Notebook 2 - Twitter API

By Michael Carlin and Brock Ogle

In [1]:
```
import json
import pandas as pd
import re
from requests_oauthlib import OAuth1Session
from requests_oauthlib import OAuth1
import requests
import webbrowser
#import the currently required modules
```

In [2]:
```
#Loads the consumer and access tokens generated in Notebook1
with open('tokens1.json') as f:
    tokens = json.load(f)
```

In [3]:
```
#Retrieves consumer and access tokens from json file
consumer_key = tokens['consumer']['consumer_key']
consumer_secret = tokens['consumer']['consumer_secret']
access_token = tokens['access_token']['access_token']
access_token_secret = tokens['access_token']['access_secret']
```

In [4]:
```
#Establishes a common beginning and ending to request url to avoid repiti
base = 'https://api.twitter.com/1.1/statuses/user_timeline.json?screen_na
end = '&count=200'
```

In [5]:
```
def auth_check(auth):
    '''
    This function checks if the access token is still valid. If valid the
    returns the valid authorization, if not the function prompts the user
    their access token via Notebook1

    Parameters:
        auth - current consumer and access token
```

```
    Return:
        auth - valid consumer and access token
    '''
    #URL provided by twitter for checking validity of authorization crede
    url = 'https://api.twitter.com/1.1/account/verify_credentials.json'
    #Establishes current authorization via requests_oauthlib module
    auth = OAuth1(consumer_key, consumer_secret, access_token, access_tok
    response = requests.get(url, auth=auth)     #Uses GET to get response
    if str(response) == '<Response [200]>':     #Checks status of respons
        return auth
    else:
        print 'access token is invalid'
        print 'please visit Notebook1 and run to completion'
```

In [6]:
```
def next_tweets(id,n,auth):
    '''
    This functions appends tweets to a list after a users first 200 tweet

    Parameters:
        id - The eariest possible tweet to retrieve data from
        n - recursive number to stop the collection of data
        auth - valid consumer and access tokens

    Return:
        id - earliest possible tweet for next iteration of next_tweets fu
        n - recursive count, decreased by 1 level before returning
    '''
    #uses &max_id parameter to collect next 200 tweets
    new_endpoint = base+user+'&max_id='+str(id)
    r = requests.get(new_endpoint,auth=auth)   #data from specified endpo
    tweets1 = r.json()                         #converts data to json for
    for tweet in tweets1:
        tweeets.append(tweet['text'])          #appends contents of each
    id = tweet['id'] - 1                       #sets earliest tweet for n
    n = n-1                                    #decreases recursive level
    return id, n
```

In [7]:
```
def tweet_bridge(n,id,auth):
    '''
    This recursive function houses the next_tweets call in order to avoid
    the earliest tweet id not properly resetting if next_tweets was calle

    Parameters:
        id - The eariest possible tweet to retrieve data from
        n - recursive number to stop the collection of data
        auth - valid consumer and access tokens

    Return:
        Function exits if level is 0
    '''
    if n >=1:                              #checks level
        id, n = next_tweets(id,n,auth)     #appends next 200 tweets to lis
        tweet_bridge(n,id,auth)            #recursive call
```

In [8]:
```
def all_tweets(n,auth, timeline_endpoint):
    '''
    This function appends the contents of a users first 200 tweets to a l
    the function necessary to extract further tweets

    Parameters:
        n - recursive number to stop the collection of data
        auth - valid consumer and access tokens
        timeline_endpoint - initial endpoint to extract data

    Return:
        calls tweet_bridge() function
    '''
    r = requests.get(timeline_endpoint, auth=auth)   #data from user's ti
    tweets = r.json()                                 #puts data in json f
    for tweet in tweets:
        tweeets.append(tweet['text'])                #adds contents of twe
    id = tweet['id'] - 1                             #sets earliest next t
    tweet_bridge(n,id,auth)
```

In [9]:
```
def dict_create(list,counts,hash_count):
    '''This calls function necessary to create a dictionary of word frequ

    Parameters:
        list - contents of all tweets
        counts - empty dictionary
        auth - empty dictionary

    Return:
        calls word_count function for tweet in list of tweets
    '''
    for item in list:
        word_count(item,counts,hash_count)        #calls counter function
```

In [10]:
```
def word_count(str,counts,hash_count):
    '''
    This function splits text of each tweet and counts individual words,
    excluding any non alphanumeric characters

    Parameters:
        str - The text of each tweet
        counts - dictionary counter of words in a user's tweets
        hash_counts - dictionary counter of hashtags in user's tweets

    Return:
        None
```

```
    '''
    words = str.split()                          #splits text of tweet on whi
    for word in words:
        word = word.lower()                      #Eliminates miscounts due to
        regex = re.compile('[^a-z0-9\#]')        #Sets regular expression to
        word = regex.sub('',word)                #Each word now only consists
        if word == '':                           #takes care of words with no
            word = 'NoAlphanumericCharacters'
        if word[0] == '#':                      #counts users hashtags
            if word in hash_count:
                hash_count[word] += 1           #appends count
            else:
                hash_count[word] = 1            #starts new count
        if word in counts:
            counts[word] += 1
        if word not in counts and word[0:4] != 'http':  #eliminates hyper
            counts[word] = 1
```

In [11]:
```
def df_ready(dict):
    '''
    This function strips the contents of a dictionary into two equal leng
    to be used in a Pandas DataFrame

    Parameters:
        dict - dictionary to be split

    Return:
        list_word - list of all words
        list_count - list of frequency of all words
    '''
    list_word = []       #blank list
    list_count = []
    for item in dict:
        list_word.append(item)        #appends words to list
        list_count.append(dict[item]) #appends frequencies to list
    return list_word, list_count
```

In [12]:
```
def retweet_finder(auth,endpoint):
    '''
    This function creates lists with times, number of retweets, and numbe
    of favorites for each user's original tweets. This function also
    removes all retweeted, unoriginal content from consideration.

    Parameters:
        auth - valid authorization credentials
        endpoint - endpoint to access data

    Return:
        times_list - time and data of each tweet
        rt_list - number of retweets for each tweet
        fav_list - number of favorites for each tweet
    '''
    times_list = []     #empty list
    rt_list = []
    fav_list = []
    r = requests.get(timeline_endpoint, auth=auth)   #gets data from endp
    tweets = r.json()                                #converts data to js
    for tweet in tweets:
        if tweet['favorite_count'] != 0:             #filters unoriginal
            created_at = tweet['created_at']         #reformats dates and
            created_at2 = created_at[4:19]           #be used in Tableau
            if created_at2[:3] == 'Dec':
                created_at1 = '12/'+ created_at2[4:6]+'/2017 '+created_at
            else:
                created_at1 = '11/'+ created_at2[4:6]+'/2017 '+created_at
            times_list.append(created_at1)           #append respective l
            rt_list.append(tweet['retweet_count'])
            fav_list.append(tweet['favorite_count'])
    return times_list, rt_list, fav_list
```

In [13]:
```
#establishes original valid authorization credentials
url = 'https://api.twitter.com/1.1/account/verify_credentials.json'
auth = OAuth1(consumer_key, consumer_secret, access_token, access_token_s
auth = auth_check(auth)
```

In [14]:
```
#Checks credentials, sets user and original endpoint
auth = auth_check(auth)
user = 'realDonaldTrump'
timeline_endpoint = base+user+end

#creates initial empty dictionary and resets tweeets list
tweeets = []
TRUMP_count = {}
TRUMP_hash_count = {}

#Call functions to create counts of each word/hashtag in tweets
all_tweets(15,auth,timeline_endpoint)
dict_create(tweeets,TRUMP_count,TRUMP_hash_count)

#Creates DataFrames to later be merged/exported
TRUMP_list_word, TRUMP_list_count = df_ready(TRUMP_count)
df_TRUMP_count = pd.DataFrame({'word':TRUMP_list_word,'freq':TRUMP_list_c
TRUMP_list_hash, TRUMP_list_hash_count = df_ready(TRUMP_hash_count)
df_TRUMP_hash = pd.DataFrame({'word':TRUMP_list_hash,'freq':TRUMP_list_ha

#Sets extra variable for Tableau analysis
df_TRUMP_count['user'] = 'Trump'
df_TRUMP_hash['user'] = 'Trump'

#Creates popularity data
timeline_endpoint = base+user+end
TRUMP_times_list, TRUMP_rt_list, TRUMP_fav_list = retweet_finder(auth,tim

#Creates popularity DataFrames and sets extra identifier variable
df_TRUMP_popular = pd.DataFrame({'created_at':TRUMP_times_list,
                                 'rt_count':TRUMP_rt_list,
                                 'fav_count':TRUMP_fav_list})
df_TRUMP_popular['user'] = 'Trump'
```

In [15]:
```
#Checks credentials, sets user and original endpoint
auth = auth_check(auth)
```

```
user = 'gop'
timeline_endpoint = base+user+end

#creates initial empty dictionary and resets tweeets list
tweeets = []
GOP_count = {}
GOP_hash_count = {}

#Call functions to create counts of each word/hashtag in tweets
all_tweets(15,auth,timeline_endpoint)
dict_create(tweeets,GOP_count,GOP_hash_count)

#Creates DataFrames to later be merged/exported
GOP_list_word, GOP_list_count = df_ready(GOP_count)
df_GOP_count = pd.DataFrame({'word':GOP_list_word,'freq':GOP_list_count})
GOP_list_hash, GOP_list_hash_count = df_ready(GOP_hash_count)
df_GOP_hash = pd.DataFrame({'word':GOP_list_hash,'freq':GOP_list_hash_cou

#Sets extra variable for Tableau analysis
df_GOP_count['user'] = 'GOP'
df_GOP_hash['user'] = 'GOP'

#Creates popularity data
timeline_endpoint = base+user+end
GOP_times_list, GOP_rt_list, GOP_fav_list = retweet_finder(auth,timeline_

#Creates popularity DataFrames and sets extra identifier variable
df_GOP_popular = pd.DataFrame({'created_at':GOP_times_list,
                               'rt_count':GOP_rt_list,
                               'fav_count':GOP_fav_list})
df_GOP_popular['user'] = 'GOP'
```

In [16]:

```
#Checks credentials, sets user and original endpoint
auth = auth_check(auth)
user = 'theDemocrats'
timeline_endpoint = base+user+end

#creates initial empty dictionary and resets tweeets list
tweeets = []
DEM_count = {}
DEM_hash_count = {}

#Call functions to create counts of each word/hashtag in tweets
```

```
all_tweets(15,auth,timeline_endpoint)
dict_create(tweeets,DEM_count,DEM_hash_count)

#Creates DataFrames to later be merged/exported
DEM_list_word, DEM_list_count = df_ready(DEM_count)
df_DEM_count = pd.DataFrame({'word':DEM_list_word,'freq':DEM_list_count})
DEM_list_hash, DEM_list_hash_count = df_ready(DEM_hash_count)
df_DEM_hash = pd.DataFrame({'word':DEM_list_hash,'freq':DEM_list_hash_cou

#Sets extra variable for Tableau analysis
df_DEM_count['user'] = 'DEM'
df_DEM_hash['user'] = 'DEM'

#Creates popularity data
timeline_endpoint = base+user+end
DEM_times_list, DEM_rt_list, DEM_fav_list = retweet_finder(auth,timeline_

#Creates popularity DataFrames and sets extra identifier variable
df_DEM_popular = pd.DataFrame({'created_at':DEM_times_list,
                               'rt_count':DEM_rt_list,
                               'fav_count':DEM_fav_list})
df_DEM_popular['user'] = 'DEM'
```

In [17]:

```
#Checks credentials, sets user and original endpoint
auth = auth_check(auth)
user = 'HouseDemocrats'
timeline_endpoint = base+user+end

#creates initial empty dictionary and resets tweeets list
tweeets = []
HouseDEM_count = {}
HouseDEM_hash_count = {}

#Call functions to create counts of each word/hashtag in tweets
all_tweets(15,auth,timeline_endpoint)
dict_create(tweeets,HouseDEM_count,HouseDEM_hash_count)

#Creates DataFrames to later be merged/exported
HouseDEM_list_word, HouseDEM_list_count = df_ready(HouseDEM_count)
df_HouseDEM_count = pd.DataFrame({'word':HouseDEM_list_word,'freq':HouseD
HouseDEM_list_hash, HouseDEM_list_hash_count = df_ready(HouseDEM_hash_cou
df_HouseDEM_hash = pd.DataFrame({'word':HouseDEM_list_hash,'freq':HouseDE
```

```
#Sets extra variable for Tableau analysis
df_HouseDEM_count['user'] = 'DEM'
df_HouseDEM_hash['user'] = 'DEM'
```

In [18]:
```
#Checks credentials, sets user and original endpoint
auth = auth_check(auth)
user = 'HouseGOP'
timeline_endpoint = base+user+end

#creates initial empty dictionary and resets tweeets list
tweeets = []
HouseGOP_count = {}
HouseGOP_hash_count = {}

#Call functions to create counts of each word/hashtag in tweets
all_tweets(15,auth,timeline_endpoint)
dict_create(tweeets,HouseGOP_count,HouseGOP_hash_count)

#Creates DataFrames to later be merged/exported
HouseGOP_list_word, HouseGOP_list_count = df_ready(HouseGOP_count)
df_HouseGOP_count = pd.DataFrame({'word':HouseGOP_list_word,'freq':HouseG
HouseGOP_list_hash, HouseGOP_list_hash_count = df_ready(HouseGOP_hash_cou
df_HouseGOP_hash = pd.DataFrame({'word':HouseGOP_list_hash,'freq':HouseGO

#Sets extra variable for Tableau analysis
df_HouseGOP_count['user'] = 'GOP'
df_HouseGOP_hash['user'] = 'GOP'
```

In [19]:
```
#Checks credentials, sets user and original endpoint
auth = auth_check(auth)
user = 'SenateGOP'
timeline_endpoint = base+user+end

#creates initial empty dictionary and resets tweeets list
tweeets = []
SenateGOP_count = {}
SenateGOP_hash_count = {}

#Call functions to create counts of each word/hashtag in tweets
all_tweets(15,auth,timeline_endpoint)
dict_create(tweeets,SenateGOP_count,SenateGOP_hash_count)

#Creates DataFrames to later be merged/exported
SenateGOP_list_word, SenateGOP_list_count = df_ready(SenateGOP_count)
df_SenateGOP_count = pd.DataFrame({'word':SenateGOP_list_word,'freq':Sena
SenateGOP_list_hash, SenateGOP_list_hash_count = df_ready(SenateGOP_hash_
df_SenateGOP_hash = pd.DataFrame({'word':SenateGOP_list_hash,'freq':Senat

#Sets extra variable for Tableau analysis
df_SenateGOP_count['user'] = 'GOP'
df_SenateGOP_hash['user'] = 'GOP'
```

In [20]:
```
#Checks credentials, sets user and original endpoint
auth = auth_check(auth)
user = 'SenateDems'
timeline_endpoint = base+user+end
```

```
#creates initial empty dictionary and resets tweeets list
tweeets = []
SenateDEM_count = {}
SenateDEM_hash_count = {}

#Call functions to create counts of each word/hashtag in tweets
all_tweets(15,auth,timeline_endpoint)
dict_create(tweeets,SenateDEM_count,SenateDEM_hash_count)

#Creates DataFrames to later be merged/exported
SenateDEM_list_word, SenateDEM_list_count = df_ready(SenateDEM_count)
df_SenateDEM_count = pd.DataFrame({'word':SenateDEM_list_word,'freq':Sena
SenateDEM_list_hash, SenateDEM_list_hash_count = df_ready(SenateDEM_hash_
df_SenateDEM_hash = pd.DataFrame({'word':SenateDEM_list_hash,'freq':Senat

#Sets extra variable for Tableau analysis
df_SenateDEM_count['user'] = 'DEM'
df_SenateDEM_hash['user'] = 'DEM'
```

In [21]:
```
#Combines selected DataFrames for easier Tableau analysis
#Exports combined DataFrame to a .csv file
frames_popular = [df_TRUMP_popular,df_GOP_popular,df_DEM_popular]
df_popular = pd.concat(frames_popular)
df_popular.to_csv('popular.csv',index=False)
```

In [22]:
```
#Combines selected DataFrames for easier Tableau analysis
#Exports combined DataFrame to a .csv file
frames_popular_Trumpless = [df_GOP_popular,df_DEM_popular]
df_popular_Trumpless = pd.concat(frames_popular_Trumpless)
df_popular_Trumpless.to_csv('popular_Trumpless.csv',index=False)
```

In [23]:
```
#Combines selected DataFrames for easier Tableau analysis
#Exports combined DataFrame to a .csv file
frames_house = [df_HouseDEM_count,df_HouseGOP_count]
df_house = pd.concat(frames_house)
df_house.to_csv('house_count.csv')
```

In [24]:
```
#Combines selected DataFrames for easier Tableau analysis
#Exports combined DataFrame to a .csv file
frames_senate = [df_SenateDEM_count,df_SenateGOP_count]
df_senate = pd.concat(frames_senate)
df_senate.to_csv('senate_count.csv')
```

In [25]:
```
#Combines selected DataFrames for easier Tableau analysis
#Exports combined DataFrame to a .csv file
frames_all = [df_DEM_count,df_GOP_count,df_TRUMP_count]
df_all = pd.concat(frames_all)
df_all.to_csv('all_count.csv')
```

In [26]:
```
#Combines selected DataFrames for easier Tableau analysis
#Exports combined DataFrame to a .csv file
frames_all_hash = [df_DEM_hash,df_GOP_hash,df_TRUMP_hash]
df_all_hash = pd.concat(frames_all_hash)
df_all_hash.to_csv('all_hash.csv')
```

In [ ]: