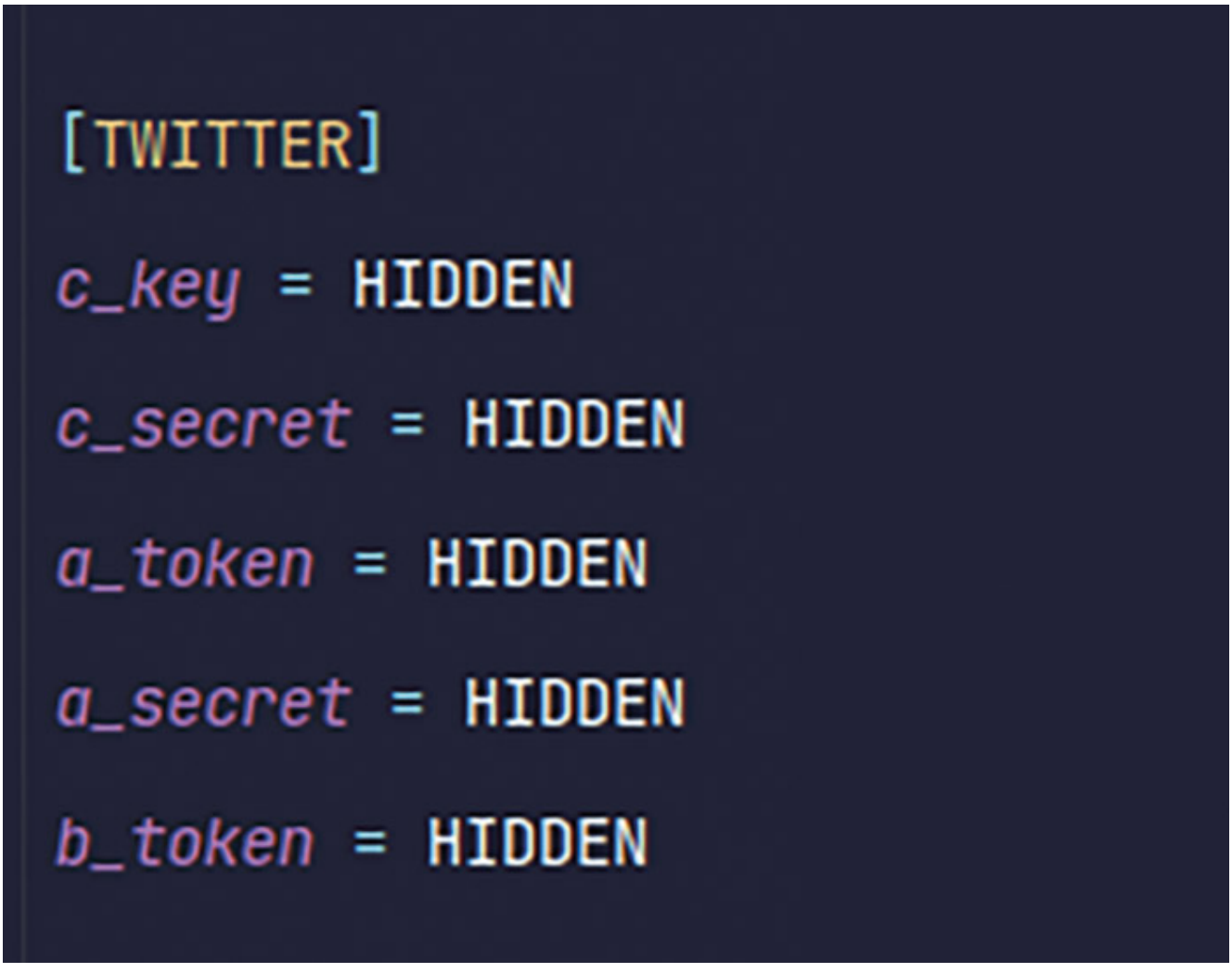

Abb. 7: Konfigurationsdatei unter Python

Um etwas bei Twitter automatisieren zu können, ist es erforderlich, sich erst einmal zu authentifizieren. So liest die Methode *create_api* zunächst die zuvor generierten Tokens, Schlüssel und Secrets unter Angabe des Abschnitts ein (Listing 13). Die eigentliche Authentifizierung beim Twitter-API erfolgt mittels der Methode *api.verify_credentials(),* wobei die zurückgegebene API-Variable für weitere Aktionen gebraucht wird [16].

Listing 13

```
import tweepy as tw
from tweepy import OAuthHandler

def create_api():
  config_details = get_config_dict('TWITTER')
  c_key = config_details['c_key']
  c_secret = config_details['c_secret']
  a_token = config_details['a_token']
  a_secret = config_details['a_secret']
  auth = OAuthHandler(c_key, c_secret)
  auth.set_access_token(a_token, a_secret)
  api = tw.API(auth, wait_on_rate_limit=True)
  try:
    api.verify_credentials()
  except Exception as e:
    print("Error creating API")
    raise e
  print("API created")
  return api
```

Bei Datenanalysen ist es relevant, Tweets zu einem bestimmten Hashtag zu analysieren. Die Suchergebnisse lassen sich dabei in einem Dataframe ablegen, um hinterher in der Lage zu sein, große Datenmengen zu analysieren. Bei dieser Suche erfolgt die Authentifizierung mit dem Bearer-Token (Inhabertoken), das ebenfalls über die Konfigurationsdatei abrufbar ist. So nimmt die Methode *get_tweets* zunächst die Authentifizierung vor

(Listing 14). Danach sucht die Methode *client.search_recent_tweets* nach Tweets der letzten sieben Tage, indem zuvor die Suchanfrage *query* definiert wird. Die Suchanfrage beinhaltet neben dem Hashtag auch die Sprache der in Frage kommenden Tweets, wobei die Suche lediglich aus *#<Suchwort>* bestehen kann. Dabei wird die maximale Zahl an Suchergebnissen auf *100* begrenzt. Außerdem lässt sich in der Suchmethode *client.search_recent_tweets* definieren, welche Felder eines Tweets gespeichert werden (**Abb. 8**). Zusätzlich lassen sich die Tweets in einem Dataframe ablegen [17].

Listing 14

```
def get_tweets(search):
  config_details = get_config_dict('TWITTER')
  token = config_details['b_token']
  client = tw.Client(bearer_token=token)
  # What to search for
  query = '#'+search+' -is:retweet lang:de'
  max_results = 100
  # We grab tweets + context annotations + create date + user name of tweeter
    tweets = client.search_recent_tweets(query=query, tweet_fields=['context_annotations', 'created_at'], user_fields=['name'], expansions='author_id', max_results=max_results)
  for tweet in tweets.data:
    print(tweet.text)
    if len(tweet.context_annotations) > 0:
      print(tweet.context_annotations)
  # Return the tweet data as a dataframe
  df2 = pd.DataFrame(tweets.data).astype(str)
  # Collect the user IDs
  df_users = pd.DataFrame(tweets.includes['users'])
```

Abb. 8: Tweets zum Hashtag #Mavropanos zusammen mit Annotationen

Abgesehen von Tweets basierend auf Hashtags ist es möglich, alle möglichen Tweets eines Twitter-Accounts herunterzuladen. Hierfür wird angenommen, dass die Tweet-ID eines Tweets zusammen mit weiteren Kategorien wie *Timestamp*, *Text* etc. in einer CSV-Datei gespeichert ist (**Abb. 9**). Dieser Twitter-Bot ist dann in der Lage, die Tweets von einem Account anhand der ID zu identifizieren und herunterzuladen.

Abb. 9: CSV-Datei zusammen mit den Tweet-IDs eines Twitter-Accounts

Die selbstdefinierte Methode *save_data_to_file* ermöglicht unter Angabe des API, dem Pfad zum Twitter-Archiv sowie dem Zielpfad für die Tweets das Herunterladen von Tweets zu automatisieren (Listing 15). Nach der Authentifizierung wird zunächst ein Twitter-Archiv namens *twitter-archive-enhanced.csv* eingelesen. Anschließend werden vom zuvor erzeugten Dataframe die Tweet-IDs extrahiert und in einer Liste abgelegt. Durch diese Liste wird anschließend iteriert, wobei pro Tweet eine JSON-Datei heruntergeladen wird. Die heruntergeladenen JSON-Dateien landen später alle in der Textdatei *tweet_json.txt*. Anhand einer selbstdefinierten Fortschrittsanzeige weiß der Benutzer sofort Bescheid, welche

Tweets sich herunterladen lassen und bei welchen Tweets der Download scheitert (**Abb. 10**). Besteht der Download hingegen aus mehreren Tausend Tweets, kommt das Rate Limit von Twitter zum Tragen und der Download wird in bestimmten Zeitintervallen für einige Minuten unterbrochen. Dadurch wird der Download einer großen Menge Tweets in die Länge gezogen.

Listing 15

```
import tweepy as tw
from tweepy import OAuthHandler
import json
from timeit import default_timer as timer
import pandas as pd

def save_data_to_file(api, sFrom, sTo):
  df = pd.read_csv(sFrom)
  tweet_ids = df.tweet_id.values
  len(tweet_ids)
  count = 0
  fails_dict = {}
  start = timer()
  # Save each tweet's returned JSON as a new line in a .txt file
  with open(sTo, 'w') as outfile:
    # This loop will likely take 20-30 minutes to run because of Twitter's rate limit
    for tweet_id in tweet_ids:
      count += 1
      print(str(count) + ": " + str(tweet_id))
      try:
                    tweet  =  api.get_status(tweet_id, tweet_mode='extended')
        print("Success")
        json.dump(tweet._json, outfile)
        outfile.write('\n')
      except tw.errors.TweepyException as e:
        print("Fail")
        fails_dict[tweet_id] = e
        pass
  end = timer()
```

```
print(end - start)
print(fails_dict)
```

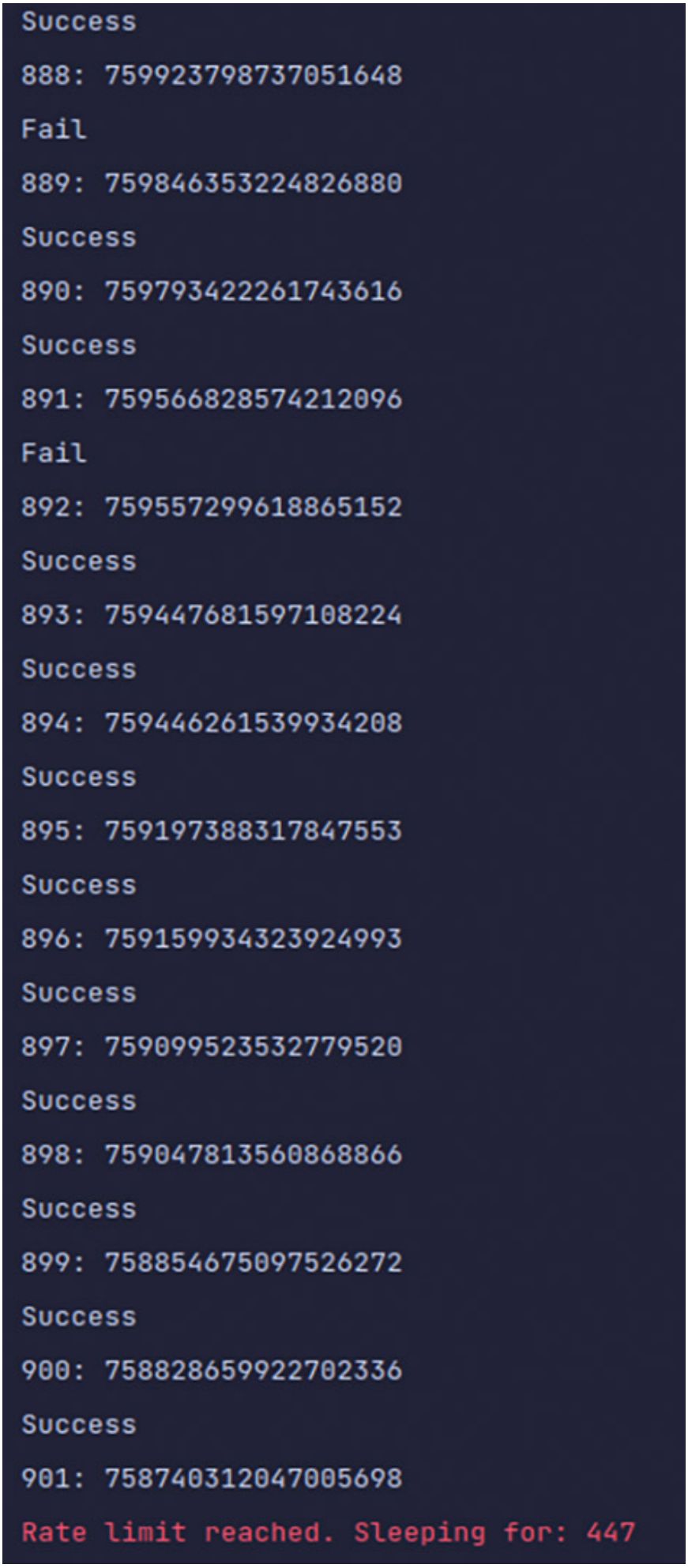

Abb. 10: Download von

Tweets

Abgesehen vom Schreiben bietet Anzela Minosi Dienstleistungen auf Legiit.com an. Dort erstellt Anzela Datenanalysen, Datenbanksoftware, Python-Skripte sowie Kommandozeilentools für den Raspberry Pi. Bevor Anzela sich selbständig gemacht hat, war sie zehn Jahre lang in den Automobil-, Bildungs- und Telekommunikationsbranchen tätig, wo sie diverse IT-Tätigkeiten ausübte: Support, Softwaretests sowie Webentwicklung. Anzela verbringt ihre Freizeit gerne an der ligurischen Küste, fährt Fahrrad und spielt Retrospiele auf dem Raspberry Pi. Sie steht für Redaktionsprojekte sowie für persönliche Beratungsgespräche zur Verfügung.

Links & Literatur

[1] PyPi: https://pypi.org/

[2] Requests: https://www.geeksforgeeks.org/get-post-requests-using-python/

[3] Datenvolumen: https://firstsiteguide.com/big-data-stats/

[4] Extract Social Media: https://pypi.org/project/extract-social-media/

[5] Requests: https://requests.readthedocs.io/en/latest/user/quickstart/

[6] Vimeo: https://pypi.org/project/vimeo-downloader/

[7] Sonderzeichen in Strings:

https://www.scaler.com/topics/remove-special-characters-from-string-python/

[8] Vimeo-Download: https://jakeroid.com/blog/how-to-download-vimeo-video-using-python/

[9] pytube: https://pytube.io/en/latest/

[10] Streams: https://pytube.io/en/latest/user/streams.html

[11] YouTube-Download: https://www.geeksforgeeks.org/pytube-python-library-download-youtube-videos/

[12] Tweepy: https://pypi.org/project/tweepy/

[13] Twitter: https://twitter.com

[14] Entwicklerportal: https://developer.twitter.com/en/support/twitter-api/developer-account

[15] Konfigurationsdatei: https://stackoverflow.com/questions/19379120/how-to-read-a-config-file-using-python

[16] Twitter-Bot: https://realpython.com/twitter-bot-python-tweepy

[17] Tweet-Suche: https://towardsdatascience.com/how-to-access-data-from-the-twitter-api-using-tweepy-python-e2d9e4d54978