F20SC

INDUSTRIAL PROGRAMMING

December 10, 2017

Jason Shawn D' Souza

Heriot Watt ID: H00202543

Heriot Watt University

Industrial Programming

Contents

INTRODUCTION

The purpose of this report is to document the work done in developing a Data Analyser using Python 3.6. The dataset used in this analyser was provided by issuu. All code was written in python using the PyCharm IDE which made life easier(fixing identation errors especially).

Assumptions

- The user computing environment supports application running in **Python 3.6**
- Users system must have Python 3.6 installed
- User must know what the issuu dataset contains
- User must know what each task does by reading the coursework specification (GUI provides hints as to what each task does)
- Application was developed on Windows 10 and should be able to run on Windows systems 7 onwards
- The external libraries used by the application
  - Pandas
  - Matplot
  - Graphviz
  - Tkinter

Document Overview

The document consists of the following:

- Requirements checklist(checklist of what was completed and what was not).

- Design choices taken into consideration for :
    1. Coding style and code design
    2. The Graphical User Interface(GUI)
    3. The Command Line User Interface(CLI)
- User guide
- Developer guide
- Testing
- Conclusion
- References to useful resources that helped in developing the application

REQUIREMENTS CHECKLIST

Given below is a list of requirements for that were given in the coursework specifications. Whether a Task has been implemented or not has been clearly mentioned, with all tasks being preceded by their number

1. Python : The core functionality has been coded in python
2. **Viewing histogram by**
   (a) **Country** - Both CLI and GUI implemented
   (b) **Continent** - Both CLI and GUI implemented
3. **Viewing histogram by**
   (a) **Verbose Browser** - Both CLI and GUI implemented
   (b) **Popular Browser** - Both CLI and GUI implemented
4. **Also likes functionality**
   (a) **Taking Document id and returning all visitor ids that read it** - Functionality implemented
   (b) **Taking visitor id and returning all document ids read** - Functionality implemented
   (c) **Alsolikes taking document id(visitorid optional)** - Functionality implemented without sorting(Joint readers working only for visitorid=None)
   (d) **Alsolikes taking top 10 sorted by number of readers** - **NOT IMPLEMENTED**
5. **Also likes graph** - Functionality implemented(Partially correct,draws graphs with right edges but some cases only)

DESIGN CONSIDERATIONS AND CODE DESIGN

File structure

The files are designed keeping in mind the functions within them

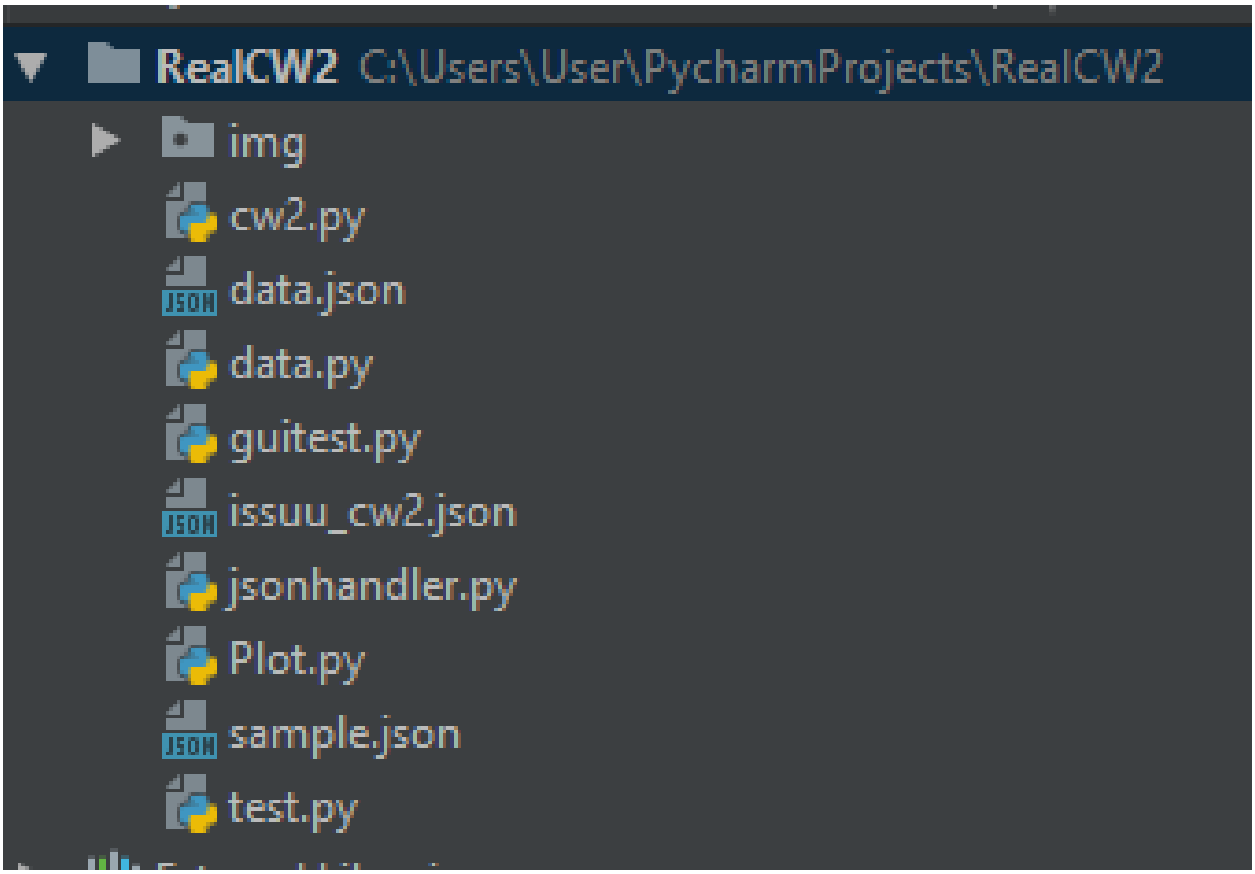

Figure 1: All files required for the application

As the Figure 1 shows the above files with various methods within them

- The cw2.py is the main part of the app and consists
    1. An if condition to run gui
    2. Else run the analyser through the commandline
- Plot.py contains a plothistogram method that is generic enough to plot the data received from the data class.

- jsonhandler.py contains the jsonreader method that reads the json file line by line and using pandas, converting it to a dataframe object to be used in the data,gui and cw2/main modules.
- data.py contains all the methods required for running the different tasks
- test.py is a dummy file that was used during development(it should be empty at the time of submission)
- The 'img' subdirectory is where the graphviz directed graph is stored as a .png

Code Design

The python code written in this coursework is more modular and scalar than the code written in C sharp(Coursework 1). Modular files and structure make it easy for people (especially software developers),recognise what the code does which makes it easier for developers to further enhance the code.

Class

There is only one class and it is in guitest.py. The class name is gui and it contains all the variables, tkinter component initialision and then placing them on the tkinter frame as can be see in **??** under the section "Developer guide"

GUI Design

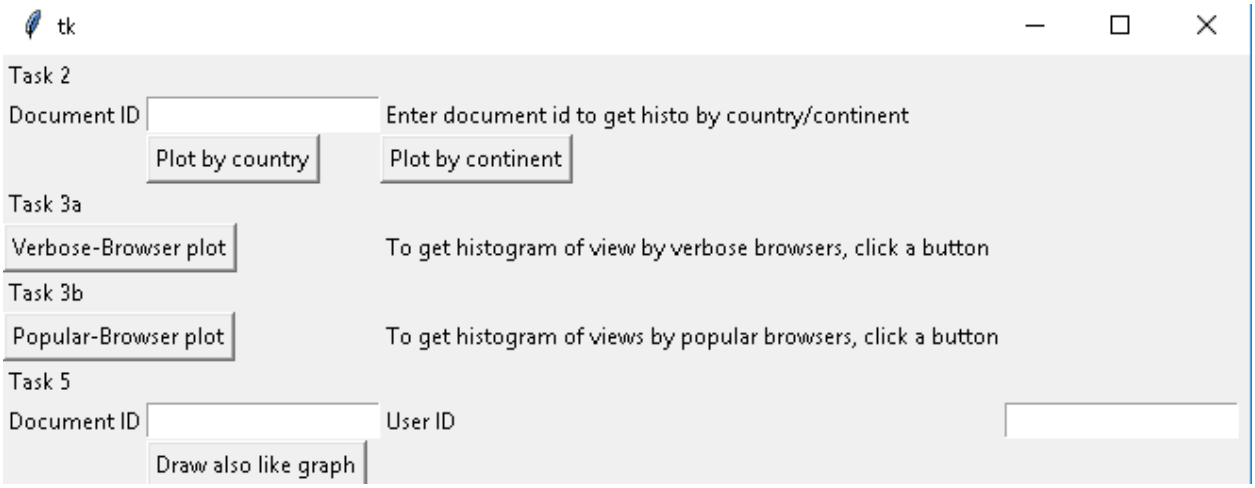

Figure 2: The GUI

Figure 2 shows the GUI. The approach was to go for a minimal UI that was easy to use and understand. Alternatively to having all tasks in one layout another option would be a tabbed layout(Tasks accessible as tabs on the UI, so each task gets its own tab). Reason for implementation of this simple GUI and not a tabbed one is due to time-constraints as some tasks took too long to complete. On clicking the button, a new window with the required histogram appears for tasks 2 and 3 (both a and b). All of them are similar to the one shown in 15

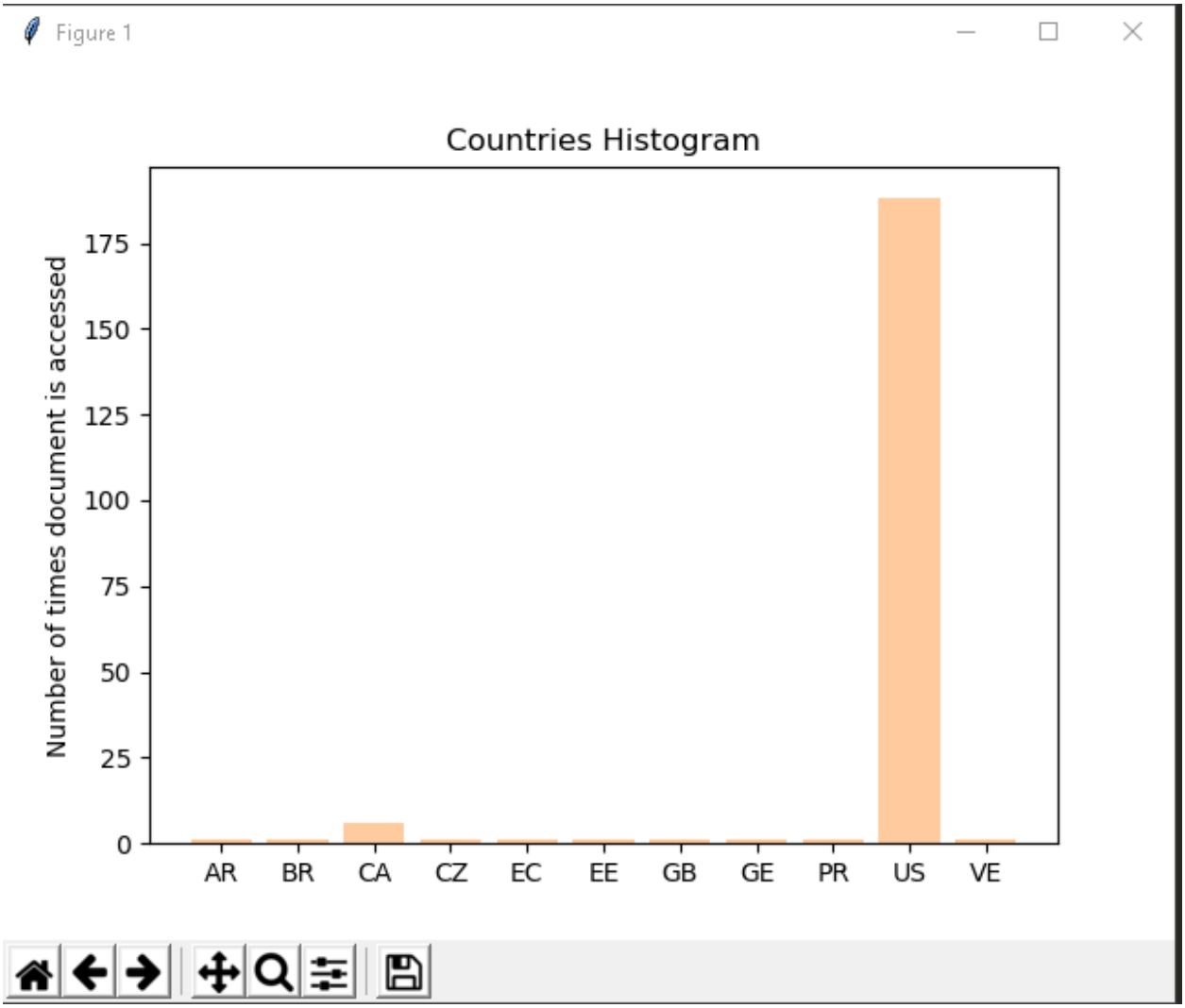

Figure 3: Countries Histrogram

The other histograms are similar and to change the layout(Figure 15 has vertical layout currently) he can change the layout in the 'data.py' class and the suitable task method.

Command Line Design

For the coursework we were asked to build a command line interface where tasks are accessible and the outputs are printed on the line itself and the histograms(if required by the task) open in a new window. There are 4 arguments that this application can take as mentioned in the coursework specifications

- **task_id** This argument is **compulsory** to be able to identify what tasks does a user want to run. The user can enter the name of the task to run it. Example: if '2a' is entered, then task 2a runs, generates a histograms and shows it outside the command-line(opens up the plot in a new window).

- **-u, –user_uuid**

  This argument is **optional** and used when a particular task requires it. Example: 3a requires no user id(or doc id) so only the taskid is required.

- **-d, –doc_id**

  This argument is also **optional** required when needed. **But it is compulsory to use it for tasks it is required ,i.e, 2a,2b,4d and 5**

- **-f, –file_name**

  This argument is **designed to be optional**. This means that if no file is mentioned, a default is used but a user can still input his/her choice of files **as long as they are in the path of all files of the project.**

USER GUIDE

The user has two ways to run the analyser application:

1. The Graphical User Interface (GUI)
2. The Command Line Interface (CLI)

It is necessary that python 3.6 is installed on the users system

Graphical User Interface (GUI) Guide

To run the GUI, the user must go to the directory with all the module files(root directory,i.e,the folder Real CW2 for me) and open the Command Prompt(Windows, shift right click or From any command prompt manually navigate to the file using relevant cd commands). On the command prompt write "py cw2.py" as shown in Figure 4

Figure 4: Using the command prompt to run the GUI

This opens the gui as shown in Figure 2

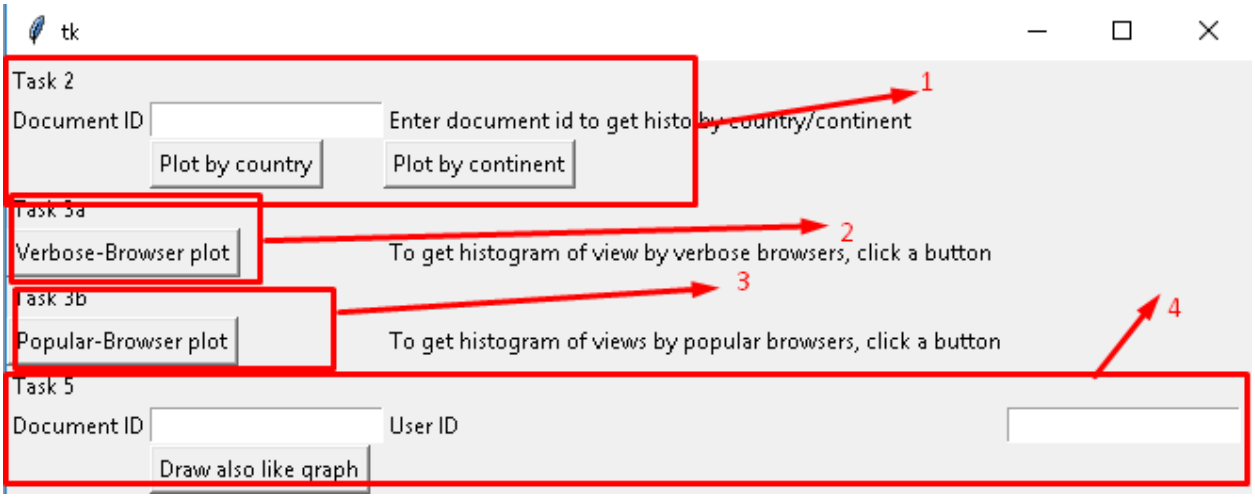

Figure 5: Breakdown of the GUI

Figure 5 shows the gui is broken down into 3 parts (NOTE:- By default the json file used is data.json, this can only be manually changed by altering the code or by using the command line interface)

1. **Task2**

   Both task 2a and 2b come under this part. To get the result of 2a(by country) the user must press the "Plot by country" option **but ONLY after entering a valid document id.**To get the result of 2b(by continent, "Plot by continent" button must be pressed(valid document id must be entered). This opens an histogram with the number of views of the document on y-axis and the countries/continents(on x-axis). This orientation is by default but to invert the axis he can change the orientation from vertical to horizontal in data.py 's relevant methods (bycountry or bycontinent) as shown in 6

```
Plot.plothistogram('Countries Histogram', xdata, ydata, 'vertical')
```

Figure 6: Changing orientation

Figure 7 shows how the histogram looks. The viewer has further built in controls such as zooming in and out and returning to previous views

(more useful information -http://matplotlib.org/users/navigation_toolbar.html)

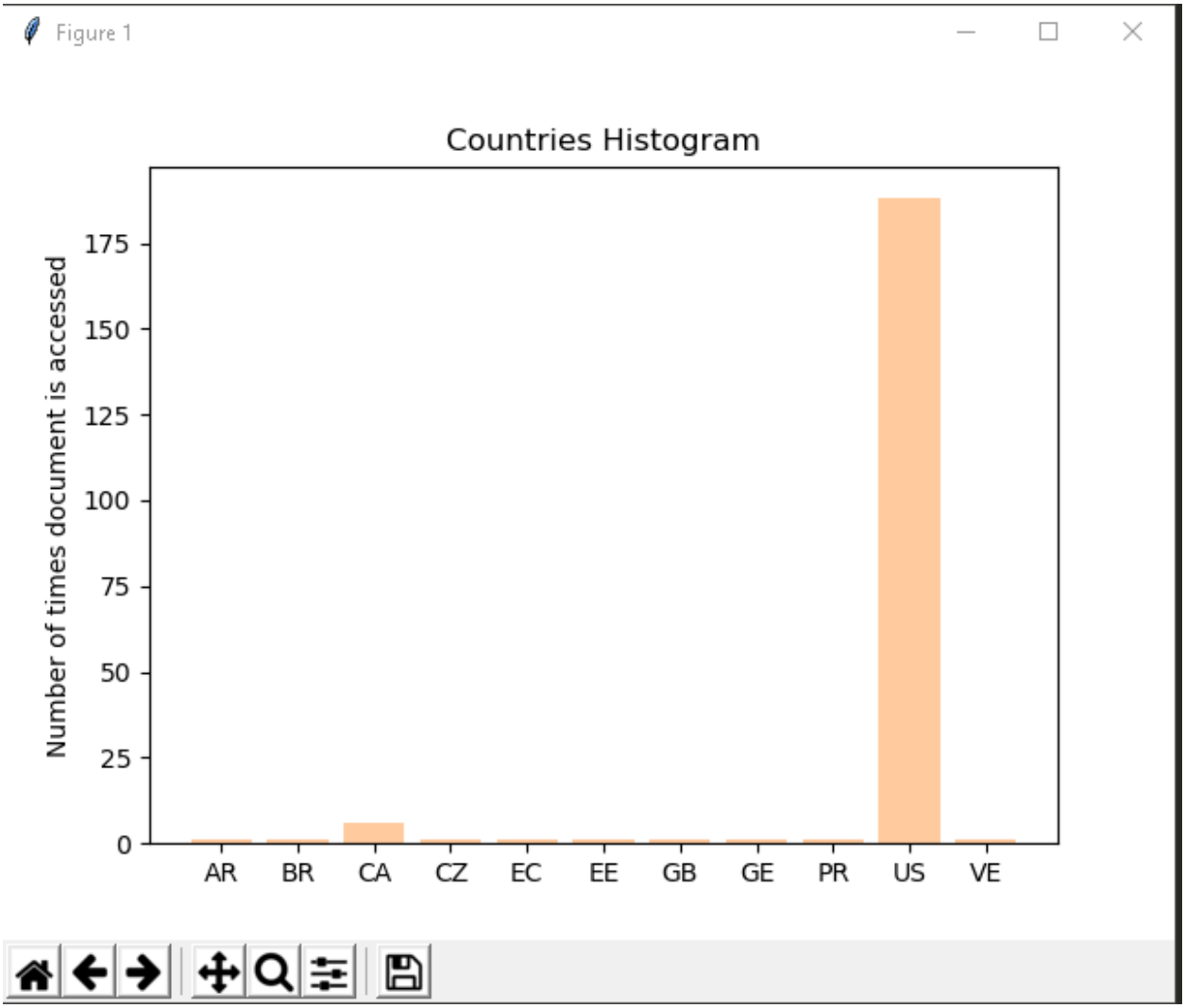

Figure 7: Countries Histrogram

2. **Task3a**

   This part requires no input. The button click results in an histogram being produced in a new window

3. **Task3b**

   This part also needs no input. Button click opens a new window with the matplotlib viewer containing the histogram by popular browsers.

4. **Task5**

   This task takes two inputs one being optional

Command Line Interface Guide

To run the GUI, the user must go to the directory with all the files/modules(root directory,i.e for me within the folder Real CW2) and open the Command Prompt(Windows, shift right click or From any command prompt manually navigate to the file using relevant cd commands). On the command prompt write "py cw2.py -h"(or "py cw2.py (relevant commands if the user knows what the commands are)" as shown in Figure 8

```
C:\Users\User\PycharmProjects\RealCW2>py cw2.py -h
usage: cw2.py [-h] [-u USER_UUID] [-f FILE_NAME] [-d DOC_UUID] [-t TASK_ID]

F20SC CW2 Data Analysis with issuu dataset

optional arguments:
  -h, --help            show this help message and exit
  -u USER_UUID, --user_uuid USER_UUID
                        Enter the uuid of a user.
  -f FILE_NAME, --file_name FILE_NAME
                        Enter the file name of the json.
  -d DOC_UUID, --doc_uuid DOC_UUID
                        Enter the document_id of the document. used for Task
                        2a, 2b and 4d
  -t TASK_ID, --task_id TASK_ID
                        Enter a tasks id to run these can be one of 2a 2b 3a
                        3b 4d 5

C:\Users\User\PycharmProjects\RealCW2>
```

Figure 8: The command line interface

Example

Running task 5 as follows with input documentid = '140227080132-c038e5546d578cf4895a66e6fd8d2dc0' and **NO** visitor/ user id

```
C:\Users\User\PycharmProjects\RealCW2>py cw2.py -t 5 -d 140227080132-c038e5546d578cf4895a66e6fd8d2dc0
<pandas.core.groupby.DataFrameGroupBy object at 0x0000021C52C51FD0>
43014347873d3062    1
cf5a4af49ff15c7b    1
Name: col, dtype: int64
<pandas.core.groupby.DataFrameGroupBy object at 0x0000021C52BEA9B0>
140227080132-c038e5546d578cf4895a66e6fd8d2dc0    1
Name: col, dtype: int64
<pandas.core.groupby.DataFrameGroupBy object at 0x0000021C3C9F1748>
140227080132-c038e5546d578cf4895a66e6fd8d2dc0    1
Name: col, dtype: int64
digraph {
        "43014347873d3062"
        "140227080132-c038e5546d578cf4895a66e6fd8d2dc0"
        "43014347873d3062" -> "140227080132-c038e5546d578cf4895a66e6fd8d2dc0"
        cf5a4af49ff15c7b
        "140227080132-c038e5546d578cf4895a66e6fd8d2dc0"
        cf5a4af49ff15c7b -> "140227080132-c038e5546d578cf4895a66e6fd8d2dc0"
}
img/g1.pnghas been created
```

Figure 9: Graphviz digraph

Figure 9 shows the commands to run task 5 on the cli and Figure 10 shows the diagram obtained

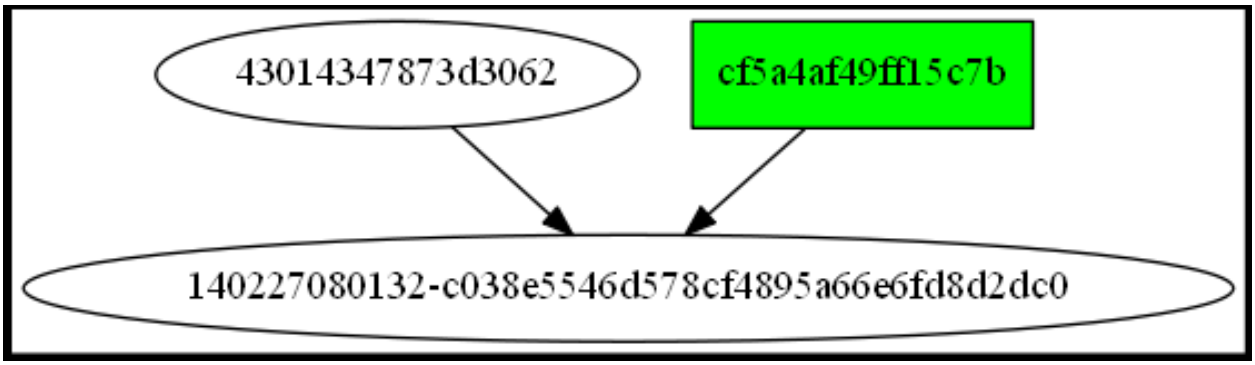

Figure 10: Graphviz digraph

The structure of the arguments to be passed is this:

"py cw2.py -u user_uuidhere -d doc_idhere -t task_id -f jsonfile"

The following example shows the use of a mandatory argument: If the user wants to run Task2a. they can type the command like this:

"py cw2.py 2a -d 140227080132-c038e5546d578cf4895a66e6fd8d2dc0"

This will execute 2a and show the histogram. For task 2a it is **necessary** to have a docid so entering no doc id for Task2a will give the result as seen in Figure 11

```
C:\Users\User\PycharmProjects\RealCW2>py cw2.py -t 2a
Enter valid document id
```

Figure 11: Running task 5 on cli

Lastly, point to note here is that the 'help' option has been directly embedded in the CLI so as to point the user in the right direction (what arguments to use). It can be accessed as seen in Figure 12 ,i.e,

"py cw2.py -h"

```
C:\Users\User\PycharmProjects\RealCW2>py cw2.py -h
usage: cw2.py [-h] [-u USER_UUID] [-f FILE_NAME] [-d DOC_UUID] [-t TASK_ID]

F20SC CW2 Data Analysis with issuu dataset

optional arguments:
  -h, --help            show this help message and exit
  -u USER_UUID, --user_uuid USER_UUID
                        Enter the uuid of a user.
  -f FILE_NAME, --file_name FILE_NAME
                        Enter the file name of the json.
  -d DOC_UUID, --doc_uuid DOC_UUID
                        Enter the document_id of the document. used for Task
                        2a, 2b and 4d
  -t TASK_ID, --task_id TASK_ID
                        Enter a tasks id to run these can be one of 2a 2b 3a
                        3b 4d 5
```

Figure 12: Cli Help

DEVELOPER GUIDE

As a prelude, any developer that doesnt have any information about pandas or python in general, all code is commented and explains everything

Libraries

Here is a list of ALL libraries used in the application (as you will notice their use when they are used in methods/classes

- os.path(getting json file from the system)
- pandas (used in many files to convert to a dataframe so as to manage and access the data efficiently)
- tkinter (used in gui class in the guitest.py to make the GUI)
- argparse (used in cw2.py to make the command line interface of the application)
- matplotlib (used in plot.py to make histograms using data(in lists) from data.py)
- from collections import Counter (in data.py to count joint readers)
- from graphviz import Digraph(in data.py to make a directed graph for task5)
- json(Used by the jsonhandler.py and is used to read the json file and then finally converted to a pandas dataframe object)
- sys (so as to get the amount of args for cli (in cw2.py))

AND ALSO THE FILES THEMSELVES

- Plot.py (in data.py)
- data.py (in guitest.py ,cw2.py)
- guitest.py (in cw2.py)

Files/Modules

cw2.py

As mentioned earlier cw2.py is the gateway to everything in the application/data analyser.

```
import json
import pandas as pd
import os.path
import sys
import json
import pandas as pd
import os.path

from tkinter import *

import jsonhandler
import guitest
import data
import argparse
#setting up jsonfile
dataframe = jsonhandler.jsonreader("data.json")
```

(a) All imports in the cw2.py and shows call made to jsonhandler.py

```
if (len(sys.argv) <= 1):
    #Use gui if 1 or more args on python promt/ MEANING: py cw2.py opens gui ------  py cw2.py blah blah opens CLI
    #Cant do <1 cause 3a 3b dont take additional args so py cw2.py 3a open gui instead of cli
    gui = guitest.gui(dataframe)
else:
    #use cli
    command = argparse.ArgumentParser(description='F20SC CW2 Data Analysis with issuu dataset')
    command.add_argument('-u', '--user_uuid',
                         help='Enter the uuid of a user.')
    command.add_argument('-f', '--file_name',
                         help='Enter the file name of the json.')
    command.add_argument('-d', '--doc_uuid',
                         help='Enter the document_id of the document.\n used for Task 2a, 2b and 4d')
    command.add_argument('-t', '--task_id',
                         help='Enter a tasks id to run these can be one of \n 2a 2b 3a 3b 4d 5')
    # managing incoming arguments and sending them to their related functions
```

(b) Calling either gui or cli and relevant arguments

Figure 13: cw2.py module

```
command_args = command.parse_args()
if command_args.task_id == '2a':
    if command_args.doc_uuid is not None:
        if command_args.file_name == None:
            dataframe = jsonhandler.jsonreader("data.json")
            data.bycountry(dataframe, command_args.doc_uuid)
        else:
            dataframe = jsonhandler.jsonreader(command_args.file_name)
            data.bycountry(dataframe, command_args.doc_uuid)
    else:
        print('Enter valid document id with -d docidhere')
elif command_args.task_id == '2b':
    if command_args.doc_uuid is not None:
        if command_args.file_name == None:
            dataframe = jsonhandler.jsonreader("data.json")
            data.bycontinent(dataframe, command_args.doc_uuid)
        else:
            dataframe = jsonhandler.jsonreader(command_args.file_name)
            data.bycontinent(dataframe, command_args.doc_uuid)
    else:
        print('Enter valid document id with -d docidhere')
elif command_args.task_id == '3a':
    if command_args.file_name == None:
        dataframe = jsonhandler.jsonreader("data.json")
        data.verbosehisto(dataframe)
    else:
        dataframe = jsonhandler.jsonreader(command_args.file_name)
        data.verbosehisto(dataframe)
elif command_args.task_id == '3b':
    if command_args.file_name == None:
        dataframe = jsonhandler.jsonreader("data.json")
        data.properhisto(dataframe)
    else:
        dataframe = jsonhandler.jsonreader(command_args.file_name)
```

(c) CLI for each task

```
elif command_args.task_id == '3b':
    if command_args.file_name == None:
        dataframe = jsonhandler.jsonreader("data.json")
        data.properhisto(dataframe)
    else:
        dataframe = jsonhandler.jsonreader(command_args.file_name)
        data.verbosehisto(dataframe)

        #elif command_args.task_id == 'Task4d':
    #data.also_liketop10(dataframe)
elif command_args.task_id == '5':
    if command_args.user_uuid is not None and command_args.doc_uuid is not None:
        if command_args.file_name == None:
            dataframe = jsonhandler.jsonreader("data.json")
            data.task5(dataframe, command_args.doc_uuid, command_args.user_uuid)
        else:
            dataframe = jsonhandler.jsonreader(command_args.file_name)
            data.task5(dataframe, command_args.doc_uuid, command_args.user_uuid)
    elif command_args.user_uuid is None and command_args.doc_uuid is not None:
        if command_args.file_name == None:
            dataframe = jsonhandler.jsonreader("data.json")
            data.task5(dataframe, command_args.doc_uuid)
        else:
            dataframe = jsonhandler.jsonreader(command_args.file_name)
            data.task5(dataframe, command_args.doc_uuid)
    else:
        print('Enter valid document id with -d docidhere. Document id is mandatory whereas user id is optional')


else:
    print('Invalid argument. Use  -h to see help')
```

(d) CLI for each task continued

Figure 13: cw2.py module

As can be seen from Figure 16(a), the key import taken for cli is argparse. Any developer using this import can add arguments (enhance my cli implementation) by adding possibly more arguments. The line of code that parses these arguments can be seen in the first line of Figure 16(c)

"command_args = command.parse_args()"

Next we move on to the jsonhandler.py

jsonhandler.py

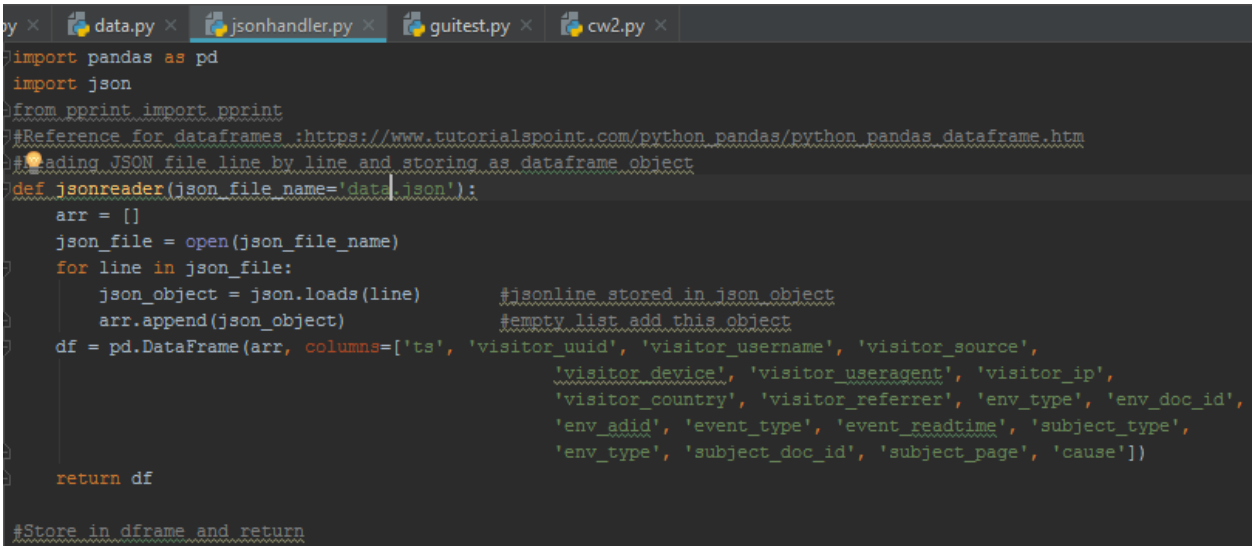

Figure 14: The jsonhandler class with its methods

The jsonreader method taks the json file name as input as can be seen from the 14. This json file is loaded line by line into an empty list and then converted to a Pandas Dataframe object with column names **exactly** as seen from the json files provided.

data.py

This is the task where all the main analysis takes place. The code being too big to screen shot will be listed here

```
1 import Plot
2 import pandas as pd
```

```
from collections import Counter
from graphviz import Digraph
from pprint import pprint
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/' #path to
    dot.exe for running graph
#PLEASE EDIT THE ABOVE PATH FOR GRAPHVIZ TO SHOW :D

#task2a
#Takes documentid and finds countries of viewers of a document
#Check for doc id
def bycountry(data, document_id):
    mid = data['subject_doc_id'] == document_id
    data = data[mid]
    grouped = data.groupby('visitor_country')
    #Group by country
    pprint(grouped)
    #print pandas object
    xdata =[]
    ydata = []
    for key ,item in grouped:
        xdata.append(key)       #append country
        ydata.append(len(item)) # append count
    Plot.plothistogram('Countries Histogram', xdata,ydata,'vertical')

#task2b
# Takes documentid and finds viewers of the document but by continent
def bycontinent(data, document_id):

    # First take data from the previous task(bycountry)

    mid = data['subject_doc_id'] == document_id
    data = data[mid]
```

```
35      grouped = data.groupby('visitor_country')
36      pprint(grouped)
37      xdata =[]
38      ydata = []
39      for key,item in grouped:
40          xdata.append(key)
41          ydata.append(len(item))
42      pprint(xdata)
43      pprint(ydata)
44      #change countries to continents
45      xdata2 = []
46      ydata2 = []
47      index = 0   #python indexes start at 0 SOURCE : -  accessing index in python for
        loops will be referenced
48      for x in xdata:
49          #make sure country exists in the countrytocontinentdict,
50          #countrytocontinent dictionary is taken from the HW MACS site
51          if xdata[index] in countrytocontinent:  #from country count compare to dict.
52              xdata2.append(countrytocontinent[x])
53              ydata2.append(ydata[index])
54          index = index +1
55      for index1 in range(0, len(xdata2)):
56          for index2 in range(index1, len(xdata2)):
57              if not index1 == index2:
58                  if xdata2[index1] == xdata2[index2]:
59                      ydata2[index1] = ydata2[index1] + ydata2[index2]
60                      xdata2[index2]= 'remove'              # The countries get removed
        along with count temp. / they still will exist on graph unless a new list is made
61      #making a final new list that will be used for plotting
62      xdatafinal =[]
63      ydatafinal = []
64      for index3 in range(0, len(xdata2)):
65          if xdata2[index3] == 'remove':      #finally will be removed
```

```
66              continue
67          xdatafinal.append(xdata2[index3])  #final lists
68          ydatafinal.append(ydata2[index3])
69      Plot.plothistogram('Histogram of continents', xdatafinal, ydatafinal,'vertical')
70
71
72
73  #Dictionary of countries to continents taken from
74  #https://www.macs.hw.ac.uk/~hwloidl/Courses/F21SC/Samples/simple_histo.py
75  countrytocontinent= {
76    'AF' : 'AS',
77    'AX' : 'EU',
78    'AL' : 'EU',
79    'DZ' : 'AF',
80    'AS' : 'OC',
81    'AD' : 'EU',
82    'AO' : 'AF',
83    'AI' : 'NA',
84    'AQ' : 'AN',
85    'AG' : 'NA',
86    'AR' : 'SA',
87    'AM' : 'AS',
88    'AW' : 'NA',
89    'AU' : 'OC',
90    'AT' : 'EU',
91    'AZ' : 'AS',
92    'BS' : 'NA',
93    'BH' : 'AS',
94    'BD' : 'AS',
95    'BB' : 'NA',
96    'BY' : 'EU',
97    'BE' : 'EU',
98    'BZ' : 'NA',
```

```
99   'BJ' : 'AF',
100  'BM' : 'NA',
101  'BT' : 'AS',
102  'BO' : 'SA',
103  'BQ' : 'NA',
104  'BA' : 'EU',
105  'BW' : 'AF',
106  'BV' : 'AN',
107  'BR' : 'SA',
108  'IO' : 'AS',
109  'VG' : 'NA',
110  'BN' : 'AS',
111  'BG' : 'EU',
112  'BF' : 'AF',
113  'BI' : 'AF',
114  'KH' : 'AS',
115  'CM' : 'AF',
116  'CA' : 'NA',
117  'CV' : 'AF',
118  'KY' : 'NA',
119  'CF' : 'AF',
120  'TD' : 'AF',
121  'CL' : 'SA',
122  'CN' : 'AS',
123  'CX' : 'AS',
124  'CC' : 'AS',
125  'CO' : 'SA',
126  'KM' : 'AF',
127  'CD' : 'AF',
128  'CG' : 'AF',
129  'CK' : 'OC',
130  'CR' : 'NA',
131  'CI' : 'AF',
```

```
132   'HR' : 'EU',
133   'CU' : 'NA',
134   'CW' : 'NA',
135   'CY' : 'AS',
136   'CZ' : 'EU',
137   'DK' : 'EU',
138   'DJ' : 'AF',
139   'DM' : 'NA',
140   'DO' : 'NA',
141   'EC' : 'SA',
142   'EG' : 'AF',
143   'SV' : 'NA',
144   'GQ' : 'AF',
145   'ER' : 'AF',
146   'EE' : 'EU',
147   'ET' : 'AF',
148   'FO' : 'EU',
149   'FK' : 'SA',
150   'FJ' : 'OC',
151   'FI' : 'EU',
152   'FR' : 'EU',
153   'GF' : 'SA',
154   'PF' : 'OC',
155   'TF' : 'AN',
156   'GA' : 'AF',
157   'GM' : 'AF',
158   'GE' : 'AS',
159   'DE' : 'EU',
160   'GH' : 'AF',
161   'GI' : 'EU',
162   'GR' : 'EU',
163   'GL' : 'NA',
164   'GD' : 'NA',
```

```
165   'GP' : 'NA',
166   'GU' : 'OC',
167   'GT' : 'NA',
168   'GG' : 'EU',
169   'GN' : 'AF',
170   'GW' : 'AF',
171   'GY' : 'SA',
172   'HT' : 'NA',
173   'HM' : 'AN',
174   'VA' : 'EU',
175   'HN' : 'NA',
176   'HK' : 'AS',
177   'HU' : 'EU',
178   'IS' : 'EU',
179   'IN' : 'AS',
180   'ID' : 'AS',
181   'IR' : 'AS',
182   'IQ' : 'AS',
183   'IE' : 'EU',
184   'IM' : 'EU',
185   'IL' : 'AS',
186   'IT' : 'EU',
187   'JM' : 'NA',
188   'JP' : 'AS',
189   'JE' : 'EU',
190   'JO' : 'AS',
191   'KZ' : 'AS',
192   'KE' : 'AF',
193   'KI' : 'OC',
194   'KP' : 'AS',
195   'KR' : 'AS',
196   'KW' : 'AS',
197   'KG' : 'AS',
```

```
198   'LA' : 'AS',
199   'LV' : 'EU',
200   'LB' : 'AS',
201   'LS' : 'AF',
202   'LR' : 'AF',
203   'LY' : 'AF',
204   'LI' : 'EU',
205   'LT' : 'EU',
206   'LU' : 'EU',
207   'MO' : 'AS',
208   'MK' : 'EU',
209   'MG' : 'AF',
210   'MW' : 'AF',
211   'MY' : 'AS',
212   'MV' : 'AS',
213   'ML' : 'AF',
214   'MT' : 'EU',
215   'MH' : 'OC',
216   'MQ' : 'NA',
217   'MR' : 'AF',
218   'MU' : 'AF',
219   'YT' : 'AF',
220   'MX' : 'NA',
221   'FM' : 'OC',
222   'MD' : 'EU',
223   'MC' : 'EU',
224   'MN' : 'AS',
225   'ME' : 'EU',
226   'MS' : 'NA',
227   'MA' : 'AF',
228   'MZ' : 'AF',
229   'MM' : 'AS',
230   'NA' : 'AF',
```

```
231   'NR' : 'OC',
232   'NP' : 'AS',
233   'NL' : 'EU',
234   'NC' : 'OC',
235   'NZ' : 'OC',
236   'NI' : 'NA',
237   'NE' : 'AF',
238   'NG' : 'AF',
239   'NU' : 'OC',
240   'NF' : 'OC',
241   'MP' : 'OC',
242   'NO' : 'EU',
243   'OM' : 'AS',
244   'PK' : 'AS',
245   'PW' : 'OC',
246   'PS' : 'AS',
247   'PA' : 'NA',
248   'PG' : 'OC',
249   'PY' : 'SA',
250   'PE' : 'SA',
251   'PH' : 'AS',
252   'PN' : 'OC',
253   'PL' : 'EU',
254   'PT' : 'EU',
255   'PR' : 'NA',
256   'QA' : 'AS',
257   'RE' : 'AF',
258   'RO' : 'EU',
259   'RU' : 'EU',
260   'RW' : 'AF',
261   'BL' : 'NA',
262   'SH' : 'AF',
263   'KN' : 'NA',
```

```
264   'LC' : 'NA',
265   'MF' : 'NA',
266   'PM' : 'NA',
267   'VC' : 'NA',
268   'WS' : 'OC',
269   'SM' : 'EU',
270   'ST' : 'AF',
271   'SA' : 'AS',
272   'SN' : 'AF',
273   'RS' : 'EU',
274   'SC' : 'AF',
275   'SL' : 'AF',
276   'SG' : 'AS',
277   'SX' : 'NA',
278   'SK' : 'EU',
279   'SI' : 'EU',
280   'SB' : 'OC',
281   'SO' : 'AF',
282   'ZA' : 'AF',
283   'GS' : 'AN',
284   'SS' : 'AF',
285   'ES' : 'EU',
286   'LK' : 'AS',
287   'SD' : 'AF',
288   'SR' : 'SA',
289   'SJ' : 'EU',
290   'SZ' : 'AF',
291   'SE' : 'EU',
292   'CH' : 'EU',
293   'SY' : 'AS',
294   'TW' : 'AS',
295   'TJ' : 'AS',
296   'TZ' : 'AF',
```

```
297   'TH' : 'AS',
298   'TL' : 'AS',
299   'TG' : 'AF',
300   'TK' : 'OC',
301   'TO' : 'OC',
302   'TT' : 'NA',
303   'TN' : 'AF',
304   'TR' : 'AS',
305   'TM' : 'AS',
306   'TC' : 'NA',
307   'TV' : 'OC',
308   'UG' : 'AF',
309   'UA' : 'EU',
310   'AE' : 'AS',
311   'GB' : 'EU',
312   'US' : 'NA',
313   'UM' : 'OC',
314   'VI' : 'NA',
315   'UY' : 'SA',
316   'UZ' : 'AS',
317   'VU' : 'OC',
318   'VE' : 'SA',
319   'VN' : 'AS',
320   'WF' : 'OC',
321   'EH' : 'AF',
322   'YE' : 'AS',
323   'ZM' : 'AF',
324   'ZW' : 'AF'
325 }
326
327
328 #bycontinent(dataframe,'140227080132-c038e5546d578cf4895a66e6fd8d2dc0')
329
```

```
330 #Task 3a
331 def verbosehisto(data):
332     #only need data no documentid
333     grouped = data.groupby('visitor_useragent')
334     xdata= []
335     ydata = []
336     for key, item in grouped:
337         xdata.append(key)
338         ydata.append(len(item))  #item.index
339     # note: too many columns mean that user has to zoom in
340     Plot.plothistogram('Verbose useragents', xdata, ydata, 'horizontal')
341     #The lines below are for CLI plotting first then show data
342     pprint('-------verbose useragents--------')
343     pprint(xdata)
344     pprint(ydata)
345     pprint('-------verbose useragents done ---------')
346
347
348 #Task 3b
349 #only need data no documentid
350 #Popular browsers - Chrome Safari Mozilla Opera, surprised no internet explorer
351 def properhisto(data):
352     grouped = data.groupby('visitor_useragent')
353     browser_name = ['Dalvik', 'Mozilla', 'UCWEB', 'Opera', 'LG-E610']  # Should use
    regex to get these !
354     browser_count = [0, 0, 0, 0, 0]
355     for k, group in grouped:
356         for index in range(0, len(browser_name)):
357             if browser_name[index] in k:
358                 browser_count[index] = browser_count[index] + len(group.index)
359     Plot.plothistogram('Popular Browsers', browser_name, browser_count, 'vertical')
360     #For CLI plt first then show
361     pprint('-------popular browsers--------')
```

```
362     pprint(browser_name)
363     pprint(browser_count)
364     pprint('----------popular browsers done---------')
365
366
367 #Task 4a
368 def getvisitors(data, document_id='130601015527-c1e2993d8290975e7ef350f078134390'):
369     readers = []
370     newreaders = []
371
372     data = data.loc[(data['subject_doc_id'] == document_id) & (data['event_type'] == "
        read")] # READERS ONLY
373     grouped = data.groupby('subject_doc_id')
374     pprint(grouped)
375     for k, group in grouped:
376         if k == document_id:
377             # convert group's column to a list
378             readers = data['visitor_uuid'].tolist()
379             for i in readers:
380                 if i not in newreaders:
381                     newreaders.append(i) #remove duplicate
382             break
383     df = pd.DataFrame(newreaders, columns=["col"])
384     print(df["col"].value_counts())
385     # returning list of visitors
386     return (readers, newreaders)
387 #a=getvisitors(dataframe,'130601015527-c1e2993d8290975e7ef350f078134390')
388
389
390 #Task 4b
391 def getdocbyvisitor(data, visitor_uuid ='f69c153f95c96fa7'):
392     docs = []
393     newdocs = []
```

```
394     data = data.loc[(data['visitor_uuid'] == visitor_uuid)& (data['event_type'] == "read
        ")] # readers only
395     grouped = data.groupby('visitor_uuid')
396     pprint(grouped)
397     for k, group in grouped:
398         if k == visitor_uuid:
399             # From a column to list
400             docs = data['subject_doc_id'].tolist()
401             for i in docs:
402                 if i not in newdocs:
403                     newdocs.append(i) # remove duplicates
404             break
405     df = pd.DataFrame(newdocs, columns=["col"]) #list w/o duplicates
406     print(df["col"].value_counts())
407     # returning list of visitors
408     return (docs, newdocs)
409 #a=getdocbyvisitor(dataframe,'1f891eb0b573e42c')
410
411
412
413 #Task 4c
414 def also_like(data,document_id='100806162735-00000000115598650cb8b514246272b5',
        visitor_uuid =None):
415     #Optional uuid
416     if visitor_uuid ==None:
417         data2 =getvisitors(data,document_id)
418         newlist = []
419         for k in data2[0]:
420             data3 = getdocbyvisitor(data,k)       # if no uuid get all visiotrs tehn for
        each get doc and append
421             newlist.append(data3)
422         pprint(newlist)
423         res_list = [x[0] for x in newlist]
```

```
424         #print(type(res_list))
425         #print(res_list)
426         a=Counter(x for sublist in res_list for x in sublist) #works doc has duplicates
    to get joint readers
427         print(a)
428 
429         #pprint(data2)
430 
431 #User id given
432     if visitor_uuid !=None:      # if uid there then get all docs for ALL POSSIBLE
    visitors
433         data2 = getvisitors(data, document_id) # from these visitors get all docs
434         newlist = []
435         secondlist = []
436         for k in data2[0]:
437                 data3 = getdocbyvisitor(data, k)
438                 newlist.append(data3)
439                 if k == visitor_uuid:
440                     test = getdocbyvisitor(data,k)
441                     secondlist.append(test)
442 
443         #print(secondlist)
444         count = [x[0] for x in secondlist]
445         #print(type(count))
446         #print(count)
447         b = Counter(x for sublist in count for x in sublist) # for the particular userid
448         print(b)
449         res_list = [x[0] for x in newlist]
450         test=[]
451         a = Counter(x for sublist in res_list for x in sublist) # For all the docs
    gotten from the uid
452         print(a)
453         list2=[]
```

```
454         for k in res_list:
455             if k in count:           # Now that we get all visiotrs docs and all docs
    show only docs visiotr read in list
456                 list2.append(k)
457         pprint(list2)
458         res_list2 = [i for i in res_list if i in list2]
459         f = Counter(x for sublist in res_list2 for x in sublist)
460         #print(found)
461         print(f)#The final but the joint readers count is wrong need to fix
462 #task 4d
463 #Not done
464 def also_liketop10(data,document_id='130601015527-c1e2993d8290975e7ef350f078134390',
    visitor_uuid ='1f891eb0b573e42c'):
465     dataneeded = also_like(data,document_id,visitor_uuid)
466
467 #Task 5
468 def task5(data,document_id,visitorid=None):
469     print("Doc Id" +document_id)
470     #b = also_liketop10(dataframe, '130601015527-c1e2993d8290975e7ef350f078134390')
471     if visitorid ==None:
472         data2 =getvisitors(data,document_id)
473         newlist = []
474         for k in data2[0]:
475             data3 = getdocbyvisitor(data,k)
476             newlist.append(data3)
477         res_list = [x[0] for x in newlist]
478         listfinal=[]
479         for i in res_list:
480             if i not in listfinal:
481                 listfinal.append(i)
482             break
483         dot = Digraph(format='png')
484         for x in data2[0]: # for all visitors
```

```
485             for y in listfinal[0]: # for docs
486                 dot.node(x) # vis
487                 dot.node(y) # doc
488                 dot.edge(x,y) # edge between them
489                 if y == document_id: # if same doc. id as input green
490                     dot.attr('node', shape='box', style='filled', fillcolor='green')
491                 else:
492                     dot.attr('node',  style='filled', fillcolor='white')
493         print(dot.source)
494         filename = dot.render(filename='img/g1', view=True) # render and view but still
    store it separately
495         print(filename + "has been created")
496     else:
497         data2 = getvisitors(data, document_id)
498         newlist = []
499         secondlist = []
500         for k in data2[0]:
501             data3 = getdocbyvisitor(data, k)
502             newlist.append(data3)
503             if k == visitorid:
504                 test = getdocbyvisitor(data, k)
505                 secondlist.append(test)
506
507         # print(secondlist)
508         count = [x[0] for x in secondlist]#only docs of given uid
509         # print(type(count))
510         # print(count)
511         res_list = [x[0] for x in newlist]#list with all docs from docid
512         test = []
513         list2 = []
514         for k in res_list:
515             if k in count:
516                 list2.append(k)
```

```
517         #pprint(list2)
518         res_list2 = [i for i in res_list if i in list2]#final required list
519         dot = Digraph(format='png')
520     for x in data2[0]:
521         for y in res_list2[0]:
522             dot.node(x)
523             dot.node(y)
524             dot.edge(x, y)
525             if y == document_id and x == visitorid:
526                 dot.attr('node', shape='box', style='filled', fillcolor='green')
527             else:
528                 dot.attr('node', style='filled', fillcolor='white')
529     print(dot.source)
530     filename = dot.render(filename='img/g1', view=True)
531     print(filename + " has been created")
```

The above listing shows how all methods are done with comments saying what the lines do. I will describe Task 2a in great detail and the other tasks dont require as much explanation as they are similar and can be understood easily using the comments. For task 2a the method is called bycountry where incoming parameters are the Dataframe data(on launching the main cw2.py), and the document_id for which country views need to be found(thus reducing the amount of data passed on).Next it is grouping is done depending on the visitor_country field using pandas groupby() method, which gives a dictionary(key ,value pair) can be easily accessed, added to the list and then passed on to the plothistogram method in Plot.py to be made into a histogram by the matplotlib library.

For Task 4 it is required to "from collections import Counter" as this will be used to get the joint readers. For task 5 **PLEASE NOTE THAT the following lines are edited if required**.

```
1 from graphviz import Digraph
2 import os
3 os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/' #path to
      dot.exe for running graph
```

```
#PLEASE EDIT THE ABOVE PATH FOR GRAPHVIZ TO SHOW :D
```

For tasks 4 and 5 key difference is that the pandas data is checked with either doc_id or visitor_uuid and also the event type(only read docs/visitors that read docs will be gotten). Using this we then groupby() as required

plot.py

```
import matplotlib.pyplot as plt
#Reference :https://www.macs.hw.ac.uk/~hwloidl/Courses/F21SC/Samples/simple_histo.py
def plothistogram(title, x_axis, y_axis, orientation='vertical'):
    #Checks for orientation by def. vertiacal histograms more pleasing to the eye
    #Slides was additional resoource
    if orientation=='vertical':
        histogram = plt.bar
        histogramlabel = plt.ylabel
        histogramticks = plt.xticks
    elif orientation=='horizontal':
        histogram = plt.barh
        histogramlabel = plt.xlabel
        histogramticks = plt.yticks
    else:
        raise Exception ('plot_histo: Invalid orientation')

    length = len(x_axis)
    histogramticks(range(length), x_axis)
    histogram(range(length), y_axis, align='center',alpha=0.4)
    histogramlabel('Number of times document is accessed')
    plt.title(title)
    plt.show()
```

Figure 15: Plot.py

Figure **??** shows the plot.py file. In this class we import the external library matplotlib and print a plot using the method plothistogram(Code here is taken using lectures or referenced/simple_histo.py) which is generic and can plot graphs with data it receives from data.py.

guitest.py

This is the module that consists of the class that is responsible for gui. Figure **??** shows excerpts of the class

```
from tkinter import *
import tkinter.messagebox

import data
class gui:
    #Variables
    root = Tk()
    global entry2a
    global entry5
    global uuid


    def __init__(self, dataframe):
        #define panda datframes obj &json file both
        self.dataframe= dataframe
#task2 comps
        label_task2 = Label(self.root, text='Task 2')
        label2a = Label(self.root, text='Document ID')
        self.entry2a = Entry(self.root)
        task2a_button = Button(self.root, text='Plot by country', command=self.task2a)
        task2b_button = Button(self.root, text='Plot by continent', command=self.task2b)

        label_task2.grid(row=0, columnspan=4, sticky=W)
        label2a.grid(row=1, column=0)
        self.entry2a.grid(row=1, column=1)
        task2a_button.grid(row=2, column=1, sticky=W)
        task2b_button.grid(row=2, column=2, sticky=W)

        hint_task2 = Label(self.root, text='Enter document id to get histo by country/continent')
        hint_task2.grid(row=1, column=2, sticky=W)

#For task 3 comps
        label_task3a = Label(self.root, text='Task 3a')
        label_task3a.grid(row=3, column=0, columnspan=4, sticky=W)
        task3a_button = Button(self.root, text='Verbose-Browser plot', command=self.task3a)
        task3a_button.grid(row=4, column=0, columnspan=2, sticky=W)

        label_task3b = Label(self.root, text='Task 3b')
        label_task3b.grid(row=5, column=0, columnspan=4, sticky=W)
```

(a) Imports in gui class, the variables and also some component initialization

```
if (len(sys.argv) <= 1):
    #Use gui if 1 or more args on python promt/ MEANING: py cw2.py opens gui -----  py cw2.py blah blah opens CLI
    #Cant do <1 cause 3a 3b dont take additional args so py cw2.py 3a open gui instead of cli
    gui = guitest.gui(dataframe)
else:
    #use cli
    command = argparse.ArgumentParser(description='F20SC CW2 Data Analysis with issuu dataset')
    command.add_argument('-u', '--user_uuid',
                         help='Enter the uuid of a user.')
    command.add_argument('-f', '--file_name',
                         help='Enter the file name of the json.')
    command.add_argument('-d', '--doc_uuid',
                         help='Enter the document_id of the document.\n used for Task 2a, 2b and 4d')
    command.add_argument('-t', '--task_id',
                         help='Enter a tasks id to run these can be one of \n 2a 2b 3a 3b 4d 5')
    # managing incoming arguments and sending them to their related functions
```

(b) Methods called by button click

Figure 16: cw2.py module

This is not commented in the code but the tasks requiring doc_id or visitor_uuid have entries that are gotten using self.entryvariablename.get() eg:- for 2a the variable is is entry2a so the entry is gotten using self.entry2a.get() and passed along with the pandas dataframe object to

the method that performs task2a in data.py (bycountry). Also, self.mainloop() lets us use the method as many times as we want as it is in a loop (as the name suggests).

TESTING

Testing for task was carried out using data provided in a mail by Ms Smitha (also scalability testing because the json file had 600k lines). The readings for task 4a were initially wrong cause I only took into account the document id and not wheter theyy had been read

```
and you should get these documents as result (see also the attached graph; you don't need to
print the 'joint readers' in Task 4, but it's useful to have this information and turn it on as
a command-line flag)
The readers of document  130601015527-c1e2993d8290975e7ef350f078134390  are:
   ['1f891eb0b573e42c', '3f64bccfd160557e', '383508ea93fd2fd1']
```

(a) Expected outcome

```
if (len(sys.argv) <= 1):
    #Use gui if 1 or more args on python promt/ MEANING: py cw2.py opens gui -----  py cw2.py blah blah opens CLI
    #Cant do <1 cause 3a 3b dont take additional args so py cw2.py 3a open gui instead of cli
    gui = guitest.gui(dataframe)
else :
    #use cli
    command = argparse.ArgumentParser(description='F20SC CW2 Data Analysis with issuu dataset')
    command.add_argument('-u', '--user_uuid',
                         help='Enter the uuid of a user.')
    command.add_argument('-f', '--file_name',
                         help='Enter the file name of the json.')
    command.add_argument('-d', '--doc_uuid',
                         help='Enter the document_id of the document.\n used for Task 2a, 2b and 4d')
    command.add_argument('-t', '--task_id',
                         help='Enter a tasks id to run these can be one of \n 2a 2b 3a 3b 4d 5')
```

(b) Outcome got

Figure 17: cw2.py module

Figure 17 is proof that 4a works so by extension both 4b and 4c work (minus the joint readers bit)

so the test is a success

Task 5 was done last minute and wasnt extensively tested but it is partially correct(Works some cases) So the test fails overall.

Tasks 2 and 3 were tested only by myself and compared with results others in the class got and these were equal(so expected outcome is assumed to be right). So the test is a success

Due to a hectic schedule with exams being around the time of submission extensive testing could not be carried out.

REFLECTIONS AND CONCLUSIONS

Python is a powerful and easy to learn language. Its used to analyse the data set (can even analyse large datasets with 600k lines) proves how powerful it is when used with the relevant libraries. Pandas is a really useful library and I am proud i was able to learn how it works as dataframes within a short span of time. This application would have been harder without pandas. Also the ability to run the application through both command line and also through GUI was a fun experience (The command line bit was fun to code). Graphviz was also fun to learn and use in the application although its results are a bit off given the results I get. On reflection, all tasks have been implemented (except 4d) but coding in python was a fun experience. To conclude, though this coursework was challenging it was made easier by the vast material availible online (which have been duly mentioned in the references) and also the lecture slide and the sample codes providing many functions that made it possible to develop this application.

REFERENCES

The following list of urls helped me in developing the analyser.

CLI using argparse

- https://docs.python.org/3.3/library/argparse.html
- https://www.youtube.com/watch?v=XYUXFR5FSxI

Stack Overflow links

- https://stackoverflow.com/questions/22691010/how-to-print-a-groupby-object
- https://stackoverflow.com/questions/11829422/counting-element-occurences-in-nested-lists
- https://stackoverflow.com/questions/14734533/how-to-access-pandas-groupby-dataframe-by-key https://stackoverflow.com/questions/10636024/python-pandas-gui-for-viewing-a-dataframe-or-matrix

Using tkinter :- https://www.youtube.com/watch?v=wNBqM28MMjs

Code References

The following code samples have been used in my application to an extent:

- plot.py uses code from
- GUI development was started using code from https://www.macs.hw.ac.uk/ hwloidl/-Courses/F21SC/Samples/feet2meter.py