DSC 450: Database Processing for Large-Scale Analytics
Take-home Final

Part 1

We will use one full day worth of tweets as our input (there are total of 4.4M tweets in this file, but we will intentionally use fewer tweets to run this final):
http://dbgroup.cdm.depaul.edu/DSC450/OneDayOfTweets.txt
Execute the following tasks with 60,000 tweets and 600,000 tweets.

a. Use python to download tweets from the web and save to a local text file (not into a database yet, just to a text file). This is as simple as it sounds, all you need is a for-loop that reads lines and writes them into a file, just don't forget to add '\n' at the end so they are, in fact, on separate lines.

   NOTE: Do not call read() or readlines(). That command will attempt to read the entire file which is too much data. Clicking on the link in the browser would cause the same problem.

```
# load 600000 tweets
num = 600000
start = time.time()
webFD = urllib.request.urlopen("http://dbgroup.cdm.depaul.edu/DSC450/OneDayOfTweets.txt")

fLocal1 = open("tweetLocal1.txt",'w')



i = 0
for line in webFD:
    fLocal1.write(line.decode("utf-8") + "\n")
    i +=1
    if i >= num:
        #cursor.executemany("INSERT...", batch)
        # save line to the local file
        break

fLocal1.close()

end = time.time()
print("Elapsed time of 1-a, load 600000: " + str(end - start) + "seconds")
```

Output:

```
Elapsed time of 1-a, load 60000: 85.4826819896698seconds
Elapsed time of 1-a, load 600000: 1157.4195201396942seconds
```

b. For *text*, *in_reply_to_user_id* and *in_reply_to_screen_name* in Tweet table and for *screen_name* in User table, find the length of the longest string in the file in 1-a and compare it to your data type size (you only have to change your table if the data type turns out to be too short). You only need to do 1-b for 600,000 tweets file and you do not need to use SQLite database.

```
fLocal = open("tweetLocal1.txt",'r')
allTweets = fLocal.readlines()
BadData=[]
list_twitterData = []
list_user = []

len_id = 0
len_reply_name = 0
len_name = 0

for tweet in allTweets:
    try:
        tdict = json.loads(tweet)
        out_UserData_name = {x: tdict["user"][x] for x in ["screen_name"] }
        out_TweetData_id = {x: tdict[x] for x in ["in_reply_to_user_id_str"] }
        out_TweetData_name = {x:tdict[x] for x in ["in_reply_to_screen_name"]}

        if out_UserData_name["screen_name"]:
            if len(out_UserData_name["screen_name"]) > len_name:
                len_name = len(out_UserData_name["screen_name"])
        if out_TweetData_id["in_reply_to_user_id_str"]:
            print(type(out_TweetData_id["in_reply_to_user_id_str"]))
            if len(out_TweetData_id["in_reply_to_user_id_str"]) > len_id:
                len_id = len(out_TweetData_id["in_reply_to_user_id_str"])
        if out_TweetData_name["in_reply_to_screen_name"]:
            if len(out_TweetData_name["in_reply_to_screen_name"]) > len_reply_name:
                len_reply_name = len(out_TweetData_name["in_reply_to_screen_name"])

    except ValueError:
        # Handle the problematic tweet, which in your case would require writing it to another file #print (type(tweet))
```

```
        BadData.append(tweet)


print("The longest length of screen_name is:", len_name)
print("The longest in_reply_to_user_id of screen_name is:", len_id)
print("The longest in_reply_to_screen_name of screen_name is:", len_reply_name)



fLocal.close()
```

Output:

```
The longest length of screen_name is: 15
The longest in_reply_to_user_id of screen_name is: 10
The longest in_reply_to_screen_name of screen_name is: 15
```

My data type length is 25.

c. Repeat what you did in part 1-a, but instead of saving tweets to the file, populate the 3-table schema that you previously created in SQLite. Be sure to execute commit and verify that the data has been successfully loaded. Report loaded row counts for each of the 3 tables.

NOTE: If your schema contains a foreign key in the Geo table or relies on TweetID as the primary key for the Geo table, you should fix your schema. Geo entries should be identified based on the location they represent. There should **not** be any "blank" Geo entries such as (ID, None, None, None).

```
i = 0


BadData=[]
list_twitterData = []
list_user = []
list_geoData = []

start_c = time.time()
for tweet in webFD:
    try:
      tdict = json.loads(tweet)
      out_UserData = {x: tdict["user"][x] for x in ["id", "name","screen_name","description","friends_count"]  }
      out_TweetData = {x: tdict[x] for x in ["created_at","id", "id_str","text", "source", "in_reply_to_user_id",
"in_reply_to_screen_name", "in_reply_to_status_id", "retweet_count","contributors"]  }


      # get the Geo Data from .txt file
      """" an example:  geo":{"type":"Point","coordinates":[25.93169839,-80.28004283]}, """
      if_geoEnabled = {x:tdict["user"][x] for x in["geo_enabled"]}
```

```
    if_geo = {x:tdict[x] for x in["geo"]}
    if if_geoEnabled["geo_enabled"] == True and if_geo["geo"] != None:
      list_geo = []
      out_GeoData = {x: if_geo["geo"][x] for x in ["type","coordinates"]}
      #out_GeoData = {x: tdict[x] for x in ["corrdinates"[0],"coordinates"[1]]}
      t_geo = tuple(out_GeoData.values())

      geo_id = out_TweetData["id"]
      geo_created_at = out_TweetData["created_at"]
      list_geo.append(geo_id) # add "id" value
      list_geo.append(geo_created_at) # add "created_at" value
      list_geo.append(t_geo[0]) # add "type" value
      list_geo.append(t_geo[1][0]) # add "cordinates" value
      list_geo.append(t_geo[1][1]) # add "cordinates" value

      #list_geoData.append(tuple(list_geo))

      stmt3 = "INSERT OR IGNORE INTO GeoData (id, created_at, type, lon, lat) VALUES (?, ?, ?, ?,?);"
      cursor.execute(stmt3, tuple(list_geo))




    #list_user.append(tuple(out_UserData.values()))
    #list_twitterData.append(tuple(out_TweetData.values()))
    stmt1 = "INSERT OR IGNORE INTO UserData (id,name,screen_name,description,friends_count) VALUES (?, ?, ?, ?,?);"
    cursor.execute(stmt1, tuple(out_UserData.values()))

    stmt2 = "INSERT OR IGNORE INTO TwitterData (created_at, id, id_str, text, source, in_reply_to_user_id,
in_reply_to_screen_name, in_reply_to_status_id, retweet_count, contributors) VALUES (?, ?, ?, ?, ?,?,?,?,?,?);"
    cursor.execute(stmt2, tuple(out_TweetData.values()))

    i +=1


  except ValueError:
    # Handle the problematic tweet, which in your case would require writing it to another file
```

```
    #print (type(tweet))
    BadData.append(tweet.decode("utf-8") )



  if i >= 60000:
    #cursor.executemany("INSERT...", batch)
    # save line to the local file
    break


conn.commit()   # finalize inserted data

stmt4 = "SELECT COUNT(*) FROM UserData;"
res = cursor.execute(stmt4)
print(res.fetchall())

stmt5 = "SELECT COUNT(*) FROM TwitterData;"
res = cursor.execute(stmt5)
print(res.fetchall())

stmt6 = "SELECT COUNT(*) FROM GeoData;"
res = cursor.execute(stmt6)
print(res.fetchall())


end_c = time.time()
print("Elapsed time of 1-c: " + str(end_c - start_c) + "seconds")
```

60000 tweets output:

```
[(57838,)]
[(51447,)]
[(1349,)]
Elapsed time of 1-c: 88.8041410446167seconds
```

600000 tweets output:

```
[(531324,)]
[(507897,)]
[(14477,)]
Elapsed time of 1-c: 1120.807655096054seconds
```

d. Use your locally saved tweet file to repeat the database population step from part-c. That is, load the tweets into the 3-table database using your saved file with tweets. This is the **same** code as in 1-c, but reading tweets from your file, not from the web.

```
 start_d = time.time()
fLocal = open("tweetLocal.txt",'r')
allTweets = fLocal.readlines()


i = 0


BadData=[]
list_twitterData = []
list_user = []
list_geoData = []



for tweet in allTweets:
    try:
      tdict = json.loads(tweet)
      out_UserData = {x: tdict["user"][x] for x in ["id", "name","screen_name","description","friends_count"]  }
      out_TweetData = {x: tdict[x] for x in ["created_at","id", "id_str","text", "source", "in_reply_to_user_id",
"in_reply_to_screen_name", "in_reply_to_status_id", "retweet_count","contributors"]  }


      # get the Geo Data from .txt file
      """" an example:  geo":{"type":"Point","coordinates":[25.93169839,-80.28004283]}, """"
      if_geoEnabled = {x:tdict["user"][x] for x in["geo_enabled"]}
      if_geo = {x:tdict[x] for x in["geo"]}
      if if_geoEnabled["geo_enabled"] == True and if_geo["geo"] != None:
        list_geo = []
        out_GeoData = {x: if_geo["geo"][x] for x in ["type","coordinates"]}
        #out_GeoData = {x: tdict[x] for x in ["corrdinates"[0],"coordinates"[1]]}
        t_geo = tuple(out_GeoData.values())


        geo_id = out_TweetData["id"]
        geo_created_at = out_TweetData["created_at"]
        list_geo.append(geo_id) # add "id" value
        list_geo.append(geo_created_at) # add "created_at" value
        list_geo.append(t_geo[0]) # add "type" value
```

```
        list_geo.append(t_geo[1][0]) # add "cordinates" value
        list_geo.append(t_geo[1][1]) # add "cordinates" value

        #list_geoData.append(tuple(list_geo))

        stmt3 = "INSERT OR IGNORE INTO GeoData (id, created_at, type, lon, lat) VALUES (?, ?, ?, ?,?);"
        cursor.execute(stmt3, tuple(list_geo))




      #list_user.append(tuple(out_UserData.values()))
      #list_twitterData.append(tuple(out_TweetData.values()))
      stmt1 = "INSERT OR IGNORE INTO UserData (id,name,screen_name,description,friends_count) VALUES (?, ?, ?, ?,?);"
      cursor.execute(stmt1, tuple(out_UserData.values()))

      stmt2 = "INSERT OR IGNORE INTO TwitterData (created_at, id, id_str, text, source, in_reply_to_user_id,
in_reply_to_screen_name, in_reply_to_status_id, retweet_count, contributors) VALUES (?, ?, ?, ?, ?,?,?,?,?,?);"
      cursor.execute(stmt2, tuple(out_TweetData.values()))

      i +=1


    except ValueError:
      # Handle the problematic tweet, which in your case would require writing it to another file
      #print (type(tweet))
      BadData.append(tweet)



    if i >= 60000:
        #cursor.executemany("INSERT...", batch)
        # save line to the local file
        break


conn.commit()   # finalize inserted data
```

```
stmt4 = "SELECT COUNT(*) FROM UserData;"
res = cursor.execute(stmt4)
print(res.fetchall())

stmt5 = "SELECT COUNT(*) FROM TwitterData;"
res = cursor.execute(stmt5)
print(res.fetchall())

stmt6 = "SELECT COUNT(*) FROM GeoData;"
res = cursor.execute(stmt6)
print(res.fetchall())



end_d = time.time()
print("Elapsed time of 1-d: " + str(end_d - start_d) + "seconds")
```

60000 tweets output:

```
[(57838,)]
[(51447,)]
[(1349,)]
Elapsed time of 1-d: 15.37907099723816seconds
```

600000 tweets output:

```
[(531324,)]
[(507897,)]
[(14477,)]
Elapsed time of 1-d: 92.43683886528015seconds
```

e. Repeat the same step with a batching size of 3000 (i.e. by inserting 3000 rows at a time with executemany instead of doing individual inserts). Since many of the tweets are missing a Geo location, its fine for the batches of Geo inserts to be smaller than 3000.

```
 # read from local file
start_e = time.time()


fLocal = open("tweetLocal1.txt",'r')
allTweets = fLocal.readlines()


i = 0


BadData=[]
list_twitterData = []
```

```
list_user = []
list_geoData = []
size_insert = 0
tweet_num = 600000

for tweet in allTweets:
    try:
      tdict = json.loads(tweet)
      out_UserData = {x: tdict["user"][x] for x in ["id", "name","screen_name","description","friends_count"]  }
      out_TweetData = {x: tdict[x] for x in ["created_at","id", "id_str","text", "source", "in_reply_to_user_id",
"in_reply_to_screen_name", "in_reply_to_status_id", "retweet_count","contributors"]  }

      # get the Geo Data from .txt file
      """" an example:  geo":{"type":"Point","coordinates":[25.93169839,-80.28004283]}, """"
      if_geoEnabled = {x:tdict["user"][x] for x in["geo_enabled"]}
      if_geo = {x:tdict[x] for x in["geo"]}
      if if_geoEnabled["geo_enabled"] == True and if_geo["geo"] != None:
        list_geo = []
        out_GeoData = {x: if_geo["geo"][x] for x in ["type","coordinates"]}
        #out_GeoData = {x: tdict[x] for x in ["corrdinates"[0],"coordinates"[1]]}
        t_geo = tuple(out_GeoData.values())

        geo_id = out_TweetData["id"]
        geo_created_at = out_TweetData["created_at"]
        list_geo.append(geo_id) # add "id" value
        list_geo.append(geo_created_at) # add "created_at" value
        list_geo.append(t_geo[0]) # add "type" value
        list_geo.append(t_geo[1][0]) # add "cordinates" value
        list_geo.append(t_geo[1][1]) # add "cordinates" value

        list_geoData.append(tuple(list_geo))
```

```
    list_user.append(tuple(out_UserData.values()))
    list_twitterData.append(tuple(out_TweetData.values()))
    size_insert +=1

    if size_insert == 3000 or (size_insert > 0 and i == tweet_num):# insert 3000 rows a time
        stmt1 = "INSERT OR IGNORE INTO UserData (id,name,screen_name,description,friends_count) VALUES (?, ?, ?,
?,?);"
        cursor.executemany(stmt1, list_user)

        stmt2 = "INSERT OR IGNORE INTO TwitterData (created_at, id, id_str, text, source, in_reply_to_user_id,
in_reply_to_screen_name, in_reply_to_status_id, retweet_count, contributors) VALUES (?, ?, ?, ?, ?,?,?,?,?,?);"
        cursor.executemany(stmt2, list_twitterData)

        stmt3 = "INSERT OR IGNORE INTO GeoData (id, created_at, type, lon, lat) VALUES (?, ?, ?, ?,?);"
        cursor.executemany(stmt3, list_geoData)
        print(size_insert)
        size_insert = 0 # reset the size_insert value
        list_twitterData = []
        list_user = []
        list_geoData = []




    i +=1


  except ValueError:
    # Handle the problematic tweet, which in your case would require writing it to another file
    #print (type(tweet))
    BadData.append(tweet)



  if i >= tweet_num:
      #cursor.executemany("INSERT...", batch)
      # save line to the local file
```

```
        break

stmt4 = "SELECT COUNT(*) FROM UserData;"
res = cursor.execute(stmt4)
print(res.fetchall())

stmt5 = "SELECT COUNT(*) FROM TwitterData;"
res = cursor.execute(stmt5)
print(res.fetchall())

stmt6 = "SELECT COUNT(*) FROM GeoData;"
res = cursor.execute(stmt6)
print(res.fetchall())



end_e = time.time()
print("Elapsed time of 1-e: " + str(end_e - start_e) + "seconds")
```

Read from Web:

60000 tweets output:

```
[(57838,)]
[(51447,)]
[(1349,)]
Elapsed time of 1-e: 73.45096707344055seconds
```

600000 tweets output:

```
[(531324,)]
[(507897,)]
[(14477,)]
Elapsed time of 1-e: 789.3154971599579seconds
```

Read from Local File:

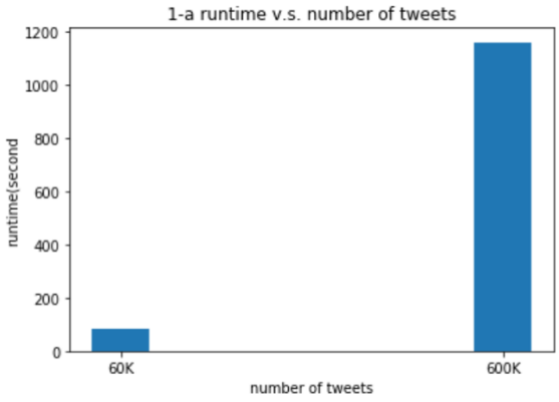
60000 tweets output:

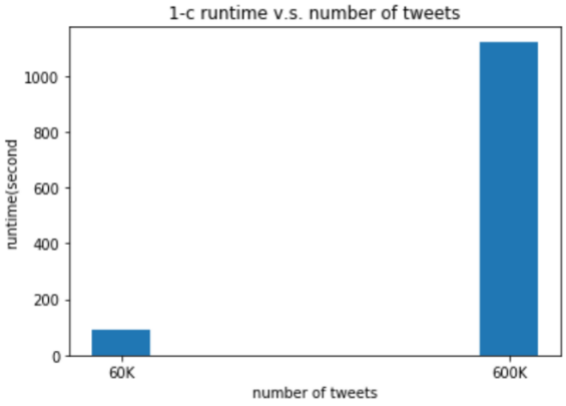
```
[(57838,)]
[(51447,)]
[(1349,)]
Elapsed time of 1-e: 7.813675880432129seconds
```

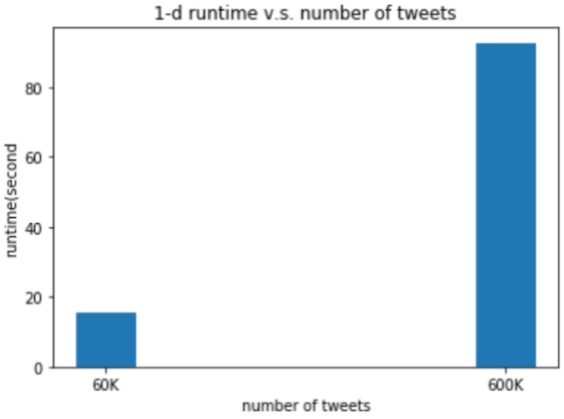
600000 tweets output:

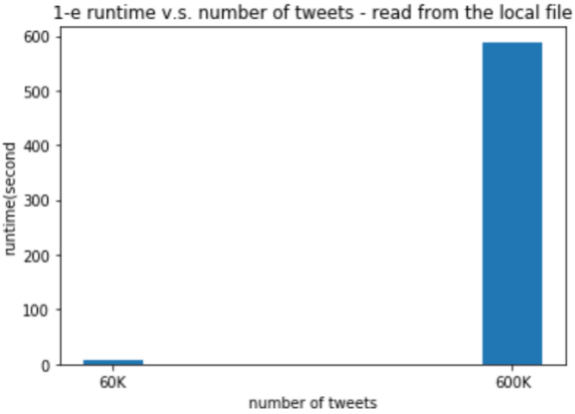
```
[(531324,)]
[(507897,)]
[(14477,)]
Elapsed time of 1-e: 587.0547931194305seconds
```

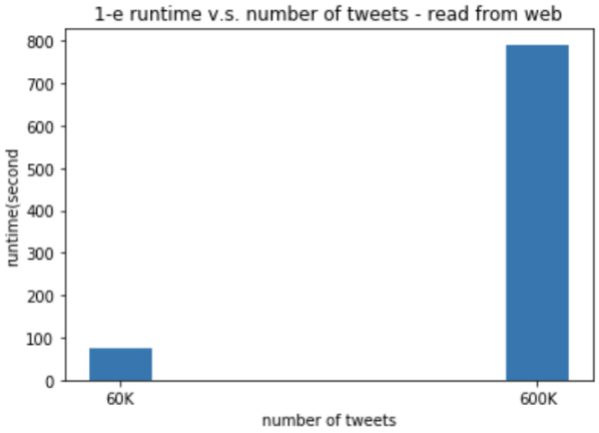
f. Plot the resulting runtimes (# of tweets versus runtimes) using matplotlib for 1-a, 1-c, 1-d, and 1-e. How does the runtime compare?











Part 2

a. Write and execute a SQL query to find the smallest longitude and latitude value for each user ID. This query does not need the User table because User ID is a foreign key in the Tweet table. E.g., something like *SELECT UserID, MIN(longitude), MIN(latitude) FROM Tweet, Geo WHERE Tweet.GeoFK = Geo.GeoID GROUP BY UserID;*

```
stmt7 = " SELECT TwitterData.id, MIN(lon), MIN(lat) FROM TwitterData, GeoData WHERE TwitterData.id =
GeoData.id GROUP BY GeoData.id;"
res = cursor.execute(stmt7)
print(res.fetchall())
```

b. Re-execute the SQL query in part 2-a 10 times and 100 times and measure the total runtime (just re-run the same exact query multiple times using a for-loop, it is as simple as it looks). Does the runtime scale linearly? (i.e., does it take 10X and 100X as much time?)

```
  start_10 = time.time()
for i in range(10):
    stmt8 = " SELECT TwitterData.id, MIN(lon), MIN(lat) FROM TwitterData, GeoData WHERE TwitterData.id = GeoData.id
GROUP BY GeoData.id;"
    res = cursor.execute(stmt8)
    #print(res.fetchall())


end_10 = time.time()
print("Elapsed time of 2-a 10 times: " + str(end_10 - start_10) + "seconds")

start_100 = time.time()
for i in range(100):
    stmt9 = " SELECT TwitterData.id, MIN(lon), MIN(lat) FROM TwitterData, GeoData WHERE TwitterData.id = GeoData.id
GROUP BY GeoData.id;"
    res = cursor.execute(stmt9)
    #print(res.fetchall())


end_100 = time.time()
print("Elapsed time of 2-a 100 times: " + str(end_100 - start_100) + "seconds")
```

Output:

```
Elapsed time of 2-a 10 times: 3.81319618225097 66seconds
Elapsed time of 2-a 100 times: 37.332002162933 35seconds
```

Q: does it take 10X and 100X as much time?

Yes, it is nearly scale linearly.

c. Write the equivalent of the 2-a query in python (without using SQL) by reading it from the file with 600,000 tweets.

```
i = 0

    BadData=[]
    list_twitterData = []
    list_user = []
    list_geoData = []
    tweet_num = 600000

    min_list = {} # key value is geo_id, values are ( min(lat), min(lon) )

    for tweet in allTweets:
        try:
            tdict = json.loads(tweet)
            #out_UserData = {x: tdict["user"][x] for x in ["id", "name","screen_name","description","friends_count"]  }
            out_TweetData = {x: tdict[x] for x in ["created_at","id", "id_str","text", "source", "in_reply_to_user_id",
"in_reply_to_screen_name", "in_reply_to_status_id", "retweet_count","contributors"]  }

            # get the Geo Data from .txt file
            """" an example:  geo":{"type":"Point","coordinates":[25.93169839,-80.28004283]}, """
            if_geoEnabled = {x:tdict["user"][x] for x in["geo_enabled"]}
            if_geo = {x:tdict[x] for x in["geo"]}
            if if_geoEnabled["geo_enabled"] == True and if_geo["geo"] != None:
                list_geo = []
                out_GeoData = {x: if_geo["geo"][x] for x in ["coordinates"]}
                #out_GeoData = {x: tdict[x] for x in ["corrdinates"[0],"coordinates"[1]]}
                t_geo = out_GeoData.values()

                geo_id = out_TweetData["id"]

                #geo_created_at = out_TweetData["created_at"]
                #list_geo.append(geo_id) # add "id" value
                #list_geo.append(geo_created_at) # add "created_at" value
```

```
            #list_geo.append(t_geo[0]) # add "type" value

            t_geo_value = []
            t_geo_value = [tuple(t_geo)[0][0],tuple(t_geo)[0][1]]
            if str(geo_id) not in min_list: #add geo id to unique id list
                min_list[str(geo_id)] = t_geo_value
            else: # this id has record in list, compare the coordinates value
                if t_geo_value[0] < min_list[str(geo_id)][0]:
                    # replace the min value
                    min_list[str(geo_id)][0] = t_geo_value[0]
                if t_geo_value[1] < min_list[str(geo_id)][1]:
                    min_list[str(geo_id)][1] = t_geo_value[1]

        i +=1


    except ValueError:
    # Handle the problematic tweet, which in your case would require writing it to another file
    #print (type(tweet))
        BadData.append(tweet)



    if i >= tweet_num:
        #cursor.executemany("INSERT...", batch)
        # save line to the local file
        break
print(len(min_list))
```

d. Re-execute the query in part 2-c 10 times and 100 times and measure the total runtime. Does the runtime scale linearly?

10 times: 356.63479495048523 seconds
100 times: 3113.652580022812 seconds

e. Write the equivalent of the 2-a query in python by using regular expressions instead of json.loads(). Do not use json.loads() here. Note that you only need to find userid and geo location (if any) for each tweet, you don't need to parse the whole thing.

```
 fLocal = open("tweetLocal.txt",'r')
allTweets = fLocal.readlines()



i = 0

BadData=[]

list_user_geoData = {}

tweet_num = 600000
min_dict = {}
unique_id_list =[]


for tweet in allTweets:
    try:



        if "\"coordinates\":[" in tweet:
            #find geo info "coordinates":[-7.351872,110.213471]},
            start = tweet.find("\"coordinates\":[")
            end = tweet.find("]")
            geo_info = tweet[start:end].replace("\"coordinates\":[", "")
            if "[" in geo_info:
              geo_info = geo_info.replace("[", "")
            print(geo_info)
            cor = geo_info.split(",")
            lan = float(cor[0])
            lon = float(cor[1])
            print(lan,lon)

            #find id
            start = tweet.find("\"id\":")
            end = tweet.find(",\"id_str")
```

```
        id_str = tweet[start:end].replace("\"id\":", '')
        print(id_str)

        # add value to the dict
        if id_str not in min_dict:
          unique_id_list.append(id_str)
          min_dict[id_str] = [lan,lon]
        else:
          if lan < min_dict[id_str][0]:
            min_dict[id_str][0]= lan
          if lon < min_dict[id_str][1]:
            min_dict[id_str][1]= lon




  except ValueError:
    # Handle the problematic tweet, which in your case would require writing it to another file
    #print (type(tweet))
    BadData.append(tweet)



  if i >= tweet_num:
      #cursor.executemany("INSERT...", batch)
      # save line to the local file
      break
  i +=1

print(len(min_dict))
```

f. Re-execute the query in part 2-e 10 times and 100 times and measure the total runtime. Does the runtime scale linearly?

```
Elapsed time 2-e, execute 10 times: 19.905847787857056seconds
Elapsed time 2-e, execute 100 times: 85.60973906517029seconds
```

Part 3

a. Using the database with 600,000 tweets, create a new table that corresponds to the join of all 3 tables in your database, including records without a geo location. This is the equivalent of a materialized view but since SQLite does not support MVs, we will use CREATE TABLE AS SELECT (instead of CREATE MATERIALIZED VIEW AS SELECT).

```
stmt7 = "CREATE Table new_table2 as SELECT  UserData.*, TwitterData.*, GeoData.* FROM UserData LEFT JOIN
TwitterData ON TwitterData.id = UserData.id LEFT JOIN GeoData ON GeoData.id = UserData.id;"
res = cursor.execute(stmt7)
stmt8 = "SELECT COUNT(*) FROM new_table2;"
res = cursor.execute(stmt8)
print(res.fetchall())
```

b. Export the contents of your table from 3-a into a new JSON file (i.e., create your own JSON file with just the keys you extracted). You do not need to replicate the structure of the input and can come up with any reasonable keys for each field stored in JSON structure (e.g., you can have longitude as "longitude" key when the location is available). How does the file size compare to the original input file?

```
stmt8 = "SELECT * FROM new_table2;"
res = cursor.execute(stmt8)
#print(res.fetchall())
items  = [dict(zip([key[0] for key in cursor.description],row)) for row in res]


with open('data.json', 'w') as f:
    json.dump(items,f)
```

The size of the original file is 1.9G
The size of the new file exported is 283MB

c. Export the contents of your table from 3-a into a .csv (comma separated value) file. How does the file size compare to the original input file and to the file in 3-b?

```
db_df = pd.read_sql_query("SELECT * FROM new_table2", conn)
db_df.to_csv('database.csv', index=False)
```

The size of the new file exported is 283MB
.csv file size: 68.6MB, which is much smaller than the json file.