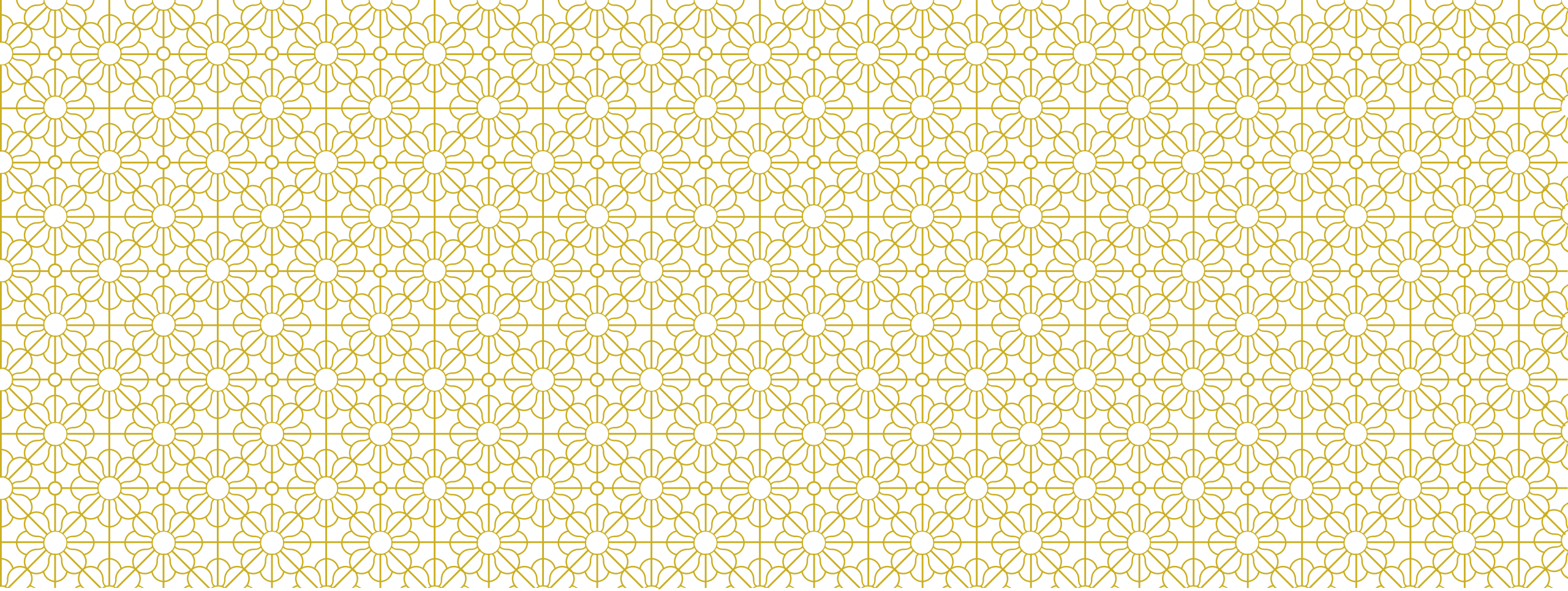


WEEK 4: PYTHON

Gates

TOPICS: PYTHON

1) Web Scraping

- HTML review
- urllib
- requests
- beautifulsoup

2) Using APIs

- JSON
- GET/POST
- urllib and requests

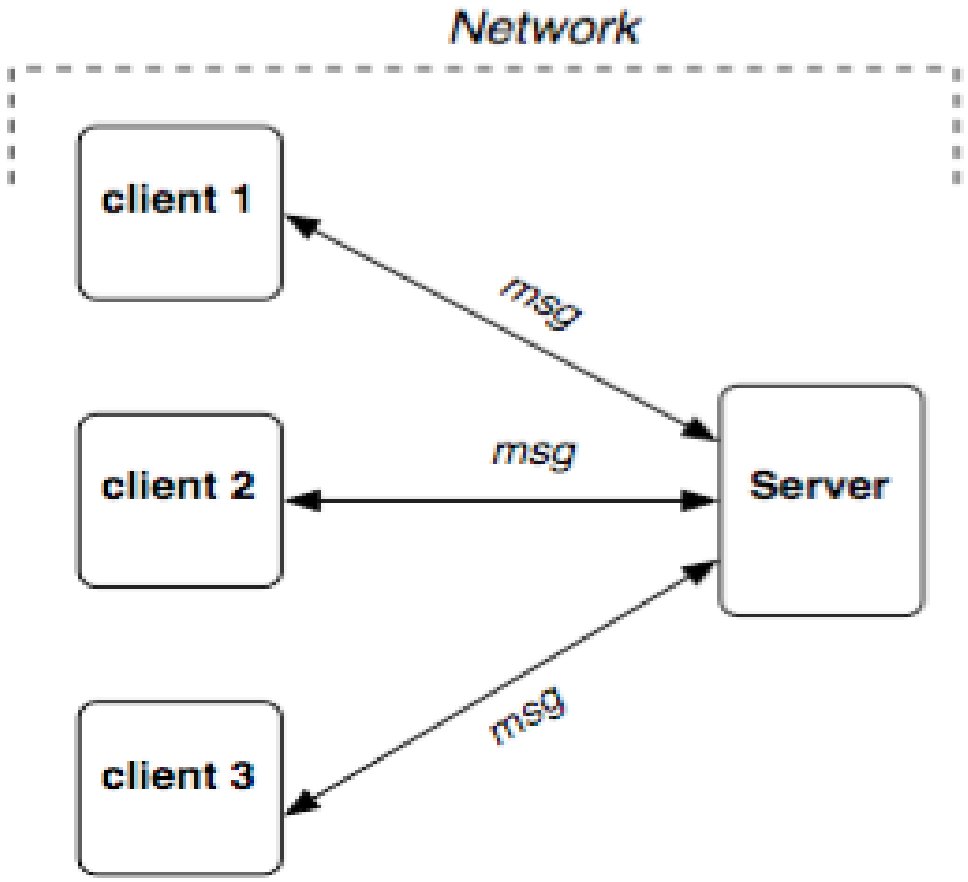
3) Twitter Mining

- and regular expressions
- tweepy
- Word Cloud visualization

CLIENT SERVER ARCHITECTURE

When we use an API, we are the “client”.

The “server” generally returns the data to us.



http://doc.sccode.org/Guides/ClientVsServer.html

HTML REMINDER

NOTE: If you do not know HTML at all – I strongly recommend learning basic HTML:

RE: https://www.w3schools.com/html/

NOTE: This above site is also great for practicing with HTML.

Hyper Text Markup Language

HTML is a language made up of **markup tags**. These tags describe how to display the content of the document/web page/

Basic HTML page: (also note that our entire class site is written in HTML/JS/CSS – you can view the source of any of the class pages to see what it looks like).

```
<html>
<body>
<h1>Data Science</h1>

<p>What fun!!</p>

</body>
</html>
```

HTML – TABLE EXAMPLE

```
<html>
<body>
<table style="width:100%">
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
</body>
</html>
```

Jill	Smith	50
Eve	Jackson	94

SCRAPING NOTES

1) Scraping means – collect or grab data from a website either directly by accessing its unprotected source or by using its approved API.

2) It is appropriate to check the site's terms and conditions.

3) Direct "Scrapers" – programs that grab data from a site source (no API) can break because site layouts change. You need to make updates often in this case.

4) Web pages are not consistent. Expect to have to clean the data a lot.

USING PYTHON TO SCRAPE DATA

Getting the html data by requesting it using HTML GET/POST URL options.

- urllib package
- requests package

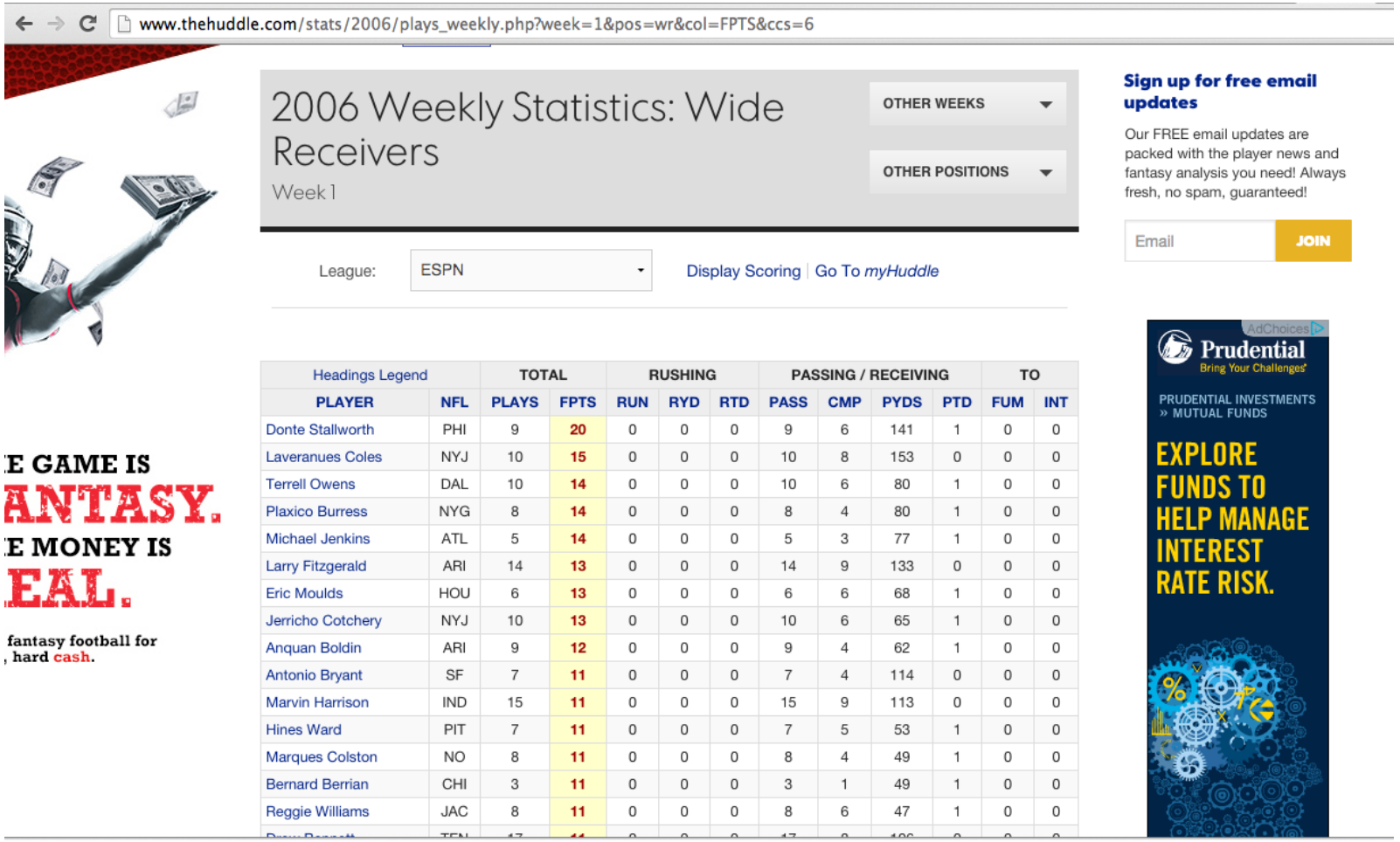
Parsing the data

- You can sometimes choose to collect the data or read the data into certain formats. The most common include:

- lxml – more common in the past.
- BeautifulSoup – very useful but does have a learning curve.
- XPath – option, but we will not cover this.
- JSON – most often used these days.

EXAMPLE 1 – HTML TABLE

1) This is a table of data that is online.
2) You can view the HTML source code for this site and can see (and grab) all of this data.
3) This is direct web scraping. It is not easy and because sites change and update often, your code will break and need updates often.



Headings Legend		TOTAL		RUSHING			PASSING / RECEIVING				TO	
PLAYER	NFL	PLAYS	FPTS	RUN	RYD	RTD	PASS	CMP	PYDS	PTD	FUM	INT
Donte Stallworth	PHI	9	20	0	0	0	9	6	141	1	0	0
Laveranues Coles	NYJ	10	15	0	0	0	10	8	153	0	0	0
Terrell Owens	DAL	10	14	0	0	0	10	6	80	1	0	0
Plaxico Burress	NYG	8	14	0	0	0	8	4	80	1	0	0
Michael Jenkins	ATL	5	14	0	0	0	5	3	77	1	0	0
Larry Fitzgerald	ARI	14	13	0	0	0	14	9	133	0	0	0
Eric Moulds	HOU	6	13	0	0	0	6	6	68	1	0	0
Jerricho Cotchery	NYJ	10	13	0	0	0	10	6	65	1	0	0
Anquan Boldin	ARI	9	12	0	0	0	9	4	62	1	0	0
Antonio Bryant	SF	7	11	0	0	0	7	4	114	0	0	0
Marvin Harrison	IND	15	11	0	0	0	15	9	113	0	0	0
Hines Ward	PIT	7	11	0	0	0	7	5	53	1	0	0
Marques Colston	NO	8	11	0	0	0	8	4	49	1	0	0
Bernard Berrian	CHI	3	11	0	0	0	3	1	49	1	0	0
Reggie Williams	JAC	8	11	0	0	0	8	6	47	1	0	0

EXAMPLE 1 – PYTHON CODE – GRABS DATA FROM THE SOURCE USING PACKAGE: URLLIB

```
#Sports Data with BS and CV
# USING urllib
#Author Gates and Singh
# Additional Ref: "Web Scraping with Python, Mitchell"

from bs4 import BeautifulSoup
from urllib.request import urlopen
import csv

def Sports():
    csvName = "TestCSV.csv"
    url="http://www.thehuddle.com/stats/2006/plays_weekly.php?week=1&pos=wr&col=FPTS&ccs=6"
    page=urlopen(url)
    soup = BeautifulSoup(page, "lxml")
    table=soup.findAll("table")[0]
    All_TR=table.findAll("tr")
    csvFile=open(csvName, "wt")
    playerwriter = csv.writer(csvFile, delimiter=',')
    playerwriter.writerow(['Player', 'Team', 'Plays', 'Fpts', 'Run', 'Ryd', 'RunTD', 'Pass', 'Cmp',
    'Pyds', 'PTD', 'Fum', 'Int'  ])
    for nextTR in All_TR:
        csvRow=[]
        for nextTD in nextTR.findAll("td"):
            csvRow.append(nextTD.text.strip())
        playerwriter.writerow(csvRow)
    csvFile.close()

Sports()
```

NOTES:

1) This code may no longer work as this Site changes often.

** However it works well now on 6/3/17 – so try it!

2) Notice that this code uses a Python packaged called "urllib". This package allows us to open a url (website) and grab all the source and data from that site.

3) The data and source will be a mess and wil need a lot of cleaning.

4) beautifulsoup is a package that helps with data viewing.

This code is on Page 18 of the Gates Python Ch 13 book portion

EXAMPLE 1 – USING PYTHON PACKAGE: **REQUESTS**

```
#Sports Data with BS and CV

#Author Gates and Singh

# Additional Ref: “Web Scraping with Python, Mitchell”

from bs4 import BeautifulSoup

import requests

import csv

def Sports():

    csvName = "TestCSV2.csv"

    url="http://www.thehuddle.com/stats/2006/plays_weekly.php?week
    =1&pos=wr&col=FPTS&ccs=6"

    page=requests.get(url)

    soup = BeautifulSoup(page.text, "lxml")
```

```
    table=soup.findAll("table")[0]
    All_TR=table.findAll("tr")
    csvFile=open(csvName, "wt")
    playerwriter = csv.writer(csvFile, delimiter=',')
    playerwriter.writerow(['Player', 'Team', 'Plays',
'Fpts', 'Run', 'Ryd', 'RunTD', 'Pass', 'Cmp',
    'Pyds', 'PTD', 'Fum', 'Int'  ])
    for nextTR in All_TR:
        csvRow=[]
        for nextTD in nextTR.findAll("td"):
            csvRow.append(nextTD.text.strip())
        playerwriter.writerow(csvRow)
    csvFile.close()

Sports()
```

NOTE: This code also works well – run it and practice.

COMPARISON: URLLIB VS REQUESTS

urllib

Standard Python 3 library

can request data across web

can handle cookies

change meta data such as headers and user-agent

requests

Standard in Anaconda – but must be installed from some Python 3 installations

can request data across web

can handle cookies

allows complete customization of headers

Great reference: Pages 178 – 182, “Web Scraping with Python”, Mitchell

MORE ON GET AND POST EXAMPLES

```
# Basic WebScraping with GET
#Author: Ami Gates – this is a basic scrape with GET
import requests
response=requests.get("http://www.mathandstatistics.com/introduction-to-statistics-video-moot")
txt = response.text
print(txt)
#Try this!
```

```
# Using urllib - type of POST
# Author Ami Gates - this uses the URL and a post ?q
import urllib
data=urllib.request.urlopen("https://pythonprogramming.
net/search/?q=Data+Analysis")
getdata=data.read()
print(getdata)
```

MORE ON POST

```
#POST with values as dict and urllib.parse
    import urllib
    myURL = "http://finance.yahoo.com/quote/AAPL?p=AAPL“    ##NOTE: test this URL to see that it works
    #The AAPL?p=AAPL is the post method used by yahoo
    values = {'p': ‘AAPL'}   #dictionary to associate the p in the post with the ticker required (AAPL)
    results=urllib.parse.urlencode(values)
    #Put data in bytes
    results=results.encode("utf-8")
    req = urllib.request.Request(myURL, results)
    #visit the site with the values
    resp = urllib.request.urlopen(req)
    resp_results=resp.read()
    print(resp_results) # notice that this prints the entire site. You would need to use regular expressions from here
```

TOPICS: PYTHON

1) Web Scraping
 - HTML review
 - urllib
 - requests
- beautifulsoup
2) **Using APIs**
 - JSON
 - GET/POST
 - urllib and requests
3) Twitter Mining
 - and regular expressions
 - tweepy
 - Word Cloud visualization

APIs

1) An API is an Application Programming Interface.

2) Many websites that maintain and collect data offer an API – such as Twitter, AirNow, Wikipedia, etc.

3) APIs allow you to access (using Python or another programming language) the data from the site.

4) There are always limitations – such as how much data you can gather at a time and how you can gather the data.

5) Most often, you will have to create a login (register) and get KEYS and PASSCODES to access the data.

6) We will discuss all the steps of using APIs including the registration, the KEYS, the packages in Python, and the HTML (GET/POST).

API ENDPOINTS

API-defined URL or location where particular data is stored.

You can retrieve data from a RESTful API by making a URL request to an endpoint using specific query parameters.

As noted, you may also need KEY information.

Each API is a little different, but the concepts are the same.

API KEYS

Most Web-based APIs require a key.

The key is unique and identifies each user.

The key is associated with each request.

API USAGE EXAMPLE

```
#Simple example 1
import requests
import pprint
query_params = { 'apikey': ‘xxx’,
                 'phrase': 'privacy’ }
endpoint = 'http://capitolwords.org/api/text.json'
response = requests.get( endpoint, params=query_params)
data = response.json
pprint.pprint(data)
```

XML AND JSON

Return values from the APIs are in XML format or JSON format – you choose.

XML looks like html markup

```
<student>
 <firstName>Jane</firstName>
 <lastName>Mouse</lastName>
</student>
```

JSON is not tagged the same way.
JSON looks like this – very dictionary

```
"students": [
        {
        "firstName": "Jane",
        "lastName": "Mouse"
        } ,
        {
        "firstName": "Jack",
        "lastName": "Dog"
        }
]
```

AIRNOW EXAMPLE CODE: API

1) The website for AirNow offers an API interface.

2) To use the API, you must register and get a KEY, etc.

3) The following three slides show Python code that accesses data from the AirNow site.

4) There are many ways to improve this code so that the results are placed into a dataframe rather than a text file, etc.

5) Review and run the code (get your own AirNow Key).

6) The Assignment will be based in part on using the AirNow API, etc.

AIRNOW EXAMPLE: PART 1

```
import urllib
from urllib.request import urlopen


def main():
    FileName="AirNowExample_2017.txt"
    #Create the File
    File=open(FileName, "w", encoding="utf-8")
    File.close()
    ziplist=["20007", "90210"]
    datelist=["2012-01-01","2013-01-01","2014-01-01","2015-01-01","2016-01-01", "2017-01-01"]

    GetAirNowData(FileName, ziplist, datelist)
```

AIRNOW EXAMPLE: PART 2

```
def GetAirNowData(FileName, ziplist,datelist):

##http://www.airnowapi.org/aq/forecast/zipCode/?format=text/csv&zipCode=20007&date=2016-09-01&distance=25&API_KEY=xxxxxxxxxx

##http://www.airnowapi.org/aq/observation/zipCode/historical/?format=text/csv&zipCode=20002&date=2014-09-03T00-0000&distance=25&API_KEY=xxxxxxxxxxxxxxx

    File=open(FileName, "a", encoding="utf-8")

    #Default Values

    baseURL="http://www.airnowapi.org/aq/"

    #date='2016-09-01'

    miles=5

    #zipcode="20007"

    ##RE: https://docs.airnowapi.org/forecastsbyzip/query

    ## RE: http://www.airnowapi.org/aq/forecast/zipCode/?format=text/

    ####csv&zipCode=20002&date=2017-02-13&distance=25&API_KEY=xxxxxxxxxxx
```

AIRNOW EXAMPLE: PART 3

```
    for zip in ziplist:
        for date in datelist:
            zipURL=baseURL + "forecast/zipCode/?" + urllib.parse.urlencode({
            'format': "text/csv",
            'zipCode':zip,
            'date':date,
            'distance':miles,
            'API_KEY':'D9AA91E7-070D-4221-867C-XXXXXXXXXXXXXXXXXXXXX'
            })
            #print(zipURL)
            response=urlopen(zipURL).read().decode('utf-8')
            File.write(response)
    File.close()


main()
```

EXAMPLE 2: AIRNOW AND URLLIB VS. REQUESTS

This link to Python code includes examples for:

- urllib versus requests
- JSON
- using APIs
- GET and POST

The code is HERE:

http://drgates.georgetown.domains/ANLY500/AirNowPythonCode.txt

TOPICS: PYTHON

1) Web Scraping

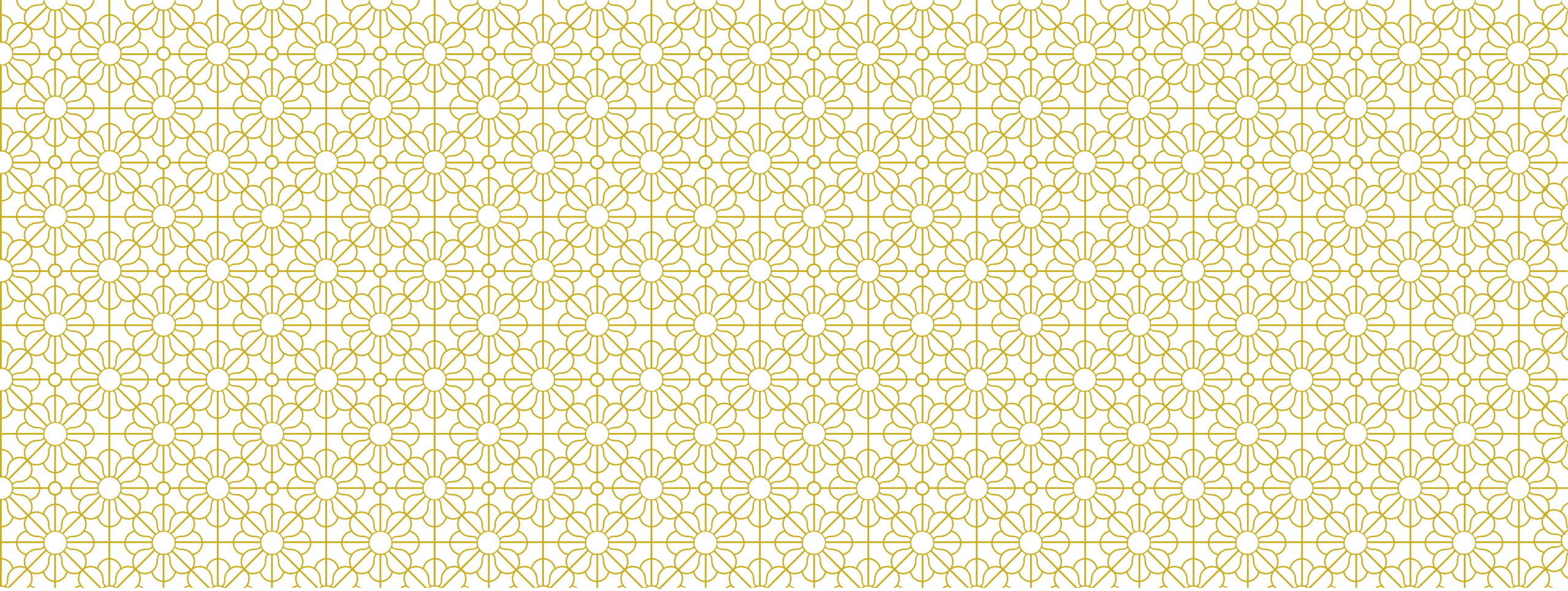
- HTML review
- urllib
- requests
- beautifulsoup

2) Using APIs

- JSON
- GET/POST
- urllib and requests

3) Twitter Mining

- and regular expressions – you can find examples of this in the twitter code I am sharing
- tweepy
- Word Cloud visualization



MINING AND VISUALIZING TWITTER WITH PYTHON

Gates

EXCELLENT RESOURCES AND REFERENCES

https://marcobonzanini.com/2015/03/02/mining-twitter-data-with-python-part-1/

https://dev.twitter.com/overview/terms/agreement-and-policy

https://dev.twitter.com/rest/public/rate-limiting

https://dev.twitter.com/rest/public/rate-limits

https://dev.twitter.com/streaming/overview

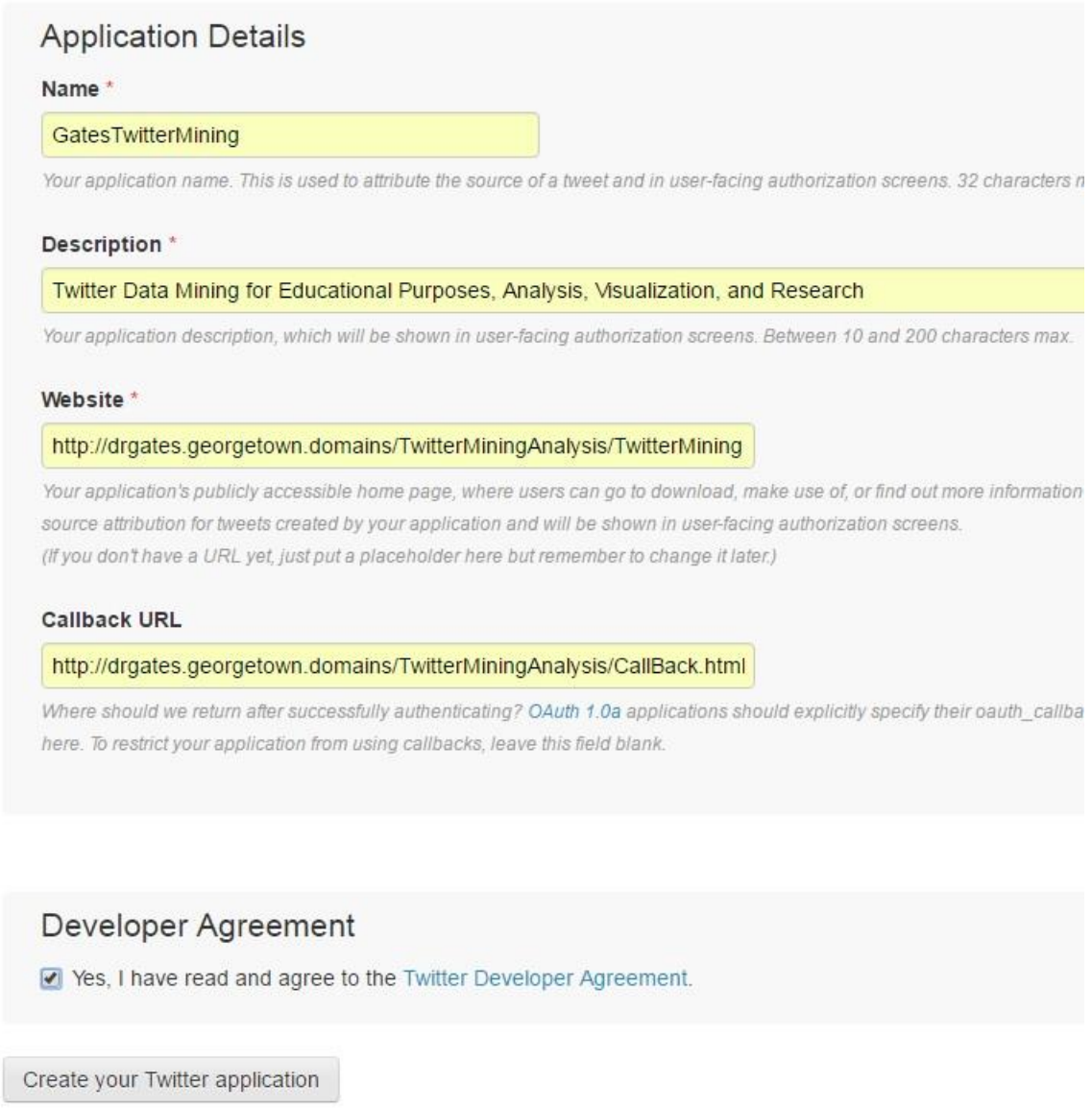
CREATE AN APP TO USE

1) Go to: https://apps.twitter.com/

2) Sign in or create a Twitter Account

3) Go back to https://apps.twitter.com/

and register a new application. “ Create New App”



REVIEW THE APPLICATION

1) Look at the “Details”

** Get and SAVE IN A SAFE PLACE the KEY and info:

Access level Read and write (modify app permissions)

Consumer Key (API Key)mnDC09Zxxxxxxxxxxxxxxxxx (manage keys and access tokens)

Callback URLhttp://drgates.georgetown.domains/TwitterMiningAnalysis/CallBack.html

Callback URL Locked: No

Sign in with Twitter: Yes

App-only authenticationhttps://api.twitter.com/oauth2/token

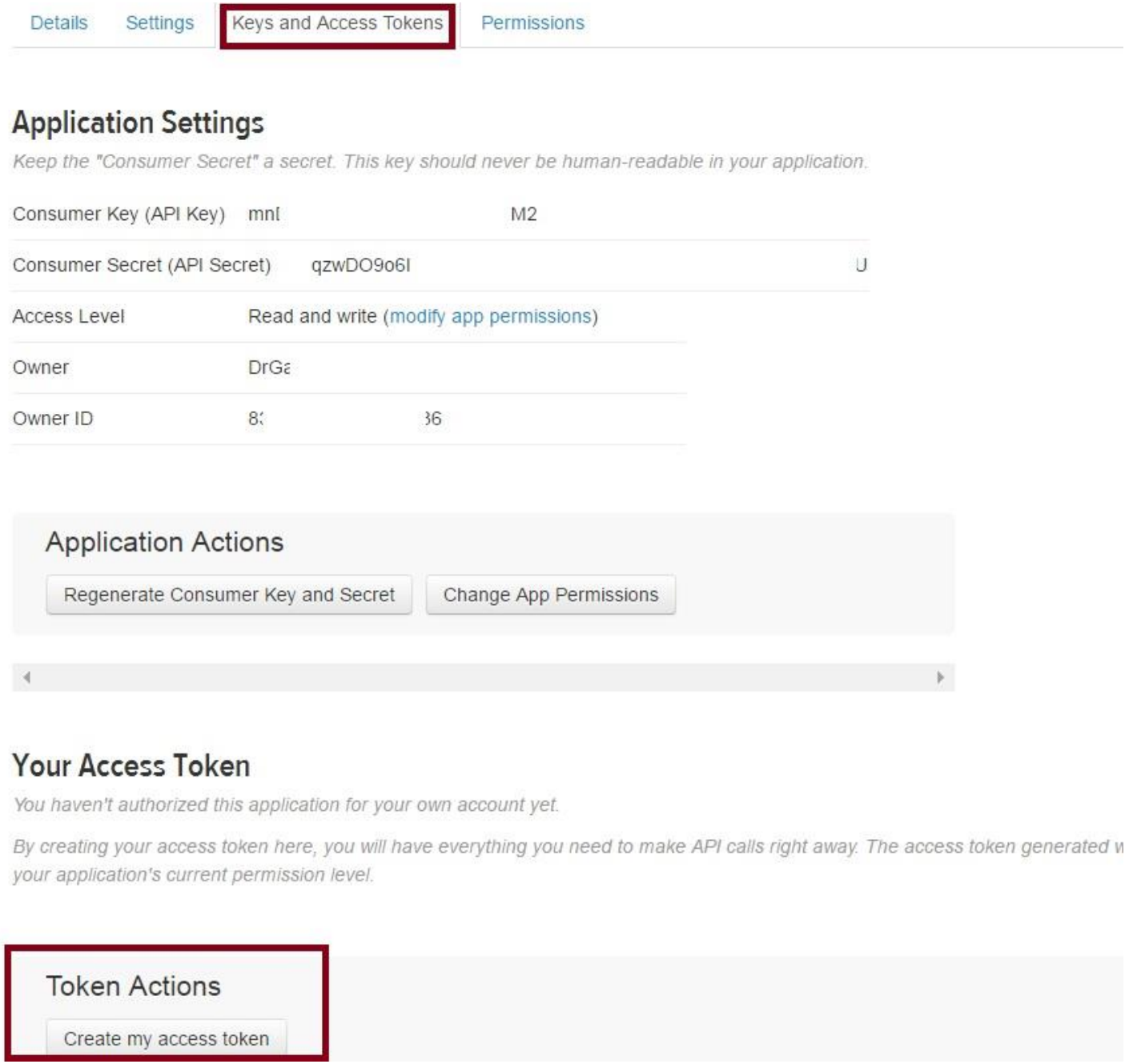
Request token URLhttps://api.twitter.com/oauth/request_token

Authorize URLhttps://api.twitter.com/oauth/authorize

Access token URLhttps://api.twitter.com/oauth/access_token

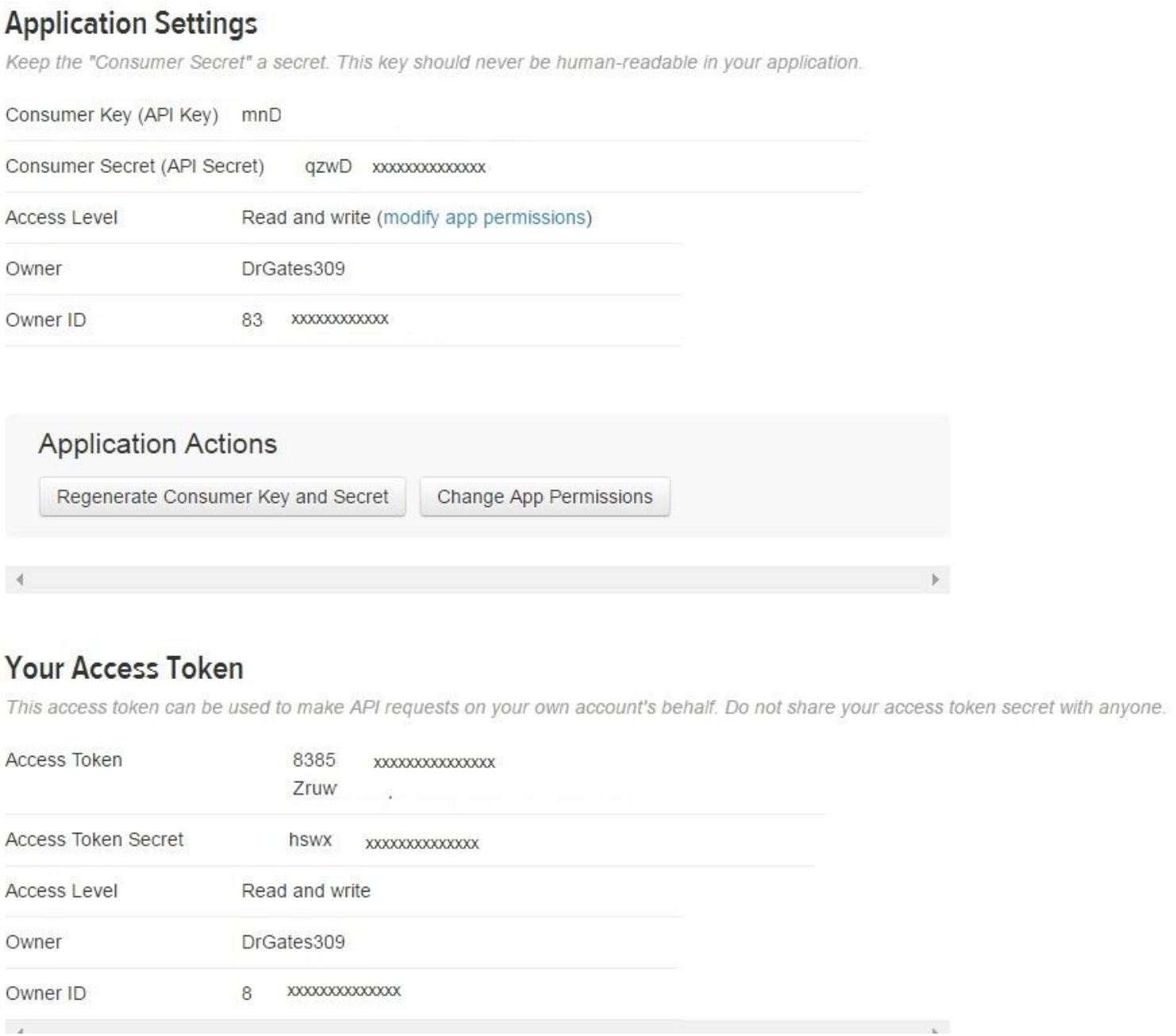
CLICK MANAGE KEYS

Choose to **Create my access token**



NEXT, YOU WILL SEE THIS

Now you have the access codes and registration ready.



Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	mnD
Consumer Secret (API Secret)	qzwD xxxxxxxxxxxxx
Access Level	Read and write (modify app permissions)
Owner	DrGates309
Owner ID	83 xxxxxxxxxxx

Application Actions

Regenerate Consumer Key and Secret | Change App Permissions

Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token	8385 xxxxxxxxxxxxx Zruw .
Access Token Secret	hswx xxxxxxxxxxxxx
Access Level	Read and write
Owner	DrGates309
Owner ID	8 xxxxxxxxxxxxx

LOG OUT AND LOG BACK IN AND YOU WILL SEE YOUR VERSION OF:

NEXT STEPS

Like most API services, Twitter has Rules and imitations:

Here are rate limits in the use of the Twitter API, as well as limitations in case you want to provide a downloadable data-set, see:

https://dev.twitter.com/overview/terms/agreement-and-policy

https://dev.twitter.com/rest/public/rate-limiting

REST API REMINDER

RE: https://dev.twitter.com/rest/public

“The REST APIs provide programmatic access to read and write Twitter data. “

You can: Create a new Tweet, read a user profile, follower data, and more.

The REST API identifies Twitter applications and users using OAuth; responses that are in JSON format.

If your intention is to monitor or process Tweets in real-time, consider using the Streaming API instead.”

WHAT IS "REST"

REST: Representational State Transfer.

- An architecture style that is a stateless, client-server based, and cacheable communications protocol that generally works with HTTP.
- A RESTful API refers to an application program interface (API) that uses HTTP and GET, PUT, POST, and DELETE; most commonly GET and POST.

Tutorial: http://www.restapitutorial.com/lessons/whatisrest.html

REST:

- Six Constraints: uniform interface, stateless, client-server, cacheable, layered, code on demand.

- Resourced-based (like user or address or thing) (rather than action based in SOAP-RPC such as get data).

A UNIFORM INTERFACE

- The UI (uniform interface) defines the interface between the client and the server.

  For example, **HTTP is the protocol, URI are the resource names** and the **HTTP verbs are the actions taken** on the resources.

- (Note: RESTful does not have to use HTTP, but most often does)

  HTTP verbs: GET, POST, PUT, DELETE)

- The **Response** when using HTTP occurs in the body or status of the HTTP.

AVAILABLE PYTHON CLIENTS: FYI

python-twitter *maintained by @bear* — this library provides a pure Python interface for the Twitter API (documentation)

tweepy *maintained by @applepie & more* — a Python wrapper for the Twitter API (documentation) (examples)

TweetPony *by @Mezgrman* — A Python library aimed at simplicity and flexibility.

Python Twitter Tools *by @sixohsix* — An extensive Python library for interfacing to the Twitter REST and streaming APIs (v1.0 and v1.1). Also features a command line Twitter client. Supports Python 2.6, 2.7, and 3.3+. (documentation)

twitter-gobject *by @tchx84* — Allows you to access Twitter's 1.1 REST API via a set of GObject based objects for easy integration with your GLib2 based code. (examples)

TwitterSearch *by @crw_koepp* — Python-based interface to the 1.1 Search API.

twython *by @ryanmcgrath* — Actively maintained, pure Python wrapper for the Twitter API. Supports both normal and streaming Twitter APIs. Supports all v1.1 endpoints, including dynamic functions so users can make use of endpoints not yet in the library. (docs)

TwitterAPI *by @boxnumber03* — A REST and Streaming API wrapper that supports python 2.x and python 3.x, TwitterAPI also includes iterators for both API's that are useful for processing streaming results as well as paged results.

Birdy *by @sect2k* — "a super awesome Twitter API client for Python"

TWEEPY

Tweepy is one of many Python-Twitter client options.

Install Tweepy on Python3:

If using Windows, on any command line type:

anaconda search –t conda tweepy

Use 3.5.0 or later as 3.4.0 does not work well with Python 3.

I used: conda **install** -c lebpride **tweepy**=3.5.0

CODE TO COLLECT TWEETS

The next few slides will offer a code option (one of many) that will connect and “listen” for a fixed number of tweets (defined by the User) on a given #YOURTOPIC choice.

CODE PART 1 (USING TWEEPY)

```
import tweepy
from tweepy import OAuthHandler
import json
from tweepy import Stream
from tweepy.streaming import StreamListener
import sys
```

CODE PART 2

```
consumer_key = 'mnDC09ZgIqUDzxxxxxxxxxxxxxxxxxxxx'
consumer_secret = 'qzwDO9o6Im6xxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
access_token = '8385586xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
access_secret = 'hswxbxERmUPxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'

auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)
api = tweepy.API(auth)
```

```
42
43 class Listener(StreamListener):
44     print("In Listener...")
45     tweet_number=0
46     #__init__ runs as soon as an instance of the class is created
47     def __init__(self, max_tweets):
48         self.max_tweets=max_tweets
49         print(self.max_tweets)
50     #on_data() is a function os StreamListener as is on_error and on_status
51     def on_data(self, data):
52         self.tweet_number+=1
53         print("In on_data", self.tweet_number)
54         try:
55             print("In on_data in try")
56             tweet=json.loads(data)
57             with open("TwitterStock.json", 'a') as f:
58                 json.dump(tweet, f)
59         except BaseException:
60             print("NOPE")
61             pass
62         if self.tweet_number>=self.max_tweets:
63             sys.exit('Limit of '+str(self.max_tweets)+' tweets reached.')
64     #method for on_error()
65     def on_error(self, status):
66         print("ERROR")
67         if(status==420):
68             print("Error ", status, "rate limited")
69             return False
70
```

THE CALLS IN THE CODE

twitter_stream = Stream(auth, Listener(**5**))

twitter_stream.filter(track=['**#stockmarket**'])

Notice that here I am looking for tweets on #stockmarket.

I am also limiting the number of tweets I collect to 5

The following slide is a Tweet in JSON

{"timestamp_ms": "1488997991805", "lang": "en", "id_str": "839544543477448710", "text": "Naysayers keep stating the #StockMarket is set to Crash in 2017? they said the same in 2016 https://t.co/Gg12IGTeqv", "place": null, "geo": null, "in_reply_to_screen_name": null, "filter_level": "low", "in_reply_to_status_id_str": null, "is_quote_status": false, "in_reply_to_user_id": null, "favorited": false, "in_reply_to_status_id": null, "entities": {"symbols": [], "urls": [{"url": "https://t.co/Gg12IGTeqv", "expanded_url": "http://tacticalinvestor.com/stock-market-crash-2017-video-reality-or-all-hype/", "display_url": "tacticalinvestor.com/stock-market-c\u2026", "indices": [92, 115]}], "user_mentions": [], "hashtags": [{"text": "StockMarket", "indices": [27, 39]}]}, "id": 839544543477448710, "favorite_count": 0, "created_at": "Wed Mar 08 18:33:11 +0000 2017", "in_reply_to_user_id_str": null, "contributors": null, "truncated": false, "retweeted": false, ……

UNDERSTANDING TWEET DATA: THE KEY ATTRIBUTES

1) text - this is the text in the tweet

2) created_at – this is the date of the creation

3) favorite_count, retweet_count: these are the number of favorites and retweets

4) favorite, retweeted: Boolean

5) lang – languages such as en (English), etc.

6) id – tweet ID

7) place, coordinates, and geo – location information

8) user – the User's profile

9) in_reply_to_user_id – the user ID if the tweet is a reply to a specific User

10) in_reply_to_status – status ID if the tweet is a reply to a specific status

SEARCHING FOR INFORMATION

- There is a considerable amount of information, mis-information, and absolute nonsense in tweet text.
- There are also many goals for mining tweets, such as looking at the number of retweets or favorites, or locations for given tweet types, etc. etc….
- One option is to mine the text of the tweets.
- The first step in doing this is to separate the "text" into a **list (bag of WORDS)** – this is often called parsing or tokenizing.

TOKENIZING THE TEXT OF A TWEET

OPTIONS

1) You can always write your own Tokenizer using regular expressions. If you have never done this, I strongly recommend it.

2) Python has a package called **nltk** which has a tokenize option.

```
from nltk.tokenize import word_tokenize
```

```
from nltk.tokenize import word_tokenize
tweet2 = "Pretend this is a tweet"

BagOfWords=word_tokenize(tweet2)
print(BagOfWords)
print(type(BagOfWords))
for eachword in BagOfWords:
    print(eachword)
```

RESULT:

```
['Pretend', 'this', 'is', 'a', 'tweet']
<class 'list'>
Pretend
this
is
a
tweet
```

ALL THE CODE

For these examples, I have created three python programs.

1) **TwitterMining.py** (I have shared a .txt version of this without my KEYS)

This one uses Listener to grab the tweets from twitter and store the results in a file.

2) **TweepyJSONReader.py**

This parses and tokenizes the text file of tweets created above.

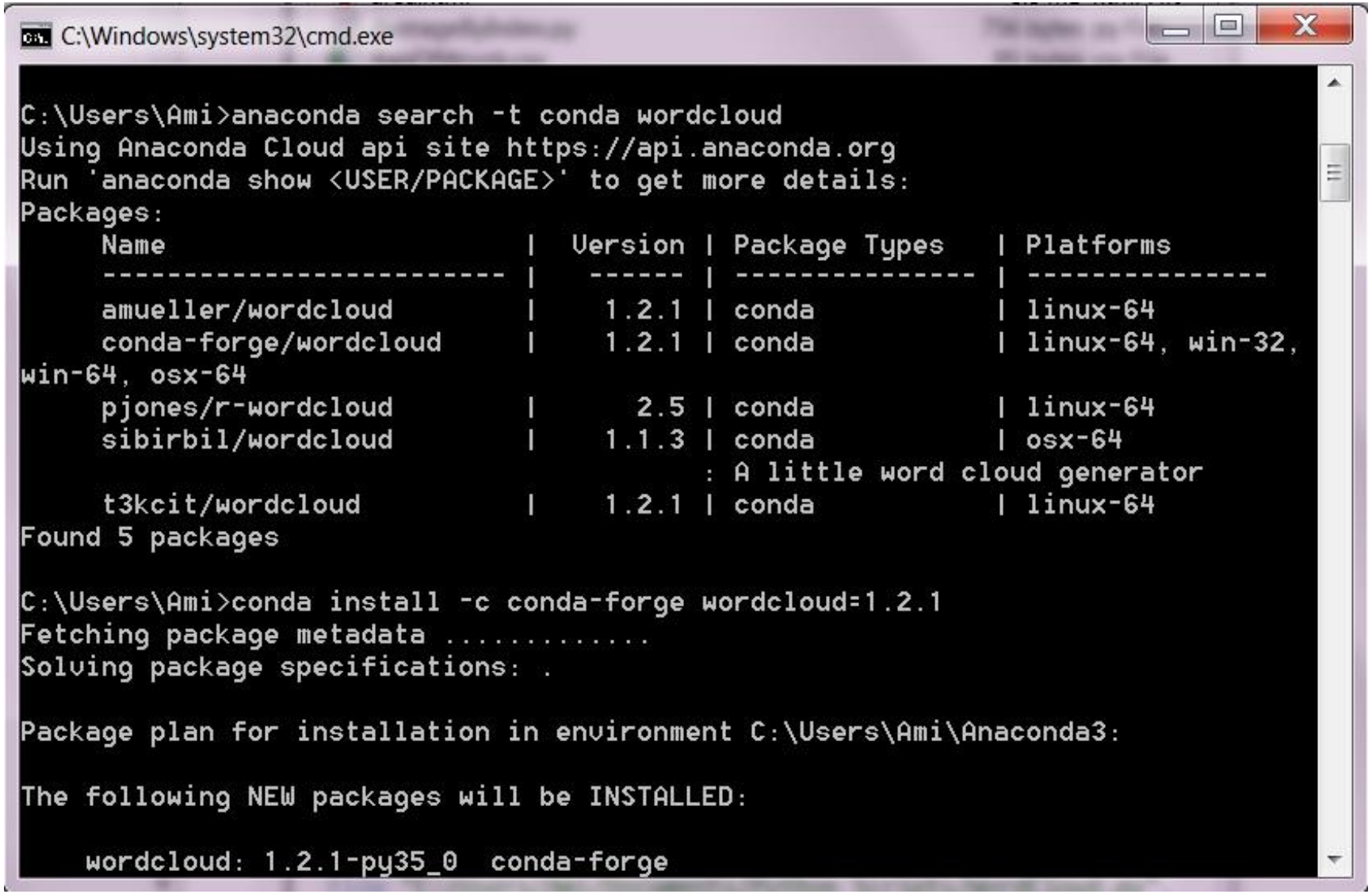
3) **WordCloud.py**

This builds a word cloud from some of the results.

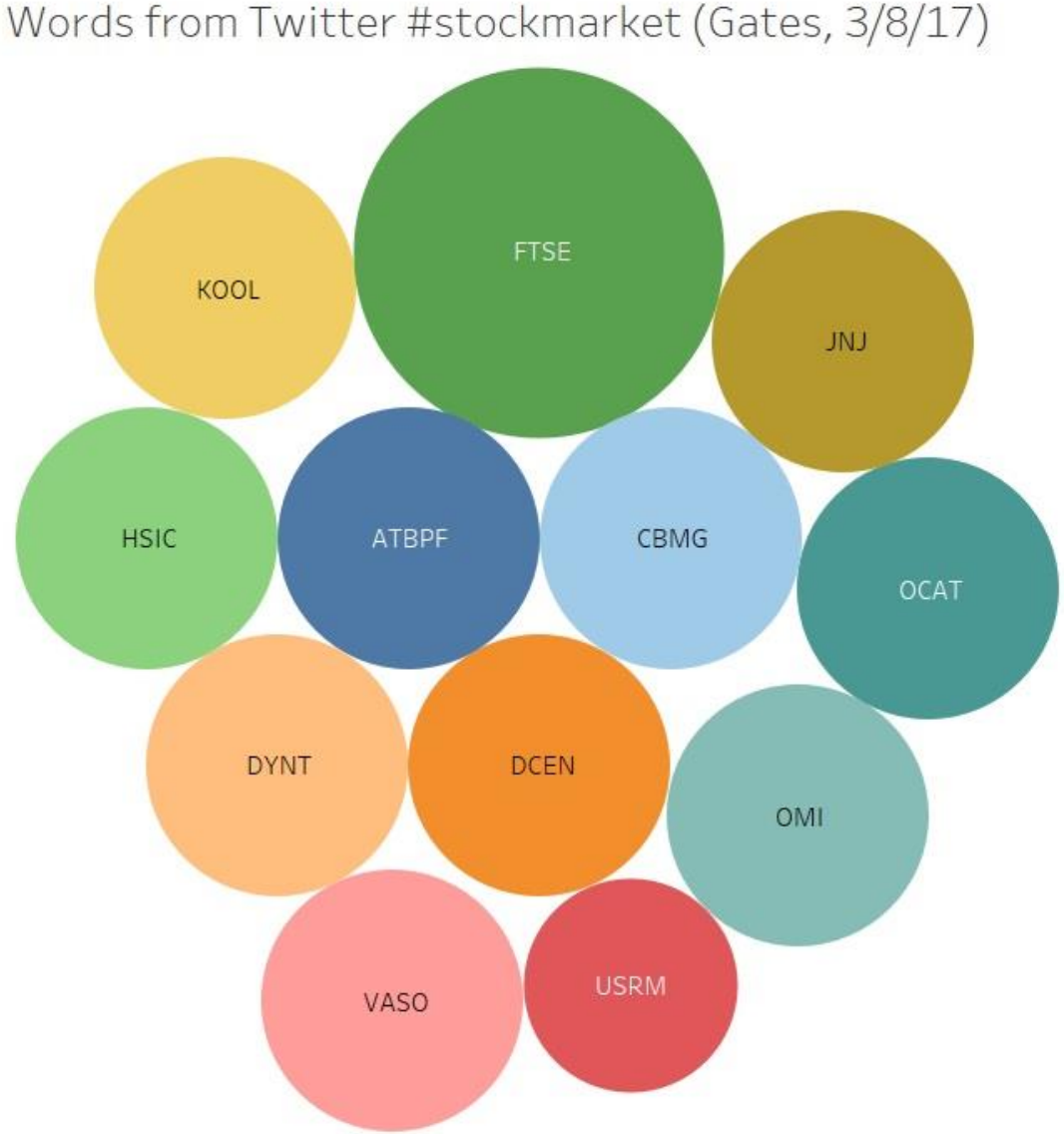
ALL CODE:

http://drgates.georgetown.domains/ANLY500/twitter/

INSTALL WORDCLOUD

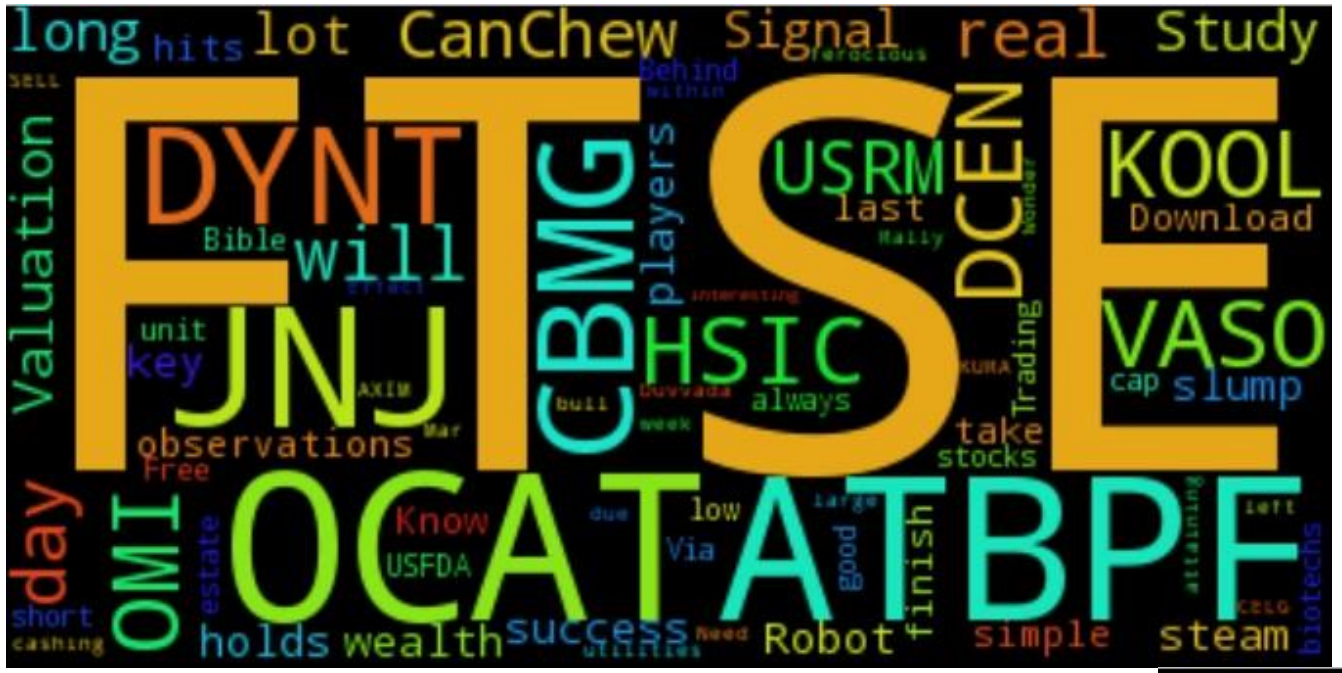


BLOB CHART TABLEAU



WordCloud Python3

REMOVING DUMB WORDS



FYI

1) FTSE: Financial Times Stock Exchanges 100 Index

2) OCAT: Ocata Therapeutics Stock

3) ATBPF: Antibe Therapeutics, Inc. Stock (penny stock)

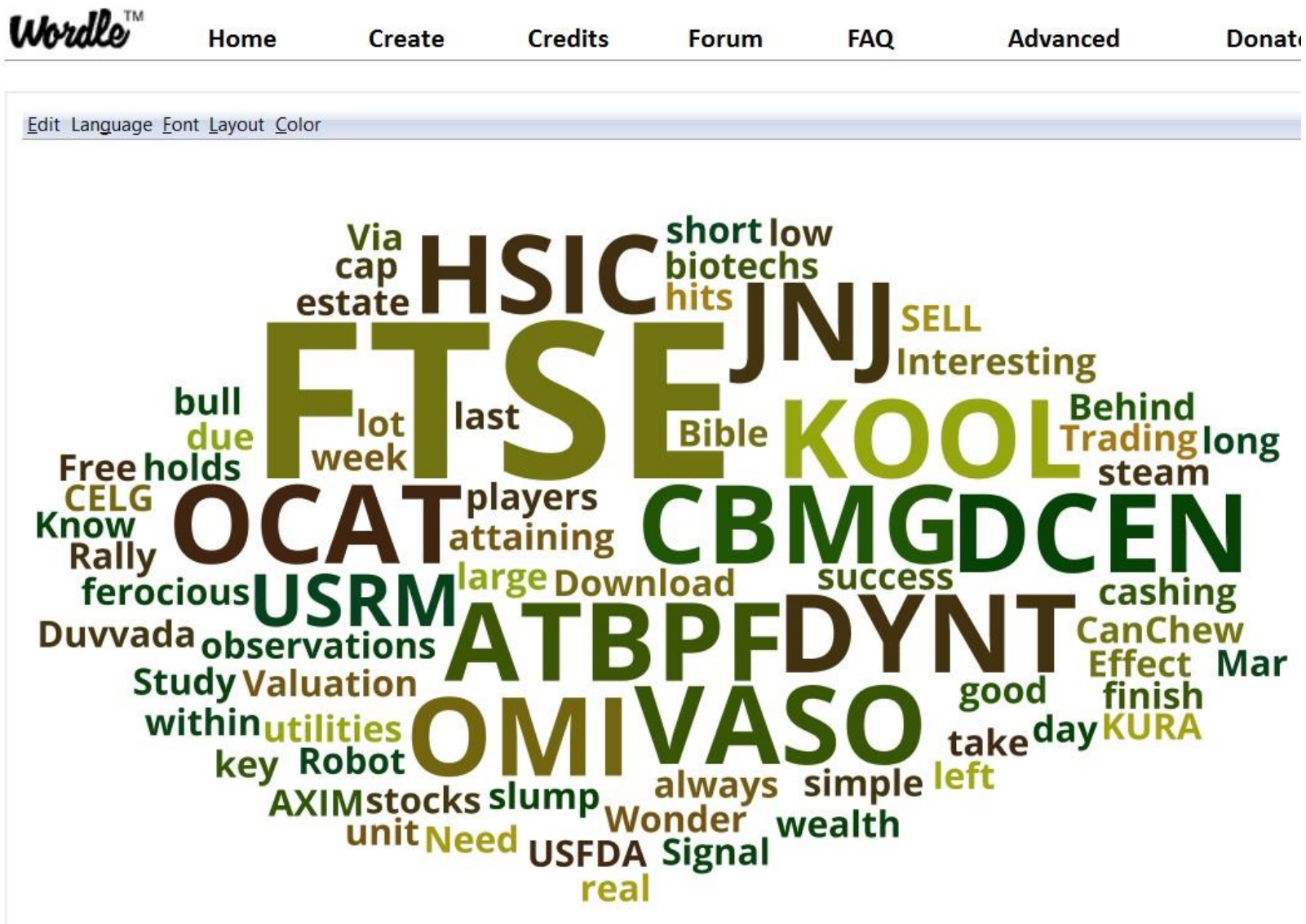
4) JNJ: Johnson and Johnson NYSE Stock (price 126.21 at this moment)

COLLECTING WORDS

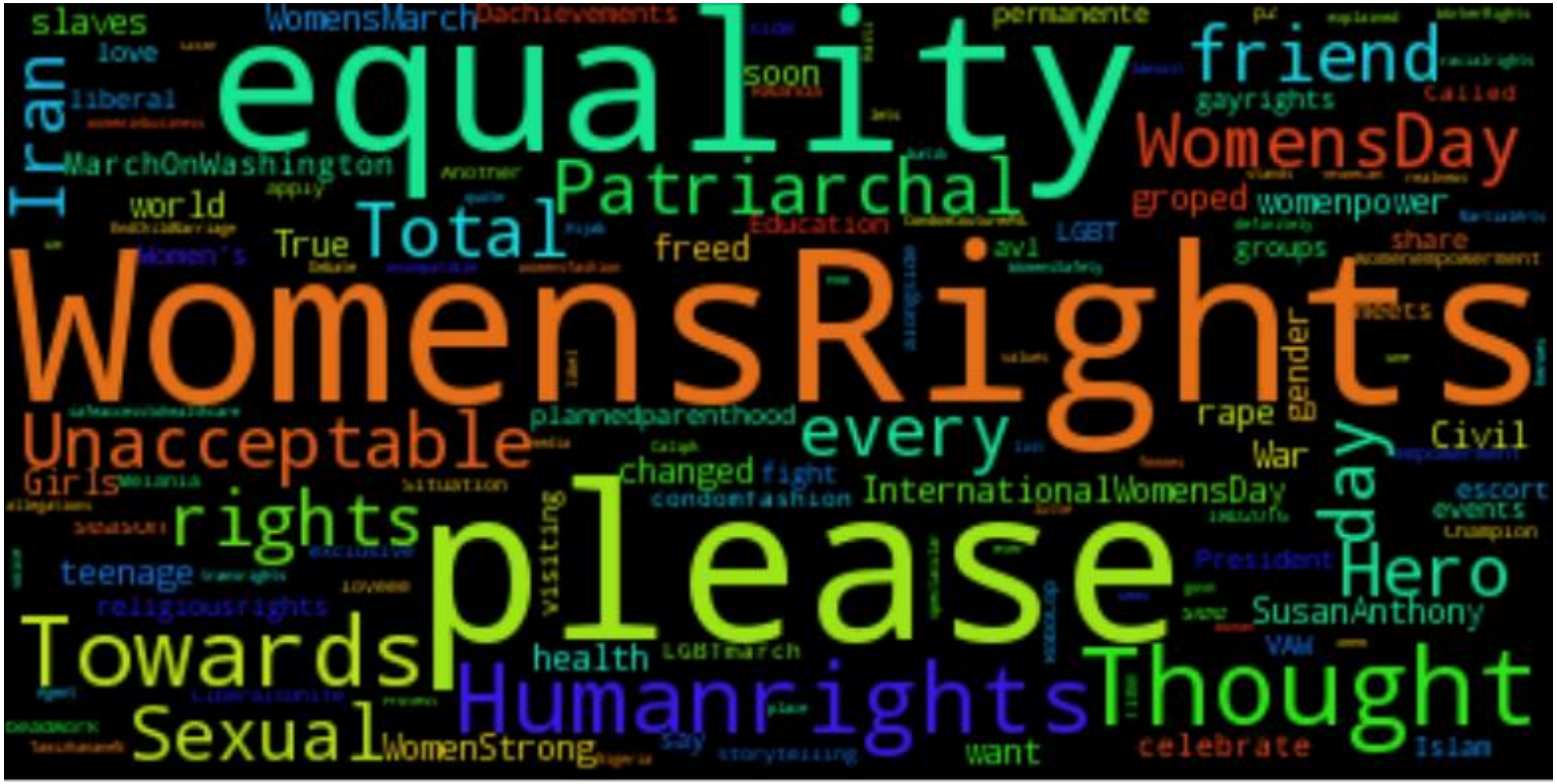
hits week low USFDA observations Duvvada unit long will the Rally the Effect over The Behind The KURA Valuation FTSE SELL Signal Mar Trading Robot stocks down for day utilities real estate slump players always finish last CELG within large cap biotechs What You Need Know About AXIM CanChew Study bull has lot more steam left due ferocious short and simple take and success OMI DYNT DCEN JNJ VASO KOOL CBMG OCAT HSIC ATBPF USRM OMI DYNT DCEN JNJ VASO KOOL CBMG OCAT HSIC ATBPF USRM OMI DYNT DCEN JNJ VASO KOOL CBMG OCAT HSIC ATBPF USRM OMI DYNT DCEN JNJ VASO KOOL CBMG OCAT HSIC ATBPF Interesting Wonder this will good for the Via holds the key attaining wealth the FTSE FTSE FTSE FTSE Are you cashing Download the Free Bible FTSE

OCAT,3
ATBPF,3
FTSE,6
OMI,3
HSIC,3
VASO,3
KOOL,3
JNJ,3
CBMG,3
DCEN,3
DYNT,3
USRM,2

WORDLE: HTTP://WWW.WORDLE.NET/



#WOMENSRIGHTS



WORDLE


womensrights
auspol
Manifesto
Womens
Read
WomensRights
Killings
Honor
Islam
True
Status
United
Nations
happiness
WomensHistoryMonth
delegates
appoints
WomensDay
Honorable
Everything
Trump
well
wealthy
another
item
Commission
Session
Caliph's
Know
cried
two
explained
teaching
Caliph
stand
healthcare
tears
International
equality

VISUALIZING TWITTER DATA

{"timestamp_ms": "1488997991805", "lang": "en", "id_str": "839544543477448710", "text": "Naysayers keep stating the #StockMarket is set to Crash in 2017? they said the same in 2016 https://t.co/Gg12IGTeqv", "place": null, "geo": null, "in_reply_to_screen_name": null, "filter_level": "low", "in_reply_to_status_id_str": null, "is_quote_status": false, "in_reply_to_user_id": null, "favorited": false, "in_reply_to_status_id": null, "entities": {"symbols": [], "urls": [{"url": "https://t.co/Gg12IGTeqv", "expanded_url": "http://tacticalinvestor.com/stock-market-crash-2017-video-reality-or-all-hype/", "display_url": "tacticalinvestor.com/stock-market-c\u2026", "indices": [92, 115]}], "user_mentions": [], "hashtags": [{"text": "StockMarket", "indices": [27, 39]}]}, "id": 839544543477448710, "favorite_count": 0, "created_at": "Wed Mar 08 18:33:11 +0000 2017", "in_reply_to_user_id_str": null, "contributors": null, "truncated": false, "retweeted": false, ...

- by time stamp
- by language
- by hashtags
- by text content or sentiment
- by urls
- by retweet
- etc.

Tweets are related, so networks can be formed.

Tweets are temporal.

Some Tweets have geo data