# Alex Teboul

## Assignment 5

**DSC 450**

**Due Sunday, June 9th**

**Problem 1 -** *Apologies on late submission - solutions not up when submitted.*
In this assignment we are going to work with a larger collection of tweets (10,000) that are available here: http://rasinsrv07.cstcis.cti.depaul.edu/CSC455/Assignment5.txt

The tweets are all on separate lines, but some of the tweets are intentionally damaged and will not parse properly. You will need to store these tweets in a separate "error" file. At the bottom of the page you can find python code that will let you skip over badly formed tweets.

**Part a.**
Create a new SQL table for the user dictionary. It should contain the following attributes "id", "name", "screen_name", "description" and "friends_count". Modify your SQL table from Assignment 4 to include "user_id" columns which will be a foreign key referencing the user table and a "geo" column (needed for 2-a).

```
#Create the sql table for the user dictionary

#imports
import urllib.request, time, json, sqlite3

conn = sqlite3.connect('Tweets_Database_v1.db')
c = conn.cursor()
wFD = urllib.request.urlopen('http://rasinsrv07.cstcis.cti.depaul.edu/CSC455/Assignment5.txt') #get the file
```

```
#create User Table
create_UserTable = '''CREATE TABLE User (
            ID              INTEGER PRIMARY KEY,
            NAME            TEXT,
            SCREEN_NAME     TEXT,
            DESCRIPTION     TEXT,
            FRIENDS_COUNT   INTEGER
         );'''
c.execute('DROP TABLE IF EXISTS User');
c.execute(create_UserTable)
```

```
#create Tweets Table (Assignment 4 + user_id + geo)
create_TweetsTable = '''CREATE TABLE Tweets (
            ID                      INTEGER PRIMARY KEY,
            Created_At              DATE,
            Text                    TEXT,
            Source                  TEXT,
            In_Reply_to_User_ID     INTEGER,
            In_Reply_to_Screen_Name TEXT,
            In_Reply_to_Status_ID   INTEGER,
            Retweet_Count           INTEGER,
            Contributors            TEXT,
            Geo                     TEXT,
            User_ID                 INTEGER,
                FOREIGN KEY (User_ID) REFERENCES User(ID)

         );'''
c.execute('DROP TABLE IF EXISTS Tweets');
c.execute(create_TweetsTable)
```

**Part b.**

Write python code that is going to read and load the Assignment5.txt file from the web and populate both of your tables (Tweet table from Assignment4 and User table from this assignment). You can use the same code from the previous assignment with an additional step of inserting into the new table. For tweets that could not parse, write them into an Assignment5_errors.txt file (do not ignore them).

```
#get the whole text file string
tweets = wFD.read()
```

```
#Convert bytes-like object tweets to 'str'
tweets=tweets.decode('utf8')
```

```
#Take the tweets string and split it up by new line character
tweetlines=tweets.split('\n')
type(tweetlines) #Check that type is now list
tweetlines[-2]   #Check that the last tweet comes out correctly to ensure that no errors in list generation
len(tweetlines)  #Check how many tweets
```

```
10001
```

```
def load_Tweets_User(allTweets):
    '''Loads the tweets into the Tweets and User Tables. Accepts allTweets list as parameter.'''

    #Open the error file before the loop
    err_file = open('Assignment5_errors.txt', 'w')
    error_counter=0

    #for every tweet in the list
    for tweet in allTweets:
        try:
            tweetDict = json.loads(tweet)

            #Placeholders for new rows in tables
            new_TweetsRow = [] #each new row to be inserted into Tweets Table
            new_UserRow = [] #each new row to be inserted into User Table

            tweetKeys = ['id_str','created_at','text','source','in_reply_to_user_id',
                         'in_reply_to_screen_name', 'in_reply_to_status_id', 'retweet_count', 'contributors',
                         'geo','user'] #a list of keys for the tweet table
            userKeys = ['id','name','screen_name','description','friends_count'] #for the user table
```

```
            # For each dictionary key in Tweets append data to the row list
            for key in tweetKeys:
                if tweetDict[key] == 'null' or tweetDict[key] == '': #Handle nulls - important for geo for 2a
                    new_TweetsRow.append(None)   # NULL as None
                elif key == 'user':
                    new_TweetsRow.append(tweetDict['user']['id']) #get the user_id
                else:
                    new_TweetsRow.append(tweetDict[key]) # value as-is

            # For each dictionary key in User append data to the row list
            for key in userKeys:
                if tweetDict['user'][key] == 'null' or tweetDict['user'][key] == '': #Handle nulls - important for geo
                    new_UserRow.append(None)   # NULL as None
                else:
                    new_UserRow.append(tweetDict['user'][key]) # value as-is

            c.execute('INSERT INTO User VALUES(?,?,?,?,?)', (new_UserRow))
            c.execute('INSERT INTO Tweets VALUES(?,?,?,?,?,?,?,?,?,?,?)', (new_TweetsRow))
```

```
        except:
            #Write the problematic tweet to a new file
            error_counter+=1
            error_out = str(error_counter) + tweet
            err_file.write(error_out)

    #Close the error file after the loop
    err_file.close()

    return error_counter
```

```
start = time.time()
load_Tweets_User(tweetlines)
end   = time.time()

print("load_Tweets_User took ", (end-start), ' seconds.')
print("Loaded ", c.execute('SELECT COUNT(*) FROM Tweets').fetchall()[0], " rows")
print("Loaded ", c.execute('SELECT COUNT(*) FROM User').fetchall()[0], " rows")
```

```
load_Tweets_User took  2.0811970233917236  seconds.
Loaded  (9501,)  rows
Loaded  (9683,)  rows
```

```
wFD.close()
c.close()
conn.commit()
conn.close()
```

---

## Problem 2

### Part a.

Write and execute SQL query to do the following: Find the tweets without associated geo entry (hint: it should involve a NULL).

**c.execute('SELECT * FROM Tweets WHERE Geo IS NULL').fetchall()**

```
In [114]: c.execute('SELECT * FROM Tweets WHERE Geo IS NULL').fetchall()
```

```
Out[114]: [(468541694279811072,
            'Tue May 20 00:00:19 +0000 2014',
            '@T_narun \nあほちんw',
            '<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>',
            2323449421,
            'T_narun',
            468428961039409150,
            0,
            None,
            None,
            1052911537),
           (468541694279819264,
            'Tue May 20 00:00:19 +0000 2014',
            'Just posted a photo http://t.co/GApvtMRQJ2',
            '<a href="http://instagram.com" rel="nofollow">Instagram</a>',
            None,
            None,
            None,
            0,
```

This is 9501 tweets.

### Part b.

Write python code that is going to perform the same computation as 2-a.

**df[df['geo'].isnull()]**

```
import pandas as pd
#import json
data = []
error_tally =0

for tweet in tweetlines:
    try:
        a_tweet = json.loads(tweet)
        data.append((a_tweet["created_at"], a_tweet["id_str"], a_tweet["text"], a_tweet["source"],
        a_tweet["in_reply_to_user_id"], a_tweet["in_reply_to_screen_name"], a_tweet["in_reply_to_status_id"],
        a_tweet["retweet_count"], a_tweet["contributors"],
        a_tweet["geo"], a_tweet["user"]))
    except:
        error_tally+=1
labels = ['created_at', 'id_str', 'text', 'source', 'in_reply_to_user_id', 'in_reply_to_screen_name',
        'in_reply_to_status_id', 'retweet_count', 'contributors', 'geo', 'user']
df=pd.DataFrame.from_records(data,columns=labels)
df.head()
```

| str | text | source | in_reply_to_user_id | in_reply_to_screen_name | in_reply_to_status_id | retweet_count | contributors | geo |
|---|---|---|---|---|---|---|---|---|
| 74 | la asusto a selena me dice es joda te vy a ext... | <a href="https://mobile.twitter.com" rel="nofo... | NaN | None | NaN | 0 | None | None |

```
df[df['geo'].isnull()]
```

| text | source | in_reply_to_user_id | in_reply_to_screen_name | in_reply_to_status_id | retweet_count | contributors | geo | user |
|---|---|---|---|---|---|---|---|---|
| a me ext... | <a href="https://mobile.twitter.com" rel="nofo... | NaN | None | NaN | 0 | None | None | {'id': 367361405, 'id_str': '367361405', 'name... |
| メ】 ットリ/ 'メカ /ー... | <a href="http://makebot.sh" rel="nofollow">ほたる... | NaN | None | NaN | 0 | None | None | {'id': 1605442621, 'id_str': '1605442621', 'na... |
| ん き まし | <a href="http://admin.pure-c.jp/prog01_test3/t... | NaN | None | NaN | 0 | None | None | {'id': 1685288690, 'id_str': |

---

## Problem 3

### Part a.

Write and execute SQL query that finds the longest and the shortest tweet text message (if there is a tie, you must return **all** shortest and longest tweet messages, not just one). This is SQL-only, using your SQLite database.

**c.execute('SELECT Text, LENGTH(Text) FROM Tweets WHERE Text = (SELECT MIN(TEXT) FROM Tweets WHERE LENGTH(Text) = (SELECT MIN(LENGTH(Text)) FROM Tweets)) OR Text = (SELECT MIN(Text) FROM Tweets WHERE LENGTH(Text) = (SELECT MAX(LENGTH(Text)) FROM Tweets))').fetchall()**

```
#c.execute('SELECT length(Text) FROM Tweets WHERE id_str=468541694288207874').fetchall()
c.execute('SELECT Text, LENGTH(Text) FROM Tweets WHERE Text = (SELECT MIN(TEXT) FROM Tweets WHERE LENGTH(Text) = (SELEC
```

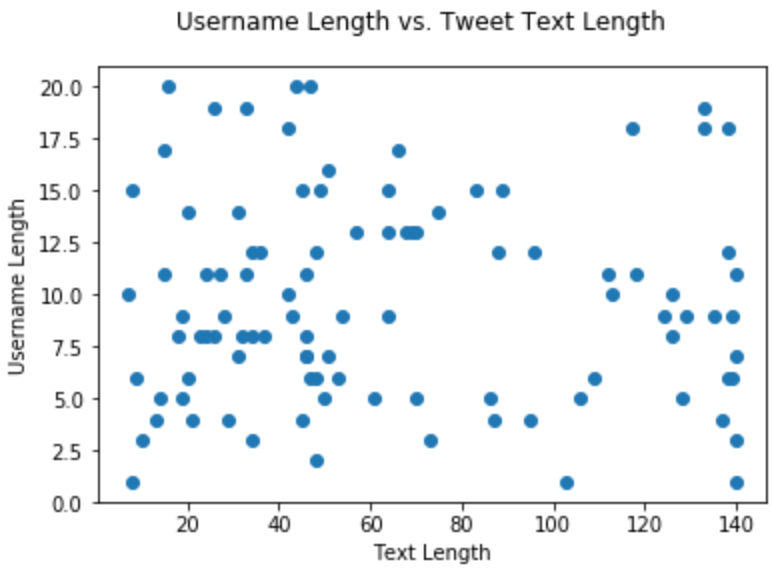
```
[('.', 1),
 ('.', 1),
 ('RT @Rileymonroeee: When you start catching feelings , becoming attached , wanting more &lt;&lt;&gt;&gt;&gt;&lt;&l
t;&lt;&lt;&gt;&lt;&gt;&gt;&gt;&lt;&lt;&gt;&gt; 😩😍🏩💞',
  164)]
```

--Cleaned up below for read-ability

```
 SELECT
   Text,
   LENGTH(Text)
FROM Tweets
WHERE Text = (
   SELECT
      MIN(TEXT)
   FROM Tweets
   WHERE LENGTH(Text) = (
      SELECT
         MIN(LENGTH(Text))
      FROM Tweets
   )
)
OR Text = (
   SELECT
      MIN(Text)
   FROM Tweets
   WHERE LENGTH(Text) = (
      SELECT
         MAX(LENGTH(Text))
      FROM Tweets
   )
);
```

**Part b.**

Use python to plot the lengths of first 100 tweets (only 100, not all of the tweets) versus the length of the username for the user on a graph. Use a scatterplot.




```
#get x and y
x = df['text'][:99].str.len()
df2 = pd.concat([df.drop(['user'], axis=1), df['user'].apply(pd.Series)], axis=1)
y = df2['name'][:99].str.len()
```

```
#plot it
from matplotlib import pyplot as plt
plt.scatter(x,y)
plt.suptitle('Username Length vs. Tweet Text Length')
plt.xlabel('Text Length')
plt.ylabel('Username Length')
```

**Part c.**

Using python, identify the top-5 most frequent terms (words separated by ' ') that are at least 6 characters or longer (i.e. ignore short terms) in the text of the tweets. It is up to you whether you prefer to use the contents of the loaded database (reading tweets from SQLite database) or the contents of the original Assignment5.txt file (reading tweets directly from the file as in Part-1).

**Top 5 words:** follow, everyone, people, really, please

```
#Note the dataframe df is the same one from previous problem
allwords = pd.Series(' '.join(df['text']).lower().split()) #Get all the unique words by splitting on ' '
allwords_conditioned = allwords[allwords.str.len() >=6] #filter the data frame to get only words of length 6 and up
allwords_conditioned.value_counts()[:5] #perform value counts and output top 5 most frequent words along with count
```

```
follow      204
everyone     71
people       66
really       63
please       63
dtype: int64
```