File Systems Forensics and Analysis (6G5Z1107)

Portfolio 1

Karan Nihalani, ID: 17023122

Contents

TASK 1

(a) Identify the names of files (excluding deleted files), if any, and their extensions and how you identified them;

One file I located was “mydetails.doc”, in the byte range 11, which is a file attribute and therefore, I identified it as a long file name, through a basic directory entry. Another file I had located is “myparentspic.jpg”, in which I used the same method, identifying it as a long file name, through the basic directory entry.

(b) Identify the names of directories, if any, and how you identified them;

Only one name of a directory was found, and it is called, “goodstuff”. I was able to identify this directory through the byte range 11 where it appeared as “10”, via the basic directory entry.

(c) Identify the names of deleted files, if any, including their extensions and how you identified them;

One deleted file was located, as well. Its name is “YPHOTO JPG” and I found out that it is a deleted file, because the first character in the 32-byte line is an “E” (“E5”).

(d) How can you identify a long file name in the root directory?

The method to identify a long file name in the root directory is to observe the file attributes column in byte 11 and utilize the Flag Value table to identify whether the data is a long file name or not, where in this case, the Flag value (in bits) is 0001 0000 (0x10).

(e) What is the maximum number of characters in a short file name excluding the extension based on the file system example provided?

Excluding the extension based on the file system example provided, the maximum number of characters in a short file name is eight characters long, whereas the total length would be 11 characters long with the extension.

TASK 2 - Methods used and Walkthrough

Creating the Python Graphical User Interface (GUI)

It was necessary to create a user interface, with the purpose of being able to complete various different searches, within a JSON file called 'Manchester_Part-1_records.json', including map plotting and regular expression searching, via the Python-supported application, *Spyder*. We had to begin with creating an instance called *"win = tk.Tk()"* and it was required to even add a title to this window. This is made possible due to the 'import' package *"import tkinter as tk"*, which is defined as the "standard Python interface to the GUI toolkit" (The Python Software Foundation, 2018). We can observe in **Fig.1** that the window has been set a maximum size and minimum size, along with *"win.resizable (0,0)"*, so that the window is unable to resize itself. In Fig.1 the different 'import' packages needed to perform the different searches, such as *"import json"*, are also indicated. We will go more in depth into these as we move on further.

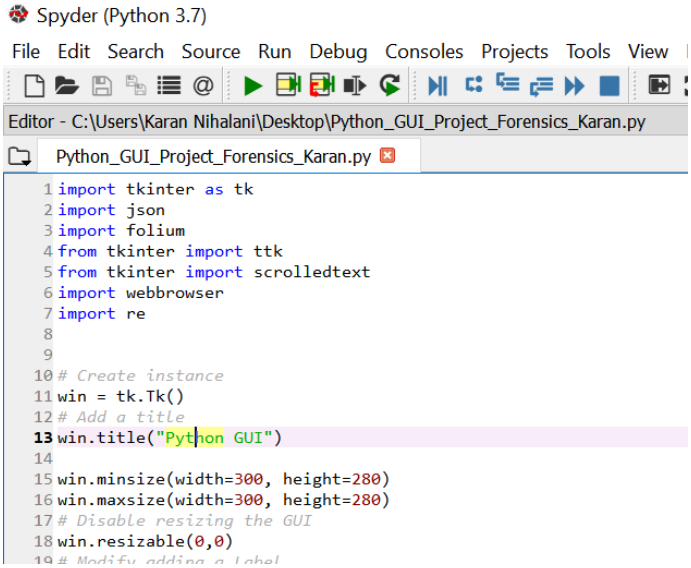

Fig.1

Simple Text Search and Map-Plotting

Simple Text Search

As what has been mentioned and shown previously, we had to make use of the import package *"import json"*, where the program would have to read the file containing the records needed for the simple text search, at least. The first essential step was to define a function which could carry out the simple text search and in this case, it has been named, *"def clickMe()"*. Within this function, the local variable *"searchkeyword"* is assigned, to be managed further on in the code, as viewed on below.

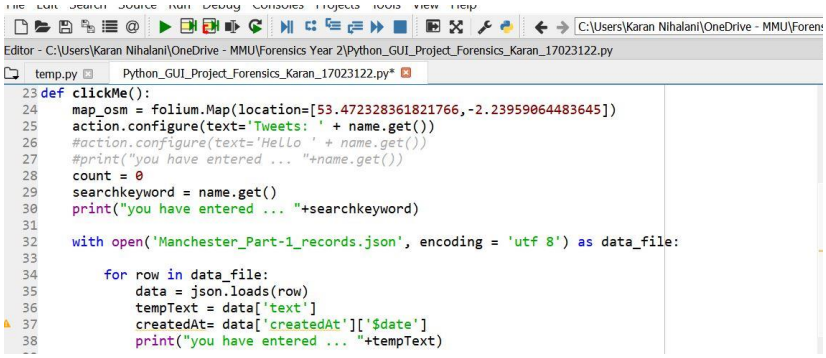

Fig.2

This local variable establishes the user's input or word to search for and is therefore utilized in an *"if"* statement, at which point the newly created program searches through all the data in the file and is able to pick out the tweet containing that same word. A button is also created (**Fig. 2.1**), so that when a user clicks it, they can visualize the tweets for the word searched inside the GUI. An example with the word "the" has been carried out demonstrating this. (**Fig. 2.2**).

```
 95
 96
 97 # Changing our Label
 98 ttk.Label(win, text="Enter a name:").grid(column=0, row=0)
 99 # Adding a Textbox Entry widget
100 name = tk.StringVar()
101 nameEntered = ttk.Entry(win, width=12, textvariable=name)
102 nameEntered.grid(column=0, row=1)
103 # Adding a Button
104 action = ttk.Button(win, text="Click for Tweet!", command=clickMe)
105 action.grid(column=2, row=1)
106
```

Fig. 2.1

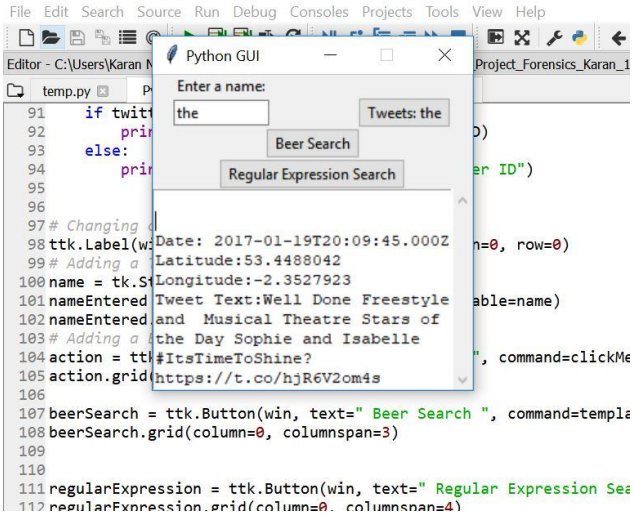

Fig. 2.2

In addition to all of this, "print" statements were typed within the code, as it aided us in clarifying the data and data types in every single tweet, such as the Date the text was tweeted, the Latitude and Longitude from where the text was tweeted from and finally, the tweet text itself. See **Fig. 2.3**.

```
print("you have entered ... "+tempText)

if searchkeyword in tempText:
    StringToScroll = "\n" + "\nDate: " + data['createdAt']['$date'] + "\nLatitude:" +
    count = count + 1
    scr.insert(tk.INSERT,StringToScroll)
    latt = data['geoLocation']['latitude']
    long = data['geoLocation']['longitude']
    folium.Marker([latt, long], popup=tempText, icon=folium.Icon(color='blue',icon='tw
    #print("Geo-Location "+str(data['geoLocation']['latitude']))
    print("Tweet Text: "+data['text'])
    print("Place Name: "+data['place']['name'])
    print("Place Full Name: "+data['place']['fullName'])
    print(" ..... Next Record .....")
    map_osm.save('plotted-map.html')
```

Fig. 2.3

Map-Plotting

With regard to plotting the selected tweets on a map, we had to primarily install 'folium' in our computers through the command line. The import package, *"import folium"* (**Fig. 1**), is placed at the beginning of the code along with all of the other import libraries. This library is what gives us permission to "create several types of Leaflet maps" in Python (Domino Data Lab, 2015) and thus opens up or creates an HTML webpage with the selected plotted points.

Looking into the syntax typed in the code, we've provided the co-ordinates of the location where all the tweets are placed in the city of Manchester, UK. Right before the *"print"* statements mentioned above, a marker called *"folium.Marker"* is exploited, marking the tweet selected on the map itself. This is demonstrated through the code in **Fig. 3** and through the HTML page in **Fig.4**.

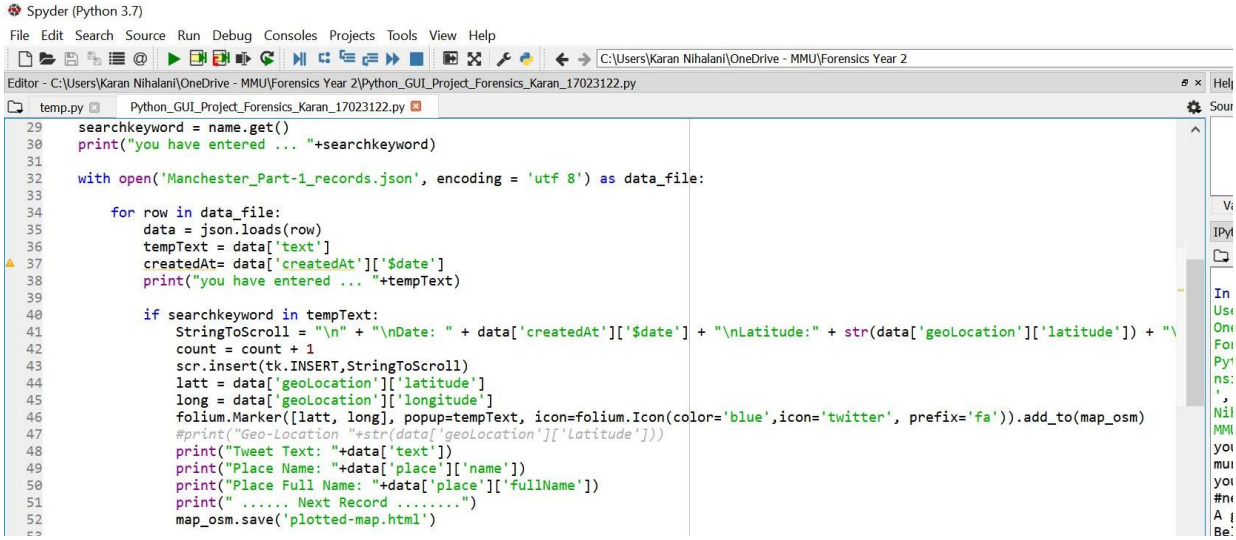

Fig. 3

Along line 46 in **Fig. 3**, it is evident that a Twitter icon for the tweet plotted is selected and is evidently displayed, as viewed in the example for the word, "muralswallpaper" (**Fig. 4**). On that same line of code, *"popup = tempText"* is applied to specify the fact that the tweet text shows up on the user's screen if the pinned location is clicked (**Fig. 5**).

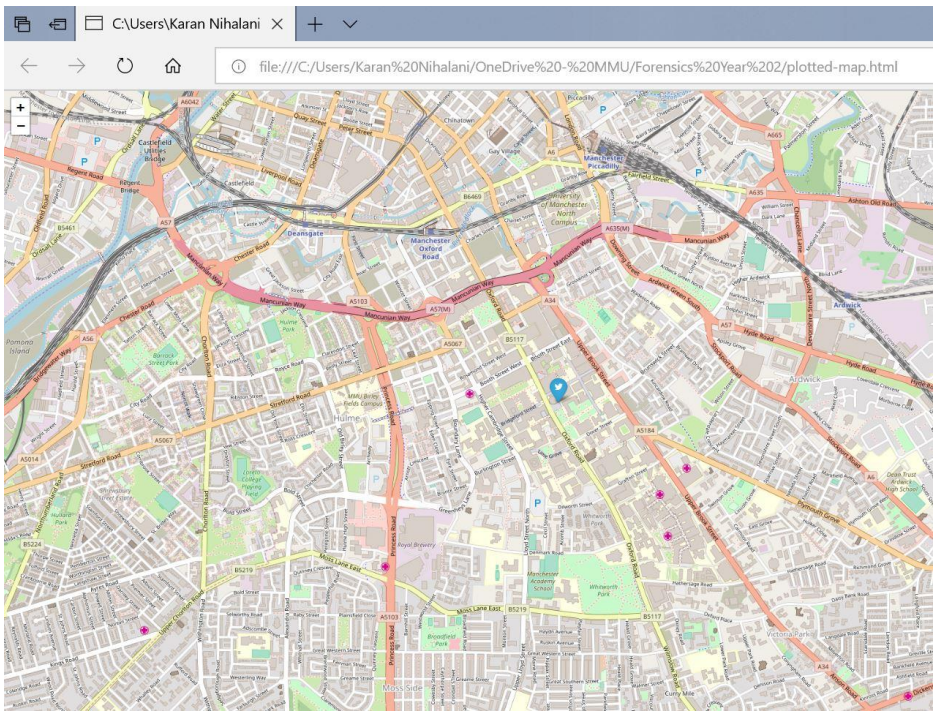

Fig. 4

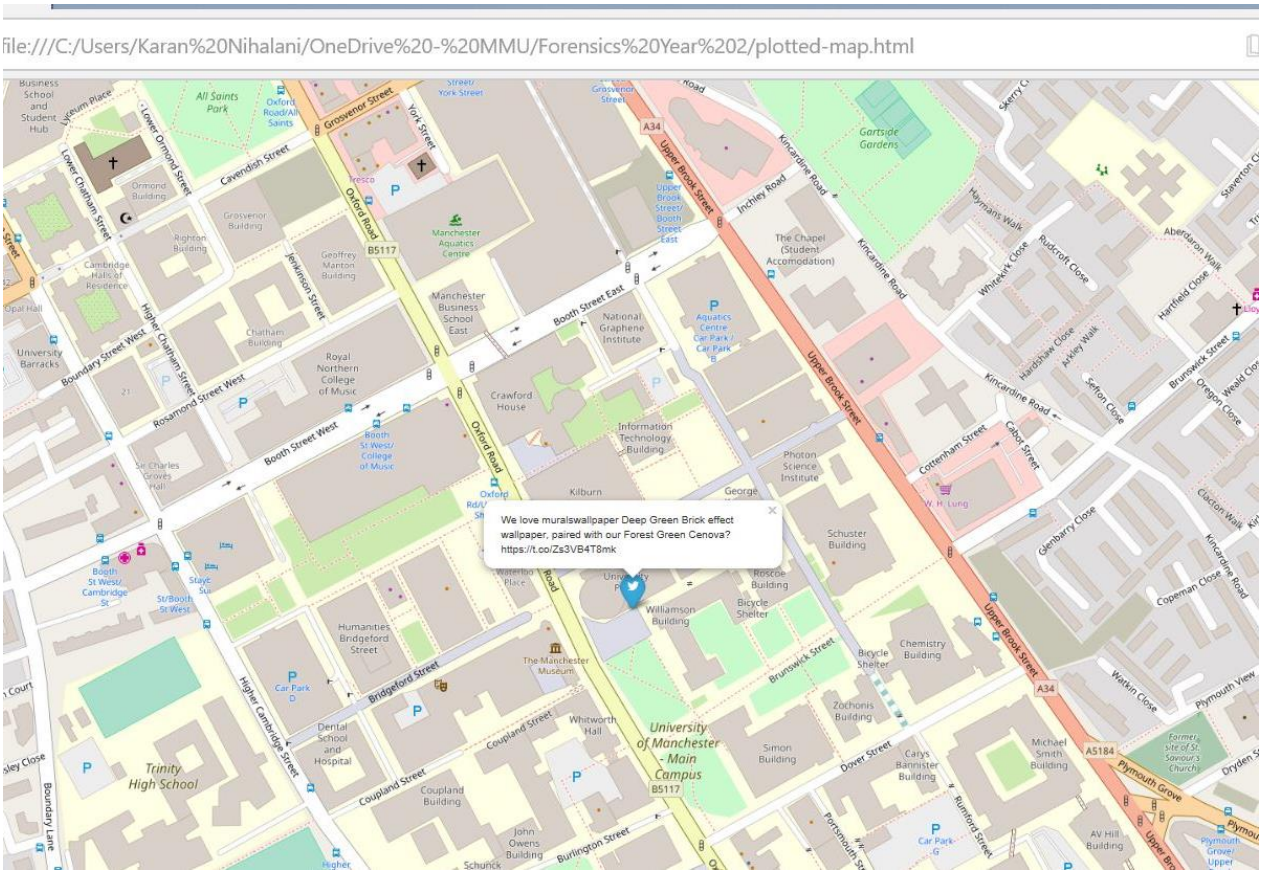

Fig. 5

Below the *"print"* statements used before, it's necessary to employ, *"map_osm.save('plotted-map.html')"*, as it saves the tweets' location on the map, just before opening a new tab on the user's default browser, putting *"webbrowser.open_new_tab('plotted-map.html')"* to use. This syntax is made accessible due to the *"import webbrowser"* library.

Template Based Searching

The basic principle of the Template Based Search is that it allows the program to detect an array, established by a main local variable that is related in context to that same array and plots the tweets that include those words affiliated to the variable. One way the program's user can benefit from this type of search is because it is a "way to allow inexperienced user to query systems without the need of knowing any specific language or how data is structured" (P. Cappellari, R. D. Virgilio, M. Miscione Universitá Roma Tre, 2009). Going through the code, we can examine that the Template Based function will undergo a search regarding people who drink beer or are linked with beer in any way. The word, *"beer"* is set as the main local variable on line 57 (**Fig. 6**) and various other words related to *"beer"* have been selected as the array the program will go through to detect the tweets that are associated with it.

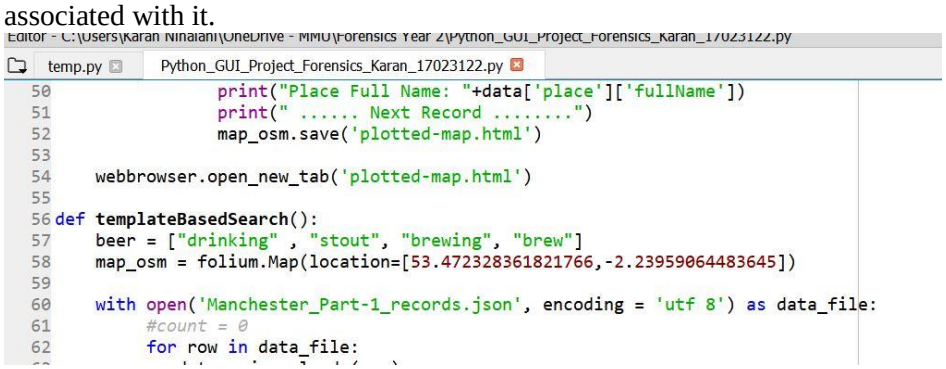

Fig. 6

When it comes down to plotting the Template Based tweet results, the same syntax as the Simple Text Search and Map Plotting is used, except that there is a small modification. As viewed in **Fig. 6.1**, a *"for"* loop is made use of, just before the *"if"* statement. This loops for all the records, in the JSON file, containing the words in the array defined above.

```
with open('Manchester_Part-1_records.json', encoding = 'utf 8') as data_file:
    #count = 0
    for row in data_file:
        data = json.loads(row)

        tempText = data['text']
        latt = data['geoLocation']['latitude']
        long = data['geoLocation']['longitude']

        for word in beer:
            if word in tempText:
                #count = count + 1
                StringToScroll = "\n" + "\nDate: " + data['createdAt']['$date'] + "\nLatitude:" + str(data['geoLo
                folium.Marker([latt, long], popup=tempText, icon=folium.Icon(color='blue',icon='twitter', prefix=
                scr.insert(tk.INSERT,StringToScroll)
                print("Tweet Text: "+data['text'])
                print("Place Name: "+data['place']['name'])
                print("Place Full Name: "+data['place']['fullName'])
                print(" ..... Next Record .....")
                map_osm.save('plotted-map.html')

webbrowser.open_new_tab('plotted-map.html')
```

Fig. 6.1

The images below (**Figs. 7** and **8**), illustrate the Template Based Search carried out for the word "beer", as it correctly plots 2 tweets containing the words within the code's array.

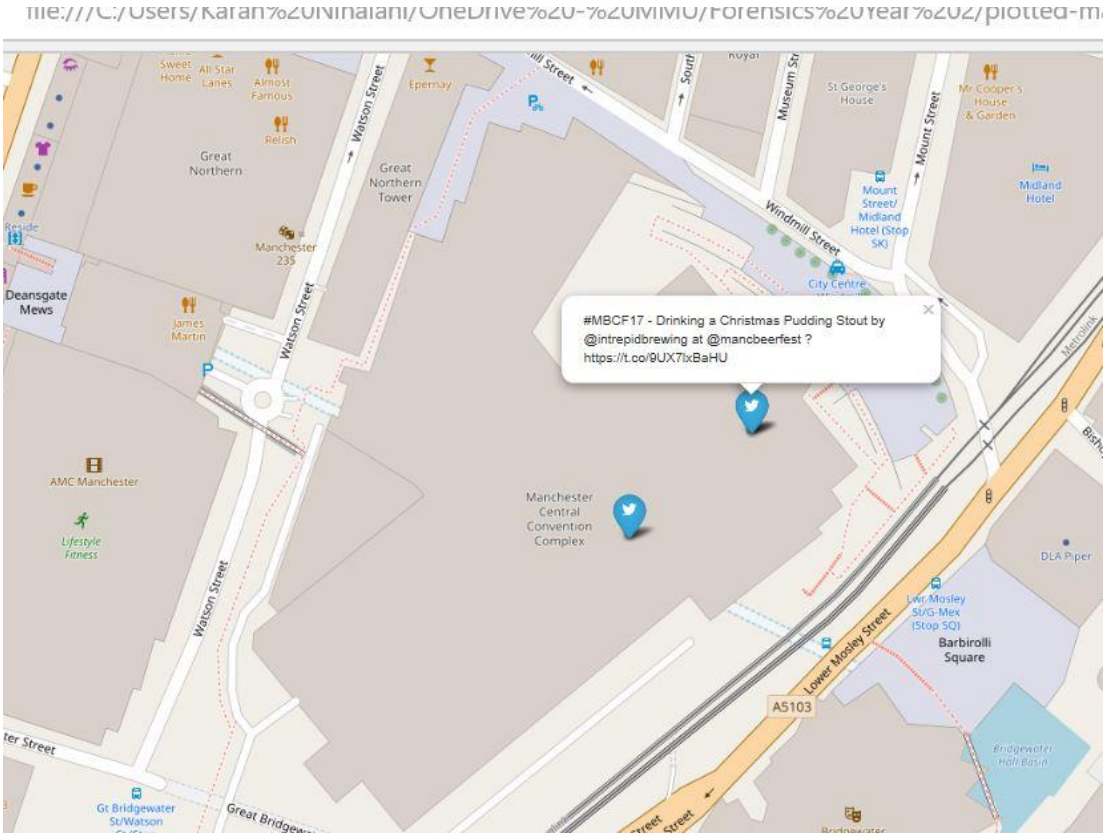

Fig. 7

Fig. 8

Regular Expression Searching

A Regular Expression Search allows the program to identify a certain pattern already mentioned in the code, within text or even a JSON file, in this case. We could say that it helps "match or find other strings or sets of strings" too (Tutorials Point, 2018). It would then proceed to reading all of the data and display it according to the pattern typed previously in the code.

In order to be able to perform the Regular Expression Search, the import module, *"import re"* had to be typed into the beginning of the code (refer back to **Fig.1**). The structure used for the code in this function, is different from the past searches, but not very complex. We can spot in **Fig. 9**, the variation of the syntax to open the JSON file. The letter *"r"* is added at the end of the command, instructing the function to read the entire file and all the records in it.

```
83 def RegularExpression():
84     # Open file
85     f = open('Manchester_Part-1_records.json', 'r')
```

Fig. 9

Moving down the code, the custom Regular Expression algorithm is written (**Fig 9.1**), as this sets the *"findall"* pattern to the amount of digits in each user's twitter ID, therefore removing and compiling all the user ID's at once.

```
86     #set the findall pattern to the amount of digits in each user's twitter ID
87     regex = re.compile ('\d\d\d\d\d\d\d\d\d\d\d\d\d\d\d\d\d\d')
88     #read the data from the file
89     twitter_ID = regex.findall(f.read())
90
```

Fig. 9.1

Moreover, the local variable, *"twitter_ID"*, is set, as it is also implemented within the *"if"* statement preceding it that authorizes all the twitter ID numbers to be displayed appropriately in the console, on the right-hand side of the screen (**Fig. 9.2**).

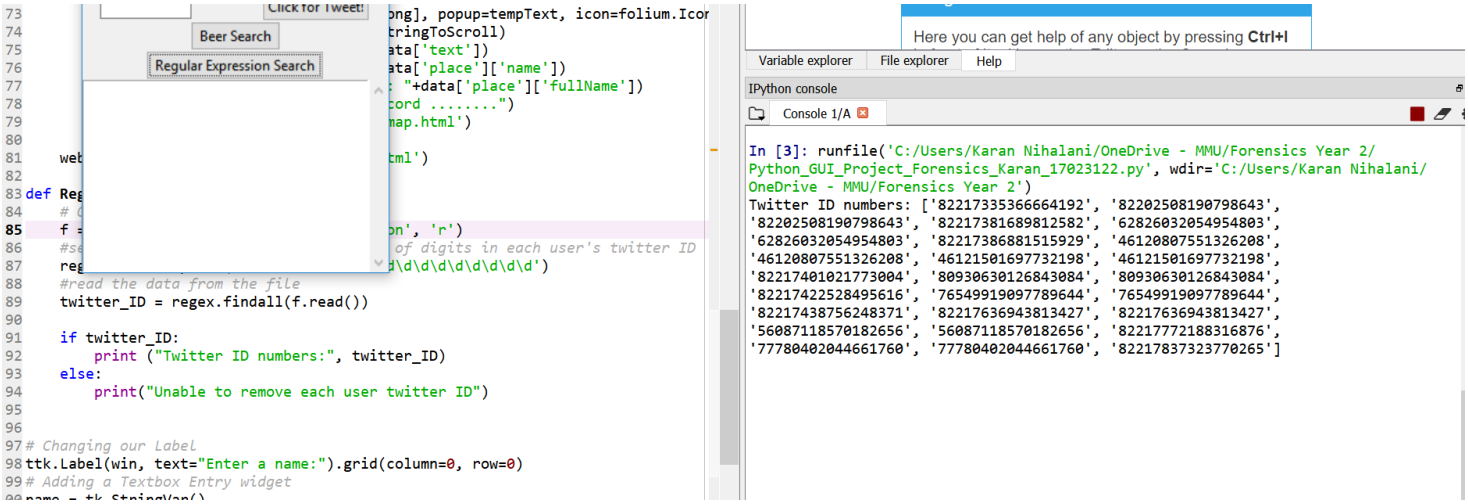

Fig. 9.2

Reflection

To conclude, I had begun this assignment with very little to no experience in Python. According to my belief, I thought that I would struggle while typing code and trying to solve different problems within the multiple lines of code, whereas, due to external research on the programming language, the variety of searches and the careful observation and precautions to try and avoid indentation level errors, this project turned out to be more than accomplishable with the amount of effort put into it. I now feel much more confident with Python, which will result very advantageous in the near future, mentioning my ability to code in this language to potential employers and finally, presenting this map plotting GUI as a proud example.

In addition, this software created could be used in the analysis of a digital forensics investigation, as one of or the most important aspects of this stage in this investigation is the need to trace evidence or search for evidence to possess. In my opinion, of all the three searches, the Template Based Search would deem the most useful. Given the example of trying to prove that there are multiple violent crimes occurring at a specific time and at a certain place, it would be possible to gather all the tweets from those days, times and locations, which contain all the words within the *violent crimes* category. In the documentation aspect of the investigation, this program would be an excellent way of showing results, to be able deduce patterns of various criminal activities, on a specific location desired on the map. This is possible because of the clear representation of the dates, times, twitter ID's and even latitude and longitude demonstrated in the Python GUI itself.

References

The Python Software Foundation. (2018) 24.1 Tkinter – Python interface to Tcl/Tk. [Online] [Accessed on 22nd November 2018]
https://docs.python.org/2/library/tkinter.html

Domino Data Lab. (2015) Creating interactive crime maps with Folium [Online] [Accessed on 4th December 2018]
https://blog.dominodatalab.com/creating-interactive-crime-maps-with-folium/

P. Cappellari, R. D. Virgilio, M. Miscione; Universitá Roma Tre (2009) A Template-Based Approach to Keyword Search over Semantic Data [Online] [Accessed on 5th December 2018]
https://pdfs.semanticscholar.org/6188/bb6b031653c5f6bead8d0aed19a32bdbd678.pdf

Tutorials Point (2018) Python Regular Expressions [Online] [Accessed on 5th December 2018]
https://www.tutorialspoint.com/python/python_reg_expressions.htm

Appendix (code):

```
import tkinter as tk
import json
import folium
from tkinter import ttk
from tkinter import scrolledtext
import webbrowser
import re



# Create instance
win = tk.Tk()
# Add a title
win.title("Python GUI")

#win.minsize(width=850, height=500)
#win.maxsize(width=850, height=500)
# Disable resizing the GUI
win.resizable(0,0)
# Modify adding a Label
aLabel = ttk.Label(win, text="A Label")
aLabel.grid(column=0, row=0)
#Modified Button Click Function
def clickMe():
    map_osm = folium.Map(location=[53.472328361821766,-2.23959064483645])
    action.configure(text='Tweets: ' + name.get())
    #action.configure(text='Hello ' + name.get())
    #print("you have entered ... "+name.get())
    count = 0
    searchkeyword = name.get()
    print("you have entered ... "+searchkeyword)
```

```
    with open('Manchester_Part-1_records.json', encoding = 'utf 8') as data_file:


        for row in data_file:
            data = json.loads(row)
            tempText = data['text']
            createdAt= data['createdAt']['$date']
            print("you have entered ... "+tempText)


            if searchkeyword in tempText:
                StringToScroll = "\n" + "\nDate: " + data['createdAt']['$date'] + "\nLatitude:" +
str(data['geoLocation']['latitude']) + "\nLongitude:" + str(data['geoLocation']['longitude']) + "\nTweet
Text:" + data['text']
                count = count + 1
                scr.insert(tk.INSERT,StringToScroll)
                latt = data['geoLocation']['latitude']
                long = data['geoLocation']['longitude']
                folium.Marker([latt, long], popup=tempText, icon=folium.Icon(color='blue',icon='twitter',
prefix='fa')).add_to(map_osm)
                #print("Geo-Location "+str(data['geoLocation']['latitude']))
                print("Tweet Text: "+data['text'])
                print("Place Name: "+data['place']['name'])
                print("Place Full Name: "+data['place']['fullName'])
                print(" ..... Next Record .....")
                map_osm.save('plotted-map.html')


    webbrowser.open_new_tab('plotted-map.html')


def templateBasedSearch():
    beer = ["drinking" , "stout", "brewing", "brew"]
    map_osm = folium.Map(location=[53.472328361821766,-2.23959064483645])


    with open('Manchester_Part-1_records.json', encoding = 'utf 8') as data_file:
        #count = 0
```

```
        for row in data_file:
            data = json.loads(row)

            tempText = data['text']
            latt = data['geoLocation']['latitude']
            long = data['geoLocation']['longitude']

            for word in beer:
                if word in tempText:
                    #count = count + 1
                    StringToScroll = "\n" + "\nDate: " + data['createdAt']['$date'] + "\nLatitude:" + str(data['geoLocation']['latitude']) + "\nLongitude:" + str(data['geoLocation']['longitude']) + "\nTweet Text:" + data['text']
                    folium.Marker([latt, long], popup=tempText, icon=folium.Icon(color='blue',icon='twitter', prefix='fa')).add_to(map_osm)
                    scr.insert(tk.INSERT,StringToScroll)
                    print("Tweet Text: "+data['text'])
                    print("Place Name: "+data['place']['name'])
                    print("Place Full Name: "+data['place']['fullName'])
                    print(" ..... Next Record .....")
                    map_osm.save('plotted-map.html')

    webbrowser.open_new_tab('plotted-map.html')

def RegularExpression():
    # Open file
    f = open('Manchester_Part-1_records.json', 'r')
    #set the findall pattern to the amount of digits in each user's twitter ID
    regex = re.compile ('\d\d\d\d\d\d\d\d\d\d\d\d\d\d\d\d\d\d')
    #read the data from the file
    twitter_ID = regex.findall(f.read())

    if twitter_ID:
```

```
        print ("Twitter ID numbers:", twitter_ID)
    else:
        print("Unable to remove each user twitter ID")


# Changing our Label
ttk.Label(win, text="Enter a name:").grid(column=0, row=0)
# Adding a Textbox Entry widget
name = tk.StringVar()
nameEntered = ttk.Entry(win, width=12, textvariable=name)
nameEntered.grid(column=0, row=1)
# Adding a Button
action = ttk.Button(win, text="Click for Tweet!", command=clickMe)
action.grid(column=2, row=1)

beerSearch = ttk.Button(win, text=" Beer Search ", command=templateBasedSearch)
beerSearch.grid(column=0, columnspan=3)



regularExpression = ttk.Button(win, text=" Regular Expression Search ",
command=RegularExpression)
regularExpression.grid(column=0, columnspan=4)

#action.configure(state='disabled') # Disable the Button Widget

# Using a scrolled Text control
scrolW = 30
scrolH = 10
scr = scrolledtext.ScrolledText(win, width=scrolW, height=scrolH, wrap=tk.WORD)
scr.grid(column=0, columnspan=3)

# Place cursor into name Entry
nameEntered.focus()
```

```
#======================
# Start GUI
#======================
win.mainloop()
```