

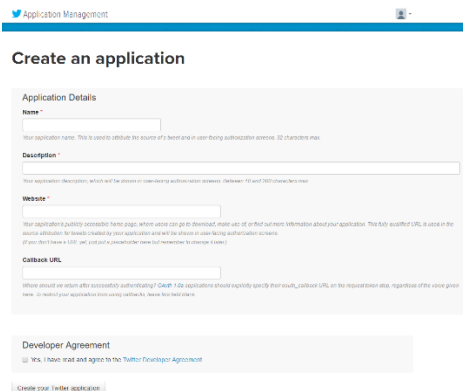
# Tutorial for data collection – for Carto master course

The following tutorial will guide you step by step to collect Twitter data and OpenStreetMap data.

## Part 1: Collecting geo-tagged twitter data via Twitter API

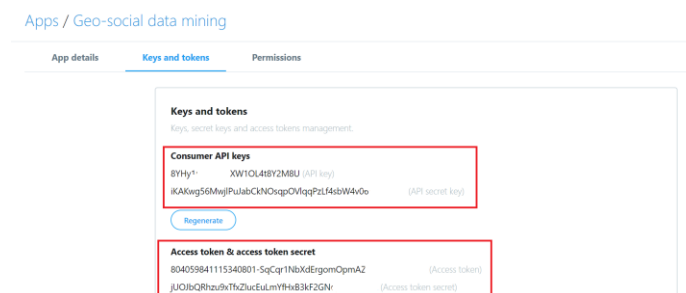
This part will guide you to collect and process twitter data. At last, you should have a “.geojson” file contains tweets which can be used as the input for D3 spatiotemporal visualization.

1. **Create a Twitter account:** go to the Twitter website and register a Twitter account. If you have a twitter account, go directly to step 2.
2. **Create a Twitter application:** thus to get the keys and tokens of your application.
  - a. Visit website <https://apps.twitter.com/>, and create your application. You should see something as follow. You can use your Github link or other personal websites to fill the field of “Website”.



The screenshot shows the 'Create an application' page on the Twitter Developer portal. It includes fields for 'Name', 'Description', 'Website', and 'Callback URL'. Below these fields is a 'Developer Agreement' section with a checkbox and a 'Create your Twitter application' button.

- b. Open “My applications” to check your keys and tokens.



The screenshot shows the 'Keys and tokens' page for a Twitter application. It displays the 'Consumer API keys' (API key and API secret key) and the 'Access token & access token secret' (Access token and Access token secret). The API key is highlighted with a red box.

3. **Install python IDE Anaconda:** Open the folder “Exercise/ Anaconda3-5.1.0-Windows-x86\_64.exe” to install Anaconda. Anaconda will help you to install some basic python packages for data processing. Please follow the **default configuration** of the installation and do not change the setting. For more details, please visit <https://www.anaconda.com/>.
4. **Install python Package Tweepy:** go to the Windows cmd command window (**note: open the cmd as administrator**), and install the Python package “Tweepy” by using the following codes:

**pip install tweepy**

5. **Python programming:**

Step 1: import the necessary packages for Twitter data collection

```
# import the python packages
import tweepy
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream
import time
import json
import random
```

Step 2: define your Twitter API keys and tokens

```
# Using your own keys and tokens to substitute the following keys and tokens
auth = OAuthHandler('8Y*****', 'iK*****')
auth.set_access_token('80*****', 'jU*****')
api = tweepy.API(auth)
```

Step 3: define a class to handle the data return from the Twitter server. By default, you will find the collected data located at “C:/Users/your\_user\_name/ Documents /Collected\_tweeters.json”.

```
# define a class of listener
# change the "Jack_collected_tweeters.json" to your favored file name

class Stdoutlistener_file_version(StreamListener):
    def on_data(self, data):
        with open('Collected_tweets.json', 'a', encoding='utf-8') as f:
            f.write(data)
    def on_error(self, status):
        print(statuses)
```

Step 4: set your own filter to collect geo-tagged tweets

```
while True:
    try:
        # define the bounding box for your geo-search
        Munich_boundingbox = [11.3855, 47.9129, 11.8520, 48.2995]
        # change the parameter parameters of "locations" to your own boundingbox below
        stream.filter(locations = Munich_boundingbox, languages = ['en', 'de'])
        break
    except Exception as e:
        # Abnormal exit: Reconnect the twitter server when
        print('A Exception happend, sleep for 60 seconds')
        nsecs = random.randint(60, 63)
        time.sleep(nsecs)
```

1. **Twitter data pre-processing:** since we need a geojson file as the input for D3 visualization. The first aim is to transfer the “.json” file to “.geojson” file. The second aim is to extract some useful information from tweets. First, we need to install some python packages, including “*geopandas*”, ‘*shapely*’ and ‘*OSMnx*’. Since ‘*OSMnx*’ relies on “*geopandas*” and ‘*shapely*’, a simple way to achieve this is to install ‘*OSMnx*’ directly by using one conda prompt as follow:

conda install -c conda-forge osmnx

Step 2: import python packages

```
import json
from shapely.geometry import Point
import geopandas as gpd
```

Step3: read the Twitter data you have downloaded

```
# read JSON file of the collected twitters
with open('Collected_tweets.json', 'r', encoding='utf-8') as f:
    lines = f.readlines()
```

Step4: define a function to extract certain fields of each tweet

```
# define a function extract certain fields
def Extract_certain_fields(tweet):
    user_name = tweet['user']['name']
    time = tweet['created_at']
    utc_ms = tweet['timestamp_ms']
    coord = tweet['coordinates']['coordinates']
    geo_point = Point(coord[0], coord[1])
    text = tweet['text']
    return user_name, time, utc_ms, geo_point, text
```

Step5: using the function you have defined in “Step 4” to process the data

```
# extracting certain fields of twitters
user_names = []
times = []
utc_mss = []
geo_points = []
texts = []
for i in range(0, len(lines), 2):
    line = lines[i]
    tweet = json.loads(line)
    if tweet['coordinates'] != None:
        user_name, time, utc_ms, point, text = Extract_certain_fields(tweet)
        user_names.append(user_name)
        times.append(time)
        utc_mss.append(utc_ms)
        geo_points.append(point)
        texts.append(text)
```

Step6: construct a GeoDataFrame to store the processed tweets

```
# construct a geodataframe to store twitter data
twitter_df = gpd.GeoDataFrame(geometry = geo_points)
twitter_df['id'] = range(len(coords))
twitter_df['username'] = user_names
twitter_df['time'] = times
twitter_df['utcms'] = utc_mss
twitter_df['text'] = texts
```

Step7: output the processed tweets into a “.geojson” file

```
# write the geojson file
# Change the file name to your own
filename = 'C:/documents/Jack_collected_twitters.geojson'
with open(filename, 'w', encoding='utf-8') as f:
    f.write(twitter_df.to_json())
```

## Part 2: Collecting map information from OpenStreetMap via python package OSMnx

In this part, we will learn how to download the city boundary, building footprints, and road networks by using OSMnx Python package. OSMnx is a Python package that lets you download spatial geometries, and construct, project, visualize and analyze street networks from OpenStreetMap's APIs. Users can download and construct walkable, drivable, or bikable urban networks with a single line of Python code, and then easily analyze and visualize them. More information from GitHub website <https://github.com/gboeing/osmnx>

OSMnx API documents: <https://osmnx.readthedocs.io/en/stable/osmnx.html>

Step 1: import the necessary package

```
import osmnx as ox
from PIL import Image
import matplotlib.pyplot as plt
# from IPython.display import Image
```

Step 2: download and output the city boundary of Munich to a ".geojson" file

```
# Downloading the administrative boundary of a city, e.g. the bounday of Munich
# Change "Munich" to your own city
gdf_city = ox.gdf_from_place('munich')

filename = 'city_boudary' + '.geojson'
with open(filename, 'w', encoding='utf-8') as f:
    f.write(gdf_city.to_json())
```

Step 3: download and output the building footprints around TUM

```
# Downloading building footprints
# The following codes will download all the buildings with 1km distance around TUM
# Since the downloading is a little bit time-cosuming , better to set a small threshold (e.g. under 3000 m)
tum = (48.1514082, 11.567568)
gdf_build = ox.buildings_from_point(point = tum, distance = 1000)

filename = 'TUM_buildings' + '.geojson'
with open(filename, 'w', encoding='utf-8') as f:
    f.write(city.to_json())
```

Step 4: download and output the street network in Munich

```
# Downloading a certain type of road network in a city, e.g. all the walkable roads in Munich
# Select a type of road network that you want to download
# ['all_private', 'all', 'bike', 'walk', 'drive', 'drive_service']
G = ox.graph_from_place('Munich', network_type='bike')
G_projected = ox.project_graph(G)
#ox.save_graph_shapefile(G_projected, filename= 'munich_road_bike', folder='C:/documents/')
ox.save_graph_shapefile(G_projected, filename= 'munich_road_bike')
```

## Part 1+: data processing

1. Change the time format into local time

Step 1: import two python packages

```
from dateutil import tz
from datetime import datetime
```

Step 2: define a function to change the utc timestamps into local time

```
# utc_timestamps: millisecond as the unit
def utc_timestamp_naive(utc_timestamps, timezone):
    local_times = []
    from_zone = tz.gettz('UTC')
    to_zone = tz.gettz(timezone)
    for utcstamp in utc_timestamps:
        utc = datetime.datetime.fromtimestamp(int(utcstamp[:-3]))
        utc = utc.replace(tzinfo=from_zone)
        local_time = utc.astimezone(to_zone)
        str_local_time = datetime.datetime.strftime(local_time, "%Y-%m-%d %H:%M:%S")
        local_times.append(str_local_time)
    return local_times
```

Step 3: call the function to transfer the time format

```
# extracting certain fields twitters
user_names = []
times = []
utc_mss = []
geo_points = []
texts = []
for i in range(0, len(lines), 2):
    line = lines[i]
    tweet = json.loads(line)
    if tweet['coordinates'] != None:
        user_name, time, utc_ms, point, text = Extract_certain_fields(tweet)
        user_names.append(user_name)
        times.append(time)
        utc_mss.append(utc_ms)
        geo_points.append(point)
        texts.append(text)

# construct a geodataframe to store twitter data
twitter_df = gpd.GeoDataFrame(geometry=geo_points)
twitter_df['id'] = range(len(coords))
twitter_df['username'] = user_names
# update the times to specify
# change to your own timezone, e.g. 'Germany/Munich'; America/New_York, ect.
times = utc_timestamp_naive(utc_timestamps=utc_mss, timezone='Germany/Munich')
twitter_df['time'] = times
twitter_df['utcms'] = utc_mss
twitter_df['text'] = texts
```

2. Text cleaning by using **regex** (Regular expression operations) module

Step 1: define a function for tweet text cleaning

