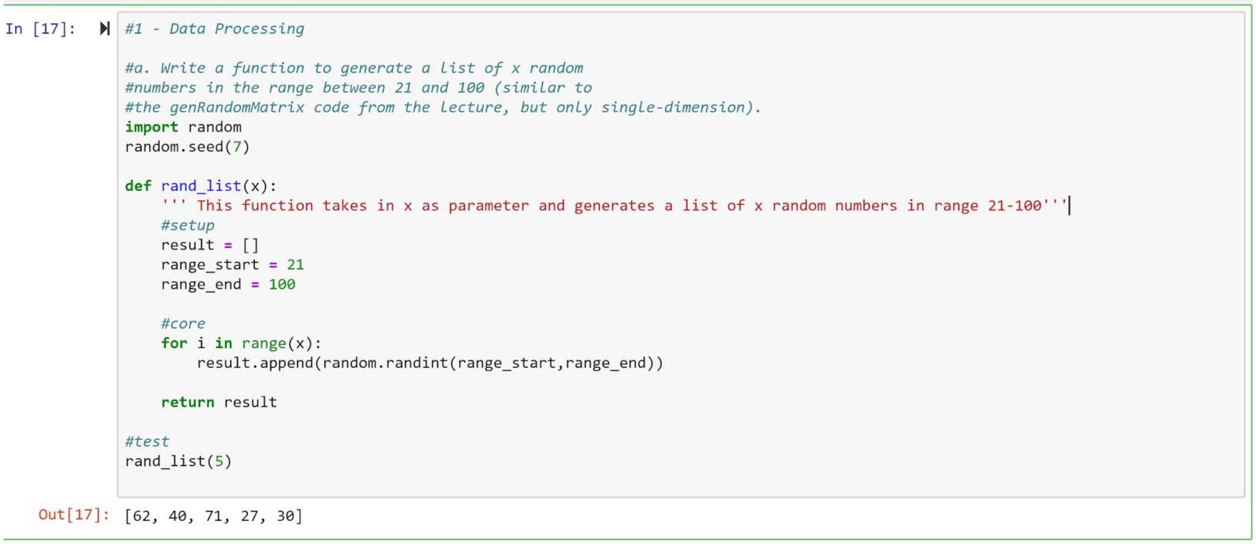
Alex Teboul
Assignment 4

CSC 455: Database Processing for Large-Scale Analytics

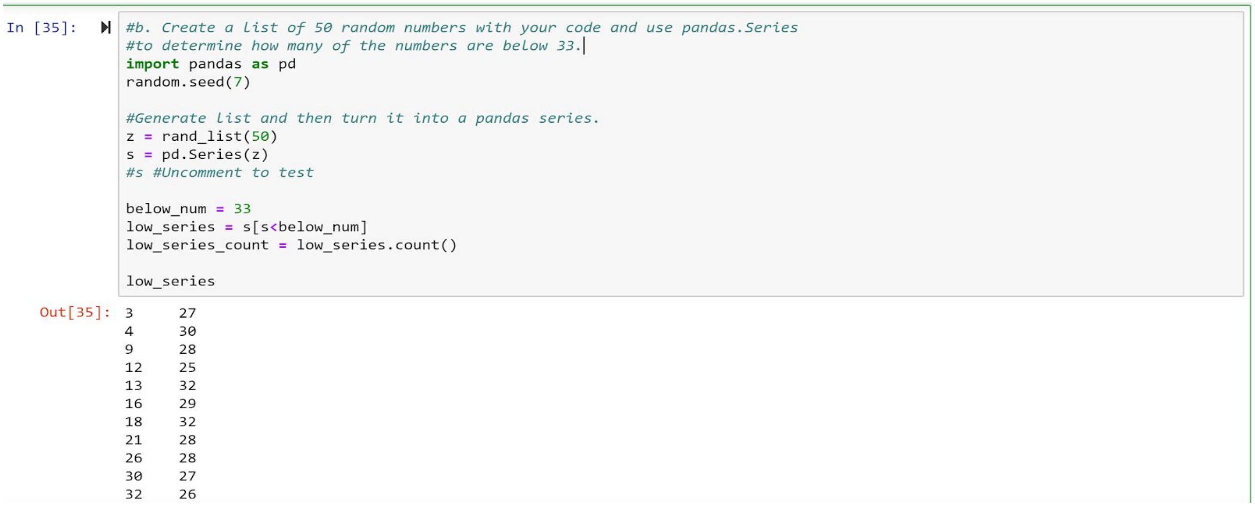
1. Data processing

a. Write a function to generate a list of x random numbers in the range between 21 and 100 (similar to the genRandomMatrix code from the lecture, but only single-dimension).

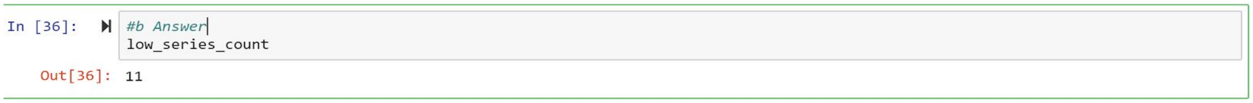


```
In [17]:  #1 - Data Processing

          #a. Write a function to generate a list of x random
          #numbers in the range between 21 and 100 (similar to
          #the genRandomMatrix code from the lecture, but only single-dimension).
          import random
          random.seed(7)

          def rand_list(x):
              ''' This function takes in x as parameter and generates a list of x random numbers in range 21-100'''
              #setup
              result = []
              range_start = 21
              range_end = 100

              #core
              for i in range(x):
                  result.append(random.randint(range_start,range_end))

              return result

          #test
          rand_list(5)

Out[17]:  [62, 40, 71, 27, 30]
```

b. Create a list of 50 random numbers with your code and use pandas.Series to determine how many of the numbers are below 33.

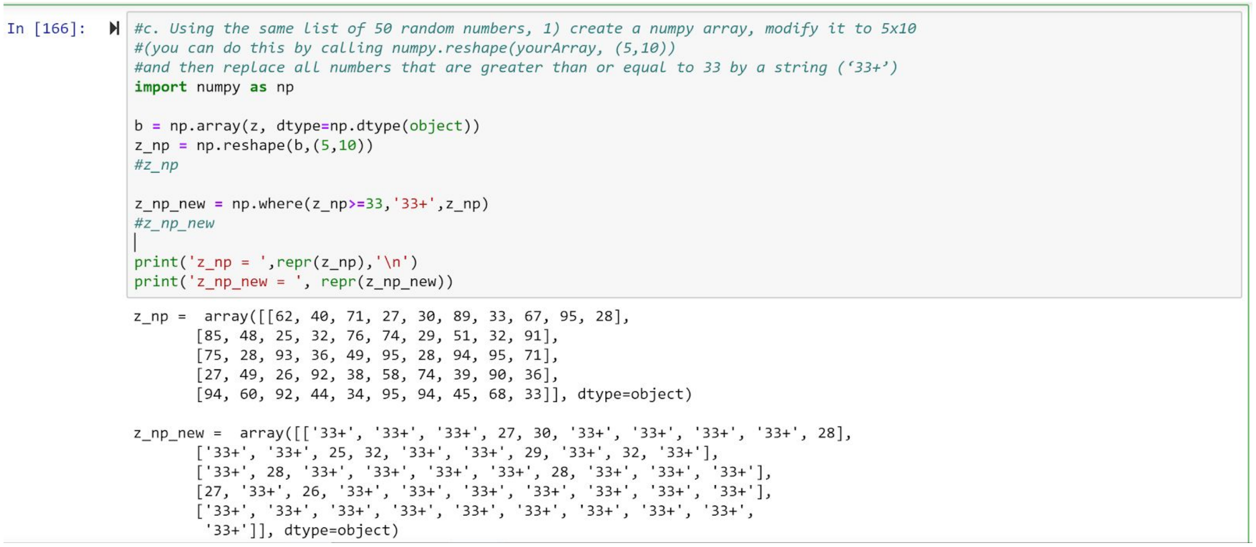


```
In [35]:  #b. Create a list of 50 random numbers with your code and use pandas.Series
          #to determine how many of the numbers are below 33.
          import pandas as pd
          random.seed(7)

          #Generate list and then turn it into a pandas series.
          z = rand_list(50)
          s = pd.Series(z)
          #s #Uncomment to test

          below_num = 33
          low_series = s[s<below_num]
          low_series_count = low_series.count()

          low_series

Out[35]:  3     27
          4     30
          9     28
          12    25
          13    32
          16    29
          18    32
          21    28
          26    28
          30    27
          32    26
```

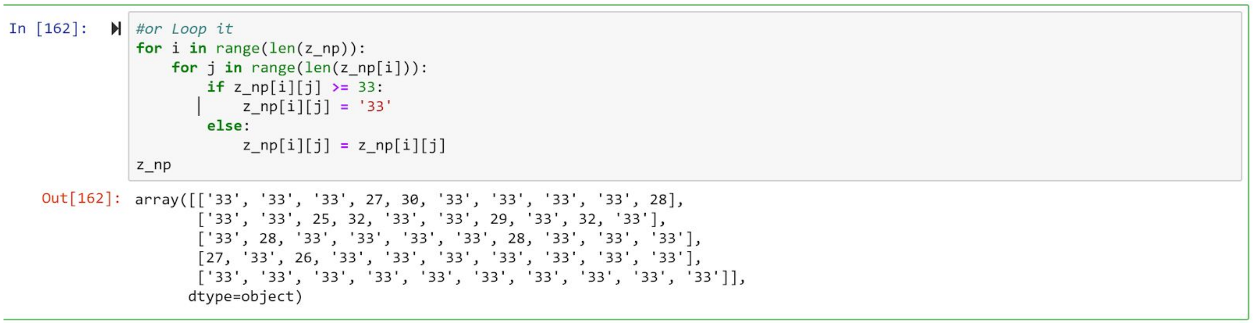


```
In [36]:  #b Answer
          low_series_count

Out[36]:  11
```

c. Using the same list of 50 random numbers, 1) create a numpy array, modify it to 5x10 (you can do this by calling numpy.reshape(yourArray, (5,10)) and then replace all numbers that are greater than or equal to 33 by a string ('33+')

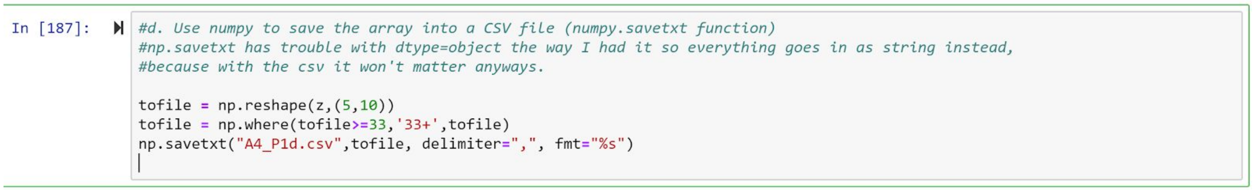


```
In [166]:  #c. Using the same list of 50 random numbers, 1) create a numpy array, modify it to 5x10
           #(you can do this by calling numpy.reshape(yourArray, (5,10))
           #and then replace all numbers that are greater than or equal to 33 by a string ('33+')
           import numpy as np

           b = np.array(z, dtype=np.dtype(object))
           z_np = np.reshape(b,(5,10))
           #z_np

           z_np_new = np.where(z_np>=33,'33+',z_np)
           #z_np_new

           print('z_np = ',repr(z_np),'\n')
           print('z_np_new = ', repr(z_np_new))

           z_np =  array([[62, 40, 71, 27, 30, 89, 33, 67, 95, 28],
                  [85, 48, 25, 32, 76, 74, 29, 51, 32, 91],
                  [75, 28, 93, 36, 49, 95, 28, 94, 95, 71],
                  [27, 49, 26, 92, 38, 58, 74, 39, 90, 36],
                  [94, 60, 92, 44, 34, 95, 94, 45, 68, 33]], dtype=object)

           z_np_new =  array([['33+', '33+', '33+', 27, 30, '33+', '33+', '33+', '33+', 28],
                  ['33+', '33+', 25, 32, '33+', '33+', 29, '33+', 32, '33+'],
                  ['33+', 28, '33+', '33+', '33+', '33+', 28, '33+', '33+', '33+'],
                  [27, '33+', 26, '33+', '33+', '33+', '33+', '33+', '33+', '33+'],
                  ['33+', '33+', '33+', '33+', '33+', '33+', '33+', '33+', '33+',
                   '33+']], dtype=object)
```

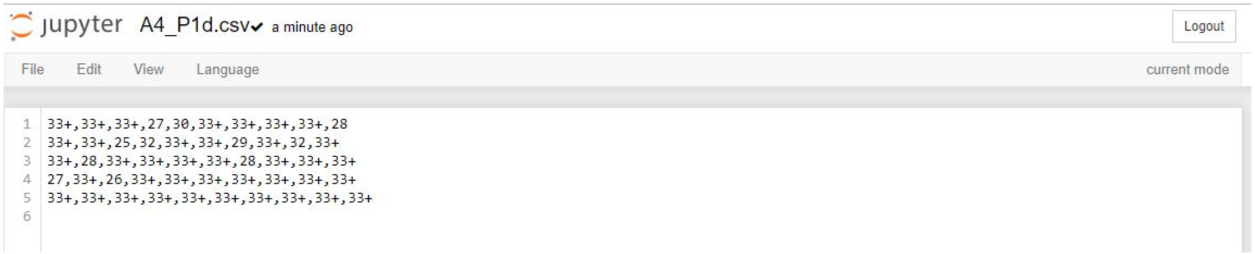


```
In [162]:  #or Loop it
           for i in range(len(z_np)):
               for j in range(len(z_np[i])):
                   if z_np[i][j] >= 33:
                       z_np[i][j] = '33'
                   else:
                       z_np[i][j] = z_np[i][j]
           z_np

Out[162]:  array([['33', '33', '33', 27, 30, '33', '33', '33', '33', 28],
                  ['33', '33', 25, 32, '33', '33', 29, '33', 32, '33'],
                  ['33', 28, '33', '33', '33', '33', 28, '33', '33', '33'],
                  [27, '33', 26, '33', '33', '33', '33', '33', '33', '33'],
                  ['33', '33', '33', '33', '33', '33', '33', '33', '33', '33']],
                 dtype=object)
```

d. Use numpy to save the array into a CSV file (numpy.savetxt function)

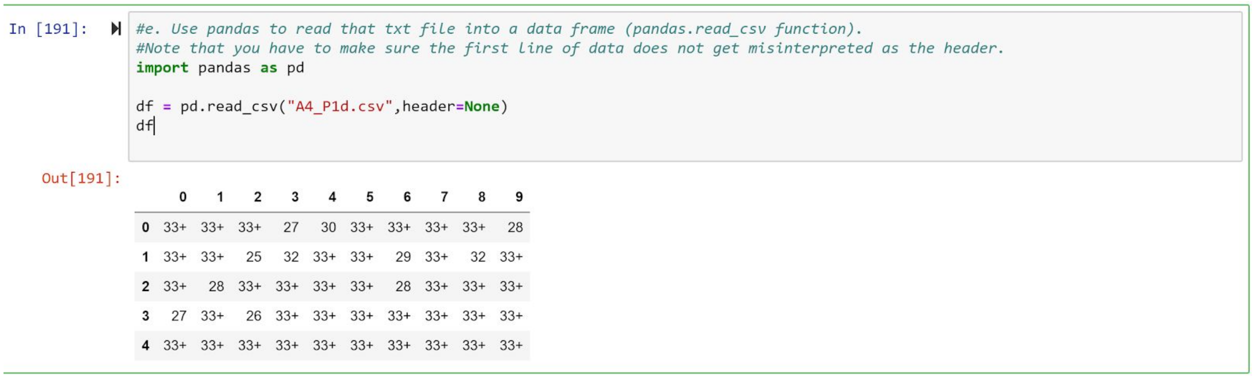


```
In [187]:  #d. Use numpy to save the array into a CSV file (numpy.savetxt function)
           #np.savetxt has trouble with dtype=object the way I had it so everything goes in as string instead,
           #because with the csv it won't matter anyways.

           tofile = np.reshape(z,(5,10))
           tofile = np.where(tofile>=33,'33+',tofile)
           np.savetxt("A4_P1d.csv",tofile, delimiter=",", fmt="%s")
```



```
33+,33+,33+,27,30,33+,33+,33+,33+,28
33+,33+,25,32,33+,33+,29,33+,32,33+
33+,28,33+,33+,33+,33+,28,33+,33+,33+
27,33+,26,33+,33+,33+,33+,33+,33+,33+
33+,33+,33+,33+,33+,33+,33+,33+,33+,33+
```

e. Use pandas to read that txt file into a data frame (pandas.read_csv function). Note that you have to make sure the first line of data does not get misinterpreted as the header.



In [191]:

```
#e. Use pandas to read that txt file into a data frame (pandas.read_csv function).
#Note that you have to make sure the first line of data does not get misinterpreted as the header.
import pandas as pd

df = pd.read_csv("A4_P1d.csv",header=None)
df
```

Out[191]:

	0	1	2	3	4	5	6	7	8	9
0	33+	33+	33+	27	30	33+	33+	33+	33+	28
1	33+	33+	25	32	33+	33+	29	33+	32	33+
2	33+	28	33+	33+	33+	33+	28	33+	33+	33+
3	27	33+	26	33+	33+	33+	33+	33+	33+	33+
4	33+	33+	33+	33+	33+	33+	33+	33+	33+	33+

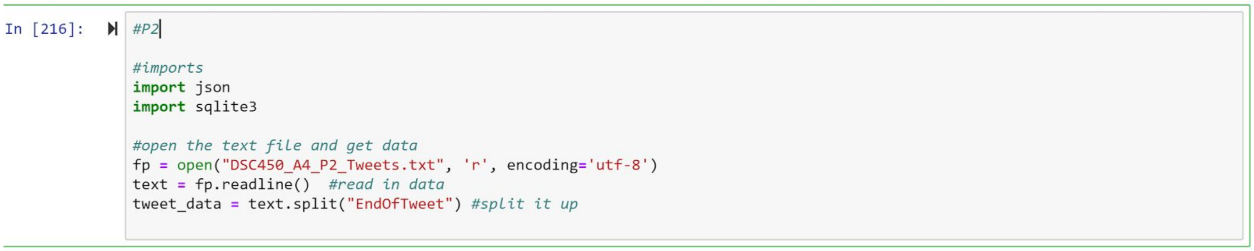
2. We are going to work with a small extract of tweets (about 200 of them), available here: http://rasinsrv07.cstcis.cti.depaul.edu/CSC455/Assignment4.txt

NOTE 1: I do not recommend trying to copy-paste this text, because there is absolutely no knowing what might come out from paste on your system. You should be able to use "Save as…" function in your browser.

NOTE 2: The input data is separated by a string "EndOfTweet" which serves as a delimiter. The text itself consists of a single line, so using readline() or readlines() will still only give you one row which needs to be split by the delimiter.

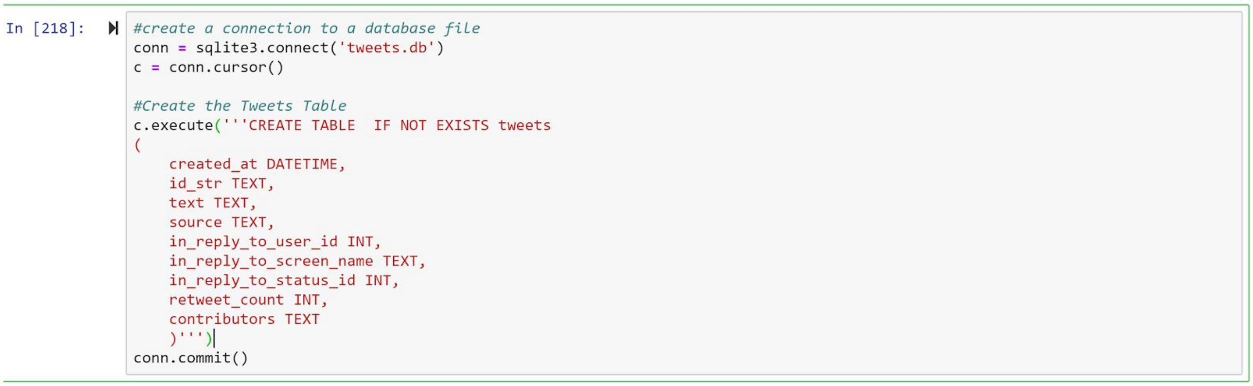
a. Create a SQL table to contain the following attributes of a tweet:

"created_at",
"id_str",
"text",
"source",
"in_reply_to_user_id",
"in_reply_to_screen_name",
"in_reply_to_status_id",
"retweet_count",
"contributors".
Please assign reasonable data types to each attribute and use SQLite for this assignment.



In [216]:

```
#P2

#imports
import json
import sqlite3

#open the text file and get data
fp = open("DSC450_A4_P2_Tweets.txt", 'r', encoding='utf-8')
text = fp.readline()  #read in data
tweet_data = text.split("EndOfTweet") #split it up
```

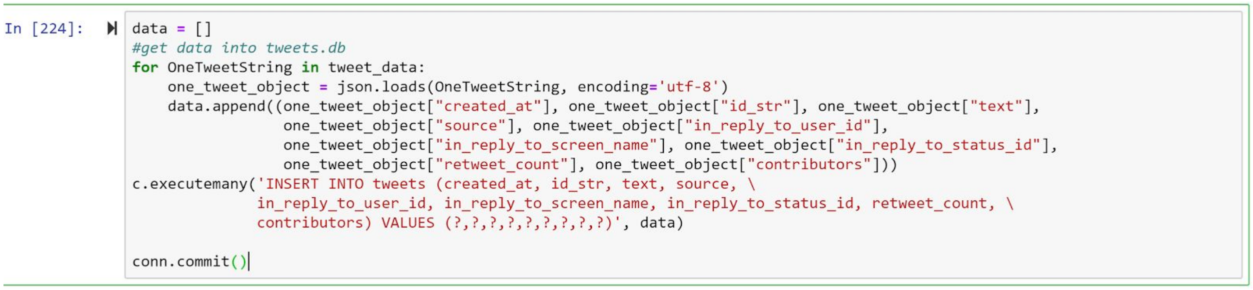


In [218]:

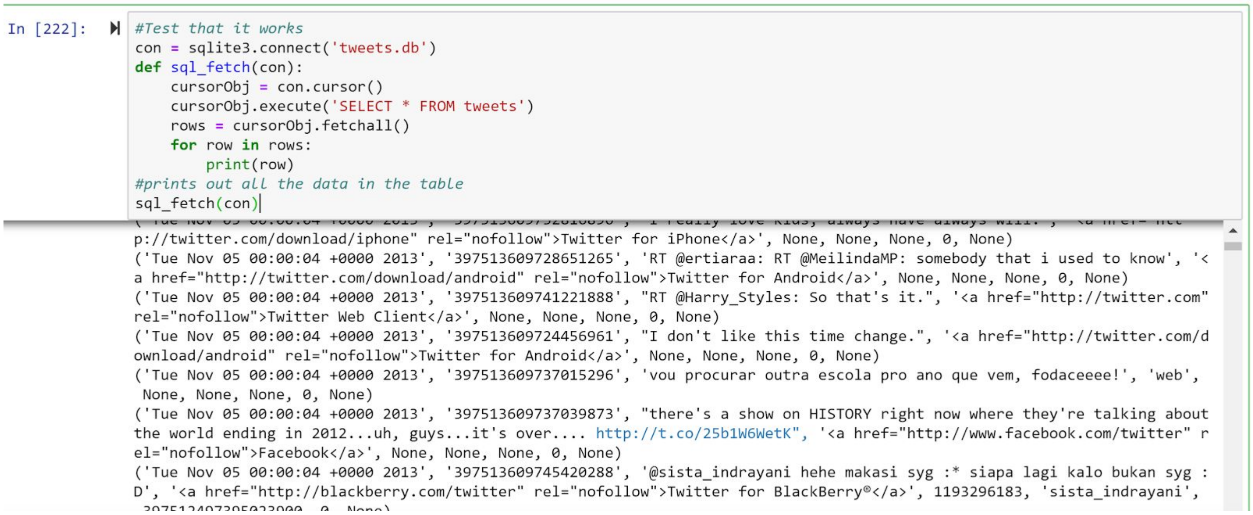
```
#create a connection to a database file
conn = sqlite3.connect('tweets.db')
c = conn.cursor()

#Create the Tweets Table
c.execute('''CREATE TABLE  IF NOT EXISTS tweets
(
    created_at DATETIME,
    id_str TEXT,
    text TEXT,
    source TEXT,
    in_reply_to_user_id INT,
    in_reply_to_screen_name TEXT,
    in_reply_to_status_id INT,
    retweet_count INT,
    contributors TEXT
    )''')
conn.commit()
```

b. Write python code to read through the Assignment4.txt file and populate your table from part a. Make sure your python code reads through the file and loads the data properly (including NULLs).

```
# importing the libraries
import json
import sqlite3

# opening the file
f = open("DSC450_A4_P2_Tweets.txt", 'r', encoding='utf-8')
text = f.readline()  # read all at once
tweet_data = text.split("EndOfTweet")  # split by delimiter

# creating connection to database file
conn = sqlite3.connect('tweetdata.db')
c = conn.cursor()

# Create table tweets
c.execute('''CREATE TABLE  IF NOT EXISTS tweet

(
          created_at DATETIME,
          id_str TEXT,
          text TEXT,
          source TEXT,
          in_reply_to_user_id INT,
          in_reply_to_screen_name TEXT,
          in_reply_to_status_id INT,
          retweet_count INT,
          contributors TEXT

)''')

data = []
# making data set for inserting in bulk

for OneTweetString in tweet_data:
          one_tweet_object = json.loads(OneTweetString, encoding='utf-8')
          data.append((one_tweet_object["created_at"], one_tweet_object["id_str"],
                  one_tweet_object["text"], one_tweet_object["source"],
                  one_tweet_object["in_reply_to_user_id"],
                  one_tweet_object["in_reply_to_screen_name"],
                  one_tweet_object["in_reply_to_status_id"],
                  one_tweet_object[“retweet_count”],
                  one_tweet_object[“contributors”]))

c.executemany('INSERT INTO tweet (created_at, id_str, text, source, in_reply_to_user_id,
in_reply_to_screen_name, in_reply_to_status_id) VALUES (?,?,?,?,?,?,?)', data)

conn.commit()
conn.close()
```

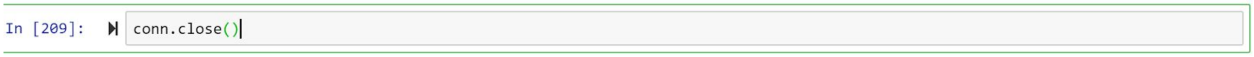


```
In [224]:
data = []
#get data into tweets.db
for OneTweetString in tweet_data:
    one_tweet_object = json.loads(OneTweetString, encoding='utf-8')
    data.append((one_tweet_object["created_at"], one_tweet_object["id_str"], one_tweet_object["text"],
                 one_tweet_object["source"], one_tweet_object["in_reply_to_user_id"],
                 one_tweet_object["in_reply_to_screen_name"], one_tweet_object["in_reply_to_status_id"],
                 one_tweet_object["retweet_count"], one_tweet_object["contributors"]))
c.executemany('INSERT INTO tweets (created_at, id_str, text, source, \
              in_reply_to_user_id, in_reply_to_screen_name, in_reply_to_status_id, retweet_count, \
              contributors) VALUES (?,?,?,?,?,?,?,?,?)', data)

conn.commit()
```



```
In [222]:  #Test that it works
           con = sqlite3.connect('tweets.db')
           def sql_fetch(con):
               cursorObj = con.cursor()
               cursorObj.execute('SELECT * FROM tweets')
               rows = cursorObj.fetchall()
               for row in rows:
                   print(row)
           #prints out all the data in the table
           sql_fetch(con)
```

```
('Tue Nov 05 00:00:04 +0000 2013', '397513609732816896', 'I really love kids, always have always will.', '<a href="htt
p://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>', None, None, None, 0, None)
('Tue Nov 05 00:00:04 +0000 2013', '397513609728651265', 'RT @ertiaraa: RT @MeilindaMP: somebody that i used to know', '<
a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>', None, None, None, 0, None)
('Tue Nov 05 00:00:04 +0000 2013', '397513609741221888', "RT @Harry_Styles: So that's it.", '<a href="http://twitter.com"
rel="nofollow">Twitter Web Client</a>', None, None, None, 0, None)
('Tue Nov 05 00:00:04 +0000 2013', '397513609724456961', "I don't like this time change.", '<a href="http://twitter.com/d
ownload/android" rel="nofollow">Twitter for Android</a>', None, None, None, 0, None)
('Tue Nov 05 00:00:04 +0000 2013', '397513609737015296', 'vou procurar outra escola pro ano que vem, fodaceeee!', 'web',
 None, None, None, 0, None)
('Tue Nov 05 00:00:04 +0000 2013', '397513609737039873', "there's a show on HISTORY right now where they're talking about
the world ending in 2012...uh, guys...it's over.... http://t.co/25b1W6WetK", '<a href="http://www.facebook.com/twitter" r
el="nofollow">Facebook</a>', None, None, None, 0, None)
('Tue Nov 05 00:00:04 +0000 2013', '397513609745420288', '@sista_indrayani hehe makasi syg :* siapa lagi kalo bukan syg :
D', '<a href="http://blackberry.com/twitter" rel="nofollow">Twitter for BlackBerry®</a>', 1193296183, 'sista_indrayani',
 397512497395023900, 0, None)
```



```
In [209]:  conn.close()
```

3. Write SQL queries to do the following:

a. Count the number of iPhone users (based on “source” attribute)

c.execute("SELECT count(*) FROM tweets WHERE source LIKE '%iPhone%' ").fetchall()

#test that it works
acheck = c.execute("SELECT * FROM tweets WHERE source LIKE '%iPhone%' ").fetchall()
len(acheck)

b. Create a view that contains only tweets from users who are not replying ("in_reply_to_user_id" is NULL)

c.execute("CREATE VIEW b AS SELECT * FROM tweets WHERE in_reply_to_user_id IS NULL ").fetchall()

#test that it works
c.execute("select * from b").fetchall()

c. Select tweets that have a "retweet_count" higher than the average "retweet_count" from the tweets in the view in part b

c.execute("SELECT * FROM b WHERE retweet_count > (SELECT AVG(retweet_count) FROM tweets)").fetchall()

*all tweets have a max of 0 so.. no tweet had a retweet_count higher than the avg.

d. Create a view that contains only “id_str”, “text” and “source” from each tweet that has a “retweet_count” of at least 5

```
c.execute("CREATE VIEW partd AS SELECT id_str,text,source FROM b WHERE
retweet_count >= 5 ").fetchall()

#Test that it works
c.execute("SELECT * FROM partd").fetchall()
```

e. Use the view from part-d to find how many tweets have a “retweet_count” of at least 5

```
 c.execute("SELECT count(*) FROM partd").fetchall()
```

f. Write python code to compute the answer from 3-e without using SQL, i.e., write code that is going to read data from the input file and answer the same question (find how many tweets have a “retweet_count” of at least 5).

```
import pandas as pd
import json

f = open("Assignment4.txt", 'r', encoding='utf-8')
text = f.readline()  # read all at once
tweet_data = text.split("EndOfTweet")  # split by delimiter

#test
tweet_data

data = []

for tweet in tweet_data:
        a_tweet = json.loads(tweet, encoding='utf-8')
        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"],
```

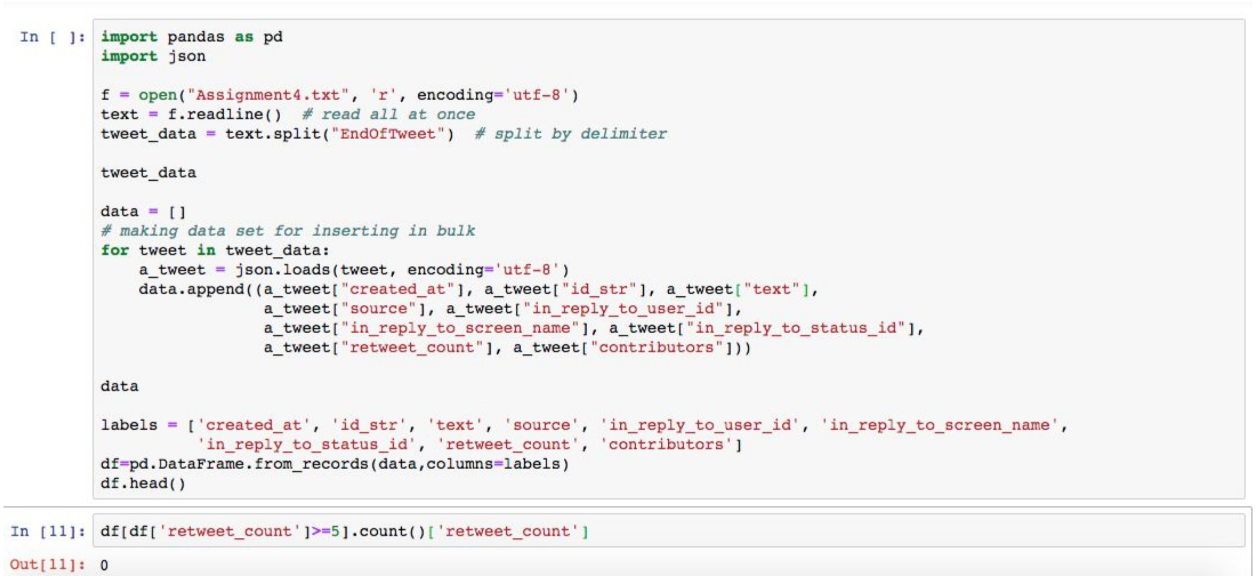
a_tweet["in_reply_to_status_id"], a_tweet["retweet_count"],
a_tweet["contributors"]))

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']

df=pd.DataFrame.from_records(data,columns=labels)
df.head()

df[df['retweet_count']>=5].count()['retweet_count'] #counter

#count=0 - there are no tweets with at least 5 retweets in the dataset



```
In [ ]: import pandas as pd
        import json

        f = open("Assignment4.txt", 'r', encoding='utf-8')
        text = f.readline()  # read all at once
        tweet_data = text.split("EndOfTweet")  # split by delimiter

        tweet_data

        data = []
        # making data set for inserting in bulk
        for tweet in tweet_data:
            a_tweet = json.loads(tweet, encoding='utf-8')
            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"]))

        data

        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']
        df=pd.DataFrame.from_records(data,columns=labels)
        df.head()

In [11]: df[df['retweet_count']>=5].count()['retweet_count']

Out[11]: 0
```

4. Write a python function that takes the name of a SQL table as parameter and then does the following:
Select all rows from that table (you can assume that the table already exists in SQLite) with all attributes from that table and output to a file a sequence of corresponding INSERT statements, one for each row from the table. Think of this as an export tool, since these INSERT statements could now be executed in Oracle (you do not need to actually execute them in Oracle that).

For example: generateInsertStatements('Students') should write to a file an insert statement from each row contained in the Students table (assuming the table is in SQLite already)

inserts.txt:

```
INSERT INTO Students VALUES (‘1’, ‘Jane’, ‘A-‘);
INSERT INTO Students VALUES (‘1’, ‘Mike’, ‘B‘);
INSERT INTO Students VALUES (‘1’, ‘Jack’, ‘B+‘);
```

Hint: as you iterate through the rows of the given table, instead of printing the output, you will want to write an INSERT SQL statement to an output file each time.

```
#imports
import sqlite3
import os

def generateInsertStatements(table):


    database = table + ’.db’                 # or whatever the name is for the database
    c = sqlite3.connect(database)                 #connect to the db

    query = 'SELECT * FROM ' + table              #query all
    cursor = c.execute(query)                     #executing sql statement

    #open the file
    inserts = open('inserts.txt', 'w')            #output file for inserts

    #iterate through each record
    for record in cursor:

        #creating insert statement
        insert_statement = 'INSERT INTO ' + table + ' VALUES ('

        #for all columns in the row
        for column in record:
            insert_statement = insert_statement + "'" + column + "',"

        insert_statement = insert_statement.rstrip(',') + ');\n'
        insert.write(insert_statement)                        #write it into a file

c.close()        #close the connection
inserts.close()  #close the inserts file
```

5. Write a PL/SQL trigger that will cap the course number column in the university.sql database at 599. That is, any time an update or an insert would provide course number 600 or higher, automatically reset the value back to 599. Be sure to verify that your trigger is working with some sample data.

```
CREATE OR REPLACE TRIGGER course_cap_trigger
        BEFORE INSERT OR UPDATE
        ON course
        FOR EACH ROW
BEGIN
        IF new.CourseNr >599
        THEN
                new.CourseNr:=599;
        END IF;
END;
#checks out

#Tables for reference from university.sql:

create table student (

  LastName      varchar(40),

  FirstName     varchar(40),

  SID          number(5),

  SSN           number(9),

  Career        varchar(4),

  Program       varchar(10),

  City         varchar(40),

  Started       number(4),


  primary key (SID),

  unique(SSN)
);


create table course (
  CID   number(4),
  CourseName    varchar(40),
  Department    varchar(4),
  CourseNr       char(3),


  primary key (CID)
);


create table studentgroup (
  GID              number(5),
  Name             varchar(40),
  PresidentID   number(5),
  Founded       number(4),


  primary key (GID),
  unique (Name),
  foreign key (PresidentID) references student(SID)
);
```

```
create table enrolled (
  StudentID     number(5),
  CourseID      number(4),
  Quarter       varchar(6),
  Year          number(4),

  primary key (StudentID, CourseID),
  foreign key (StudentID) references student(SID),
  foreign key (CourseID) references course(CID)
);

create table memberof (
  StudentID     number(5),
  GroupID       number(5),
  Joined        number(4),

  primary key (StudentID, GroupID),
  foreign key (StudentID) references student(SID),
  foreign key (GroupID) references studentgroup(GID)
);
```