



# DATA ANALYSIS OF A DOCUMENT TRACKER

Industrial Programming

F20SC  
Hans-Wolfgang Loidl

Stuart Marples – SM861  
H00156296

## Contents

Contents .....	2
Introduction .....	3
Requirements Checklist .....	4
Design Consideration .....	5
User Guide .....	6
Developer Guide .....	9
Testing.....	13
Reflections on programming language and implementation .....	15
Conclusions .....	16

## Introduction

Throughout this report I will be talking about the Data Analysis of a Document Tracker program that has been created and the process that went into creating it. There will be a user's guide which will describe how to use the program for new users and supply a developer's guide for any developers that may want to understand and develop more to the program. Furthermore, there is a testing section to show how effectively the program works and what bugs and problems it may run into.

The Data Analysis of a Document Tracker will contain features such as counting the number of viewers from different countries and continents and displaying this data in a histogram. It will also find out the most used browser through all the viewers and the display the top 10 viewers of the site on a histogram showing the number of documents that they have read. Finally, it will allow the user to see various other "related" documents which will consist of a list of read documents from users who have read the current document.

The project was created using Python 3 and using the json, matplotlib and tkinter libraries which will be discussed in more detail in the Design Considerations section.

## Requirements Checklist

Requirement	Completed?	Comments
Views Country – Takes Document UUID	Yes	
Views Country – Display as histogram	Yes	
Views Continent – Takes Document UUID	Yes	
Views Continent – Display as histogram	Yes	
Views by browser – Display as histogram	Yes	
Views by browser – Display only browser name	Yes	
Reader Profiles – Display top 10 readers	Yes	
Also Likes – Use document UUID to return Visitor UUIDs of readers	Yes	
Also Likes – Use visitor UUIDs to return Document UUIDs that have been read by visitor	Yes	Some documents do not have a document UUID so instead it will return them as “unknown”.
Also Likes – Use visitor(optional) and document UUID and sorting function to return list of “liked” documents sorted by sorting function parameter	Yes	
Also Likes – list documents sorted by the readership profile for sorting the documents	Yes	Sorts the documents into the top 10 that have been read for the longest amount of time, but since not a lot of data stored the time spent being read, the time could be set to 0.
Also Likes – list documents sorted by the number of readers of the same document	Yes	
Also Likes – Provide Document UUID for above two cases and produce top 10 documents	Yes	
Also Likes – optionally provide visitor UUID for above two cases and produce top 10 documents	Yes	
GUI – Created GUI to input and display data	Yes	
Command Line Usage	Mostly	You can enter the details and view the data correctly. But with the first 4 tasks the form does not load properly in the background and loads a small empty form.

## Design Consideration

To store the various data such as the name of countries, continents or browsers with a count I used a dictionary data type. This allowed for easy addition, sorting and deletion depending on what is needed by the function being used.

For the creation of the various functions, I used a for loop which would iterate through the details of every line of information in the json file which would then be able to find specific details from the line such as country a document was accessed from. This would then get added to the dictionary if it did not exist already in the dictionary, if it did exist it would increase the count of the country that has accessed the document. A disadvantage to my implementation of this is that I could not manage to put the reading section of code into its own function and therefore have reused a similar chunk of code in most functions.

I utilised various libraries for the creation of this project. The json library was used to parse through the large amount of data supplied by Issuu to find specific details. The advantage to this library is its common use which made it easy to find support for any issues I had trying to parse the data and it is an easy library to use for parsing through data.

The matplotlib library was used for the creation of the various histograms used in the project. An advantage to using this library was that it allowed for easy use to transfer details from a dictionary into a histogram to display the data. It was also easy to use and allowed for different histograms to be created using various without needing to change the code meaning I could use one function to create all the histograms.

I also utilised the OrderedDict library which helped in ordering the Also Likes function to display the top 10 values for the various sorting methods. This was a useful library because it allowed me to organise a dictionary for displaying without creating another function or using a lot of complex methods. But this library was only utilised once through the project for sorting the Also Likes function.

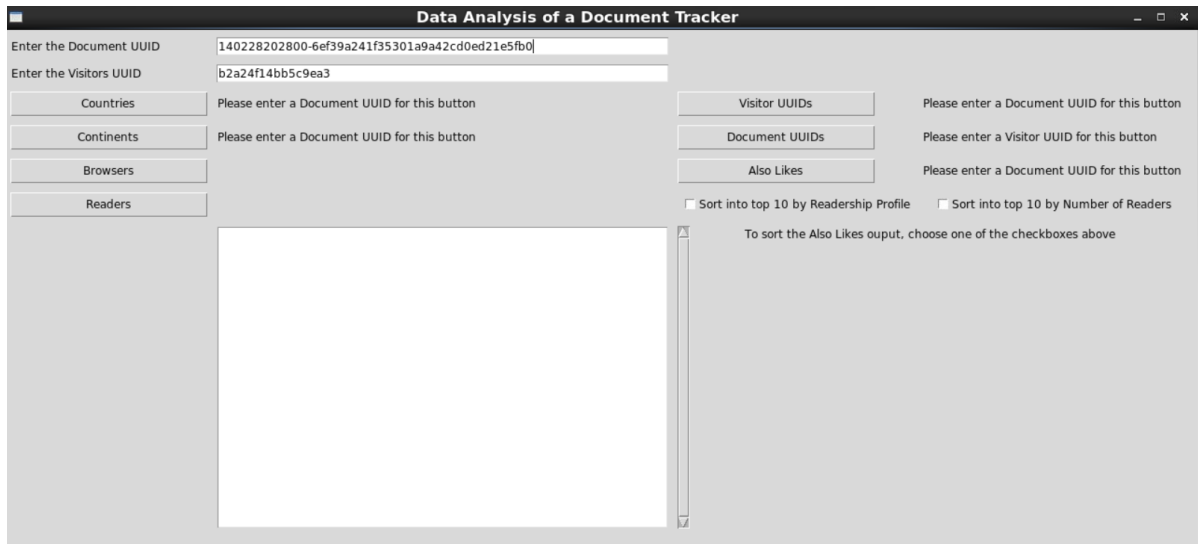
Finally, the tkinter library was used to create the Graphical User Interface (GUI) to allow the user to input data and click buttons to run the various functions. An advantage to this library was the ease for creating the GUI with the positions for buttons and labels where I wanted them and the simplicity for creating a label as it would only take 1 line while with some other libraries and languages I have used in the past it has been more complex.

Python was a good language to use throughout the creation of this project as it is easy to use and allows for quick error handling throughout the project. The simplicity of the language allowed for easy implementation of the various libraries and quick creation of the functions.

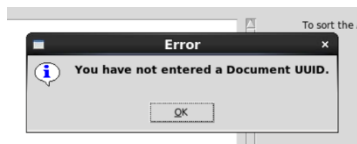
While reading in the json files I realised that there is not always a time or document UUID supplied with every line of data, so if no time was supplied it would be set to 0 and if no document UUID was supplied it would be stored as "unknown".

## User Guide

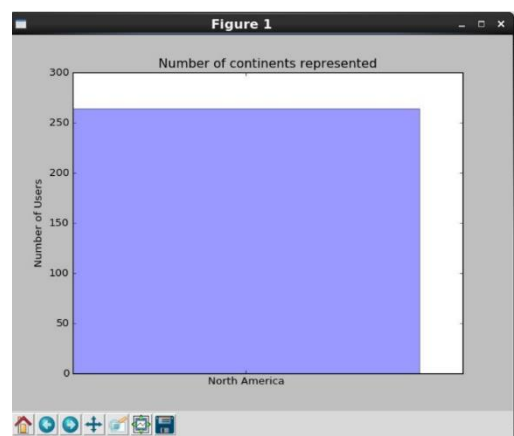
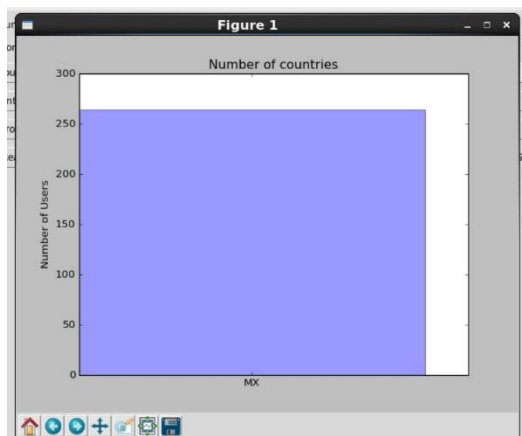
When the user starts the program, they will be presented with the home page which will have some default data in the two entry boxes for document UUID and for a user's UUID that can be changed by the user.



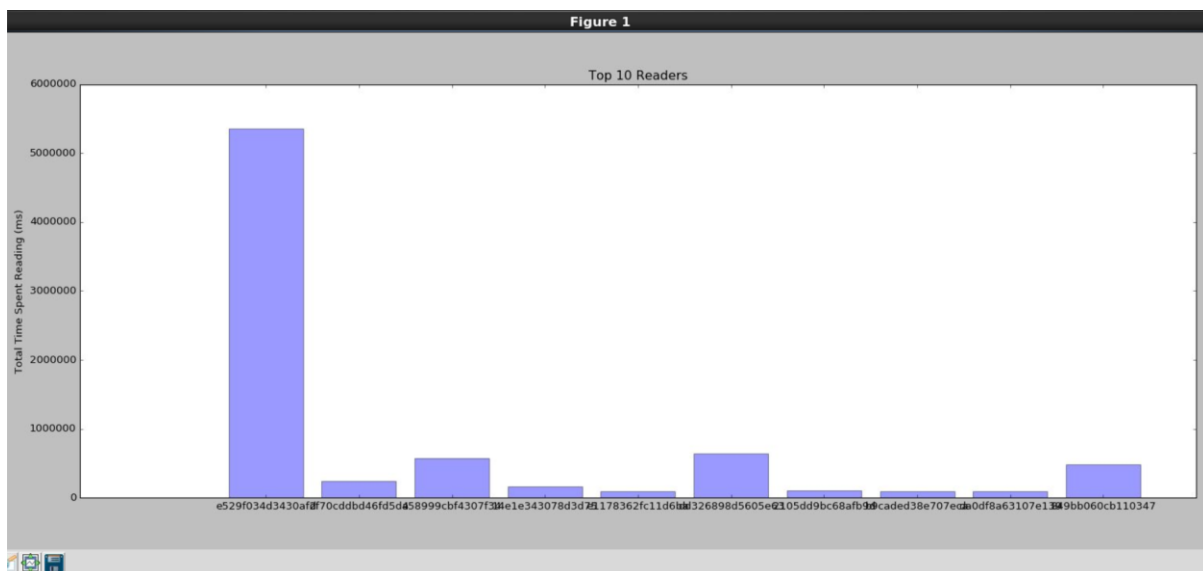
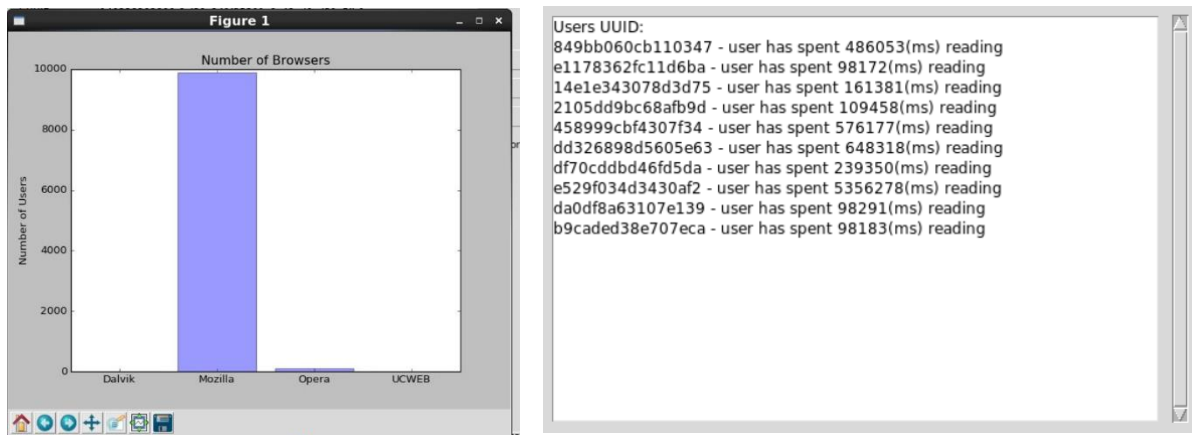
For each button, it states whether you need to enter a document UUID or a visitor UUID to run. If this has not been entered or gives no results, the user will be displayed with an alert to tell them so. This alert will occur if any function delivers no results and if any buttons that require a Document UUID or visitor UUID are not supplied with one.



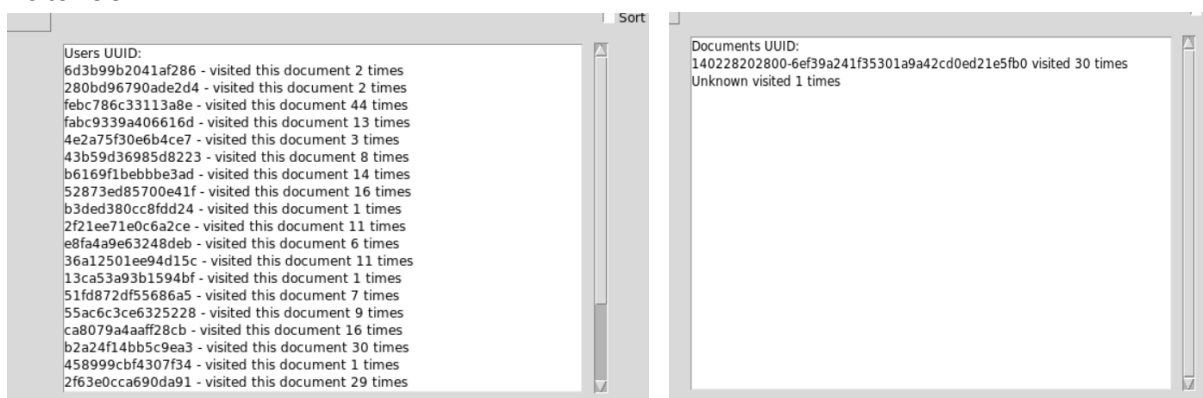
When the user supplies a viable document UUID and press the Countries button or the Continents button, they will be supplied with a histogram that displays the results of this data.



The next two buttons do not require a user UUID or a document UUID to display their data. When pressed, the Browsers button will create a histogram of the browsers used to access documents. The Readers button will create a histogram and fill the listbox with details of the top 10 viewers that have spent the longest amount of time reading documents.

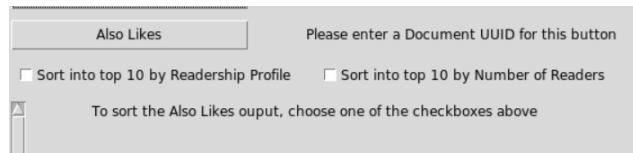


The Visitor UUIDs button will return a list of the visitor UUIDs that have read a specific document supplied by the user, with a count of the number of times the visitor read the document. The Document UUIDs button will return a list of the document UUIDs that have been read by a specific visitor UUID.



There is finally the Also Likes section where you enter a document UUID and optionally, a visitor UUID for a document to receive a list of document UUIDs that other visitors who have read the original document have read.

This output can be sorted into a list by checking one of the two available checkboxes if the user wishes to only see the top 10 in a specific order. It can sort them into the top 10 documents from the number of readers the documents had or into a list of top 10 documents based on the amount of time that the readers spent reading the documents.

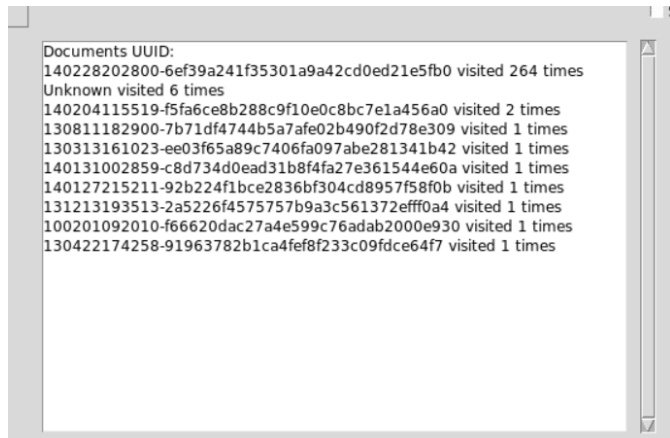


Also Likes

Please enter a Document UUID for this button

☐ Sort into top 10 by Readership Profile ☐ Sort into top 10 by Number of Readers

To sort the Also Likes output, choose one of the checkboxes above



Documents UUID:

140228202800-6ef39a241f35301a9a42cd0ed21e5fb0	visited 264 times
Unknown	visited 6 times
140204115519-f5fa6ce8b288c9f10e0c8bc7e1a456a0	visited 2 times
130811182900-7b71df4744b5a7afe02b490f2d78e309	visited 1 times
130313161023-ee03f65a89c7406fa097abe281341b42	visited 1 times
140131002859-c8d734d0ead31b8f4fa27e361544e60a	visited 1 times
140127215211-92b224f1bce2836bf304cd8957f58f0b	visited 1 times
131213193513-2a5226f4575757b9a3c561372efff0a4	visited 1 times
100201092010-f66620dac27a4e599c76adab2000e930	visited 1 times
130422174258-91963782b1ca4fef8f233c09fdce64f7	visited 1 times



## Developer Guide

The various python files have been split depending on what they store. There is a file named “cw2.py” which only contains the code to start the program and the code for the command line interface, “Coursework.py” which contains the code for the GUI, a file named “myConstants.py” which contains the constant values for a countries two letter code and then constant values that relate this two-letter code to a continent and a fourth file which contains all the functions used by this program named “myFunctions.py”.

The cw2.py file was used to run the program normally and to store the code for the command line interface. There is a function that is used to find the task that has been entered and run the appropriate function for the task number. There is also a series of if statements to correctly read in the users input to see if they have entered a visitors or documents UUID or both and request that they do so if they attempt to run a task that requires one of these values. If this fails it will load the program normally for the user to attempt to use as an alternative,

```
def taskNumber(task):
    if task == "1":
        if DUUID == "":
            print("You must enter a Document UUID to run this task")
        else:
            myFunctions.findCountry(DUOID)
            Coursework.GUI()
    elif task == "2":
        if DUUID == "":
            print("You must enter a Document UUID to run this task")
        else:
            myFunctions.findContinent(DUOID)
    elif task == "3":
        myFunctions.findBrowser()
    elif task == "4":
        myFunctions.findBrowser()
```

The file Coursework.py was used for the various GUI functions that would create the main form and populate it with the appropriate buttons, entry boxes, labels, checkboxes and a listbox. The listbox and scrollbar were made global due to the listbox being utilised in various functions for entering

```
#code for the scrollbar and the listbox
yScroll = tk.Scrollbar(mainframe, orient=tk.VERTICAL)
yScroll.grid(column=3, row=9, sticky=N+S+W)
lb = tk.Listbox(mainframe, width = 65, height = 20, yscrollcommand=yScroll.set)
yScroll['command'] = lb.yview
```

data. I utilised the grid method which splits the form into a grid format allowing for easy placement of objects in the form and used N, W, E and S to place an object in the top, left, right or bottom of the grid it has been placed in for slightly more precision. After that the main GUI function is there to place the non-global variables used in the main creation of the form. This includes the two entry boxes used to enter the document UUID and viewer UUID which start by displaying default data from the issuu\_cw2.json file. The creation of the buttons and labels are like the listbox where they use the grid method with a width and appropriate text set, but have the command set to run a function from the “myFunctions.py” file with the possibility of passing through the document or viewer UUID that has been entered. Each button is accompanied by a label which dictates what values need to be entered for the use of that button. For example the code below uses the “findVisitor” function which will use the document UUID entered to view the visitors of the document and tells the user that this button requires the document UUID.

```
ttk.Button(mainframe, width = 25, text="Visitor UUIDs", command=lambda: myFunctions.findVisitor(docUUID.get())).grid(column=3, row=3, sticky=W)
ttk.Label(mainframe, text="Please enter a Document UUID for this button").grid(column=4, row=3, sticky=W)
```

There are multiple functions created to enter the appropriate details into the listbox for the ease of

the user which will include stating at the top what is being shown and displaying it in a more readable format. The only difference to these functions tends to be

```
#function to insert Users UUID data into the listbox
def lbInsert(values):
    lb.delete(0, lb.size())
    lb.insert(0, "Users UUID:")
    for v in values:
        lb.insert(1, v + " - visited this document " + str(values[v]) + " times")
```

the wording in the for loop and for the certain sorting functions it will display the data in the appropriate order and not randomly. There are finally multiple functions created to display error messages such as the error message stating there is no visitor UUID entered when one is required to run the function.

The “myConstants.py” file is just a long list of country codes to continent codes and continent codes to continent names.

```
#constants to change 2 letter continent code to full name
continents = {
    'AF' : 'Africa',
    'AS' : 'Asia',
    'EU' : 'Europe',
    'NA' : 'North America',
    'SA' : 'South America',
    'OC' : 'Oceania',
    'AN' : 'Antarctica',
    'ZZ' : 'Unknown'
}
```

Finally, the “myFunctions.py” file stores all the non-GUI functions used throughout the program. There is one global variable named “alsoLikes” which is originally set to False as it dictates when the alsoLikes function is being used. This was created because the alsoLikes function uses the findDocument and findVisitor functions which will output data on completion so alsoLikes is set to True, it will not output the data in those functions and returns the data for manipulation by the alsoLike function.

The first function is findCountry which takes a document UUID as input and checks to make sure there is a UUID entered for use and otherwise will output an error message using one of the error functions mentioned above in the Coursework.py file. It will then begin to read through the json file using the repeated section of code displayed below. Unfortunately, I could not get this into a function

of its own and therefore was reused in every function in this file. It will start by reading the entire file line by line and then will find a specific value in each line to store for use in the function. In this case also tries to match the document UUID of the current line to the UUID entered by the user and if this fails will move on to the next line in the json file but if it succeeds it will assign to a variable which in this case is “visitor\_country”. It will then see if this country name is already in the dictionary

```
country = ind['visitor_country']
existCountry = False
for c in countries:
    if c == country:
        countries[country] = countries[country] + 1
        existCountry = True
if existCountry == False:
    countries.update({country : 1})
```

of values and if it is will add 1 to the counter for that country and if not will add the country to the dictionary with the starting count of 1. I utilised another Boolean variable “existsCountry” to see if a value was added to the dictionary by the end of the for loop as if it was it would be set to True and if nothing had been added it would be stored as False so that the country could be added to the dictionary only once. There will be a final check to see if any results have been found and if not, an error message will be displayed to the user to alert them to this, otherwise the data will be passed

```
show_histo(countries,orient="vert",label = "Number of Users", title="Number of countries")
```

through to the show\_histo function for display along with an orientation, and axis names and titles.

The next function is the findBrowser function which is very similar to the previous function where it will read through the file and set the variable to be the “visitor\_useragent”. And then will display the data in the same way as shown above but there is an extra line that was used to split a lot of the excess detail away from the browser name. All the browser names were stored at the start with a “/” followed by a version number which allowed me to look for the first forward slash and store the values in front of it and after it. This then let me store the exact browser names such as “Mozilla”.

The findTime function was created for sorting in the alsoLike function where it would take in a document UUID and a visitor UUID and find the the user spent reading this document and would return this time. There were a few tries and excepts in this because not every line in the json file stores the time or the document UUID and sometimes its under a different variable name so there are a few attempts at finding the required details and if unable they will return the time with 0. If they can find and match both of the UUIDs the time will then be returned if one is available.

```
try:
    docId = ind['env_doc_id']
except KeyError:
    try:
        docId = ind['subject_doc_id']
    except KeyError:
        return 0
#If the document and visitor UUID match the ones requested,
if DUUID == docId and visId == VUID:
    try:
        time = ind['pagereadtime']
    except:
        try:
            time = ind['event_readtime']
        except:
            time = 0
    return time
```

The alsoLike function aims to find the visitors of the document that the user has entered and then find the documents that those visitors have viewed which creates a list of documents that the user may also like. This function starts by setting the alsoLikes variable to True so to not get an incorrect output when other functions are called. This function utilised 3 separate dictionaries in its implementation, one to store the document UUIDs with a count, one to store the visitor UUIDs, and another to store the time spent reading a document for use in the readership profile sorting algorithm. The function would check to see if a sorting algorithm is applied and if they are both checked then an error message is displayed and nothing occurs. It then checks to see if the user has entered just a document UUID or both a document UUID and a visitor UUID and run the appropriate findDocument or findVisitor function on these values. If there is a visitor UUID it will add the visitor to the dictionary and add all the documents they have read to the documents dictionary with the number of views which will have been returned by the findDocument function. If the user didn't enter the visitor UUID then the program will skip to this step where it will use the list of visitors to find every document that those visitors have read and add it to the dictionary or increase the count assuming that the visitor is not already in the dictionary as we don't want to count the same visitors views more than once.

This uses the same methodology as the previous functions but with more for loops and complexity due to the extra dictionaries. At the end of the function there

```
if nr == 1:
    sortReader = {}
    temp = [(k, documents[k]) for k in sorted(documents, key=documents.get, reverse=True)[:10]]
    for t in temp:
        sortReader.update({t[0] : t[1]})
    Coursework.lbSortInsert(sortReader)
elif rp == 1:
    sortReader = {}
    temp = [(k, rpTime[k]) for k in sorted(rpTime, key=rpTime.get, reverse=True)[:10]]
    for t in temp:
        sortReader.update({t[0] : t[1]})
    Coursework.lbSortRpInsert(sortReader)
else:
    Coursework.lbDocInsert(documents)
alsoLikes = False
```

is an if statement which will find what sort of sorting algorithm has been activated. If none then it will print all of the values, but if there is a sorting algorithm then it will find the top 10 values of its required field and store them into a temporary variable. This is because they do not get stored as a dictionary so we then add these top 10 values into a dictionary and pass them into the appropriate listbox insert function which will sort the top 10 into order for display. It will then set alsoLikes back to false when the function is finished running.

The next two functions are the findDocument and findVisitor functions which are almost identical except for the values that they read from the json file. They both will read in each line from the json file and compare it the UUID supplied and if matched will add this value to the dictionary or increase the count for it in the dictionary. The only difference between these functions and others is the alsoLikes if statement which will stop the data being display when run and will instead return the data for use in the alsoLike function.

```
if alsoLikes == False:
    if not bool(documents):
        Coursework.noResults()
    else:
        Coursework.lbDocInsert(documents)
```

The findReader function works similarly to the above functions to find a list of all readers in the json file with the appropriate count for how many documents they have read. But then it will find the top 10 readers by using the same method as the alsoLike function above where it will find the top 10 and then re enter them into a dictionary for display in a histogram and are sent off to a function in Coursework.py for display in a listbox.

The show\_histo function will take in the data, orientation, labels and titles. It will then create the histogram correctly depending on the orientation that was passed through or throw an error if an unusable orientation was passed through. It will then populate the histogram with the data that was passed through and assign the label and titles to the graph.

```

bar_label = plt.ylabel
elif orient=="vert":
    bar_fun = plt.bar
    bar_ticks = plt.xticks
    bar_label = plt.ylabel
else:
    raise Exception("show_histo: Unknown orientation: %s" % orient)
n = len(dict)
bar_fun(range(n), list(dict.values()), align='center', alpha=0.4)
bar_ticks(range(n), list(dict.keys()))
bar_label(label)
plt.title(title)
plt.show()

```

The final function is the findContinent function which is exactly the same as the findCountry function except there are two lines that will change the country codes to continent codes and then continent codes to continents using the myConstants file.

```

country = find_reader_country
miniCont = myConstants.centry_to_cont[country] #!
continent = myConstants.continents[miniCont] #u!

```

## Testing

The testing being done will utilise the “issuu\_cw2.json” file for input as the test data.

### Document UUID input

Test	Expected Output	Actual Output	Comments
Document UUID = “140228202800-6ef39a241f35301a9a42cd0ed21e5fb0” on countries	Histogram showing viewers countries that have viewed this document.	Histogram showing viewers countries that have viewed this document.	
Document UUID = “140228202800-”	Message saying this receives no output.	Message saying this receives no output.	
Document UUID = “”	Message saying that no document UUID has been supplied.	Message saying that no document UUID has been supplied.	
Document UUID = “Test”	Message saying this receives no output.	Message saying this receives no results.	
Document UUID = “140206010823-b14c9d966be950314215c17923a04af7” on continents	Histogram showing viewers countries that have viewed this document.	Histogram showing viewers countries that have viewed this document.	
Document UUID = “!£\$%^&*()_+=”	Message saying this receives no results.	Message saying this receives no results.	
Document UUID = “100201092010-f66620dac27a4e599c76adab2000e930” on visitor UUID	Output listing the various users who viewed this document.	Output listing the various users who viewed this document.	

### Sorting Algorithm Input

Test	Expected Output	Actual Output	Comments
No checkboxes selected	List of all values from Also Likes.	List of all values from Also Likes.	
Readership Profile sort selected	Output the top 10 documents with the most time spent being read.	Output the top 10 documents with the most time spent being read.	

Number of reader's sort selected	Output the top 10 documents with the most viewers.	Output the top 10 documents with the most viewers.	
Both selected	Error MessageBox saying you cannot have both selected.	Error MessageBox saying you cannot have both selected.	

### Visitor UUID Input

Test	Expected Output	Actual Output	Comments
Viewer UUID = "b2a24f14bb5c9ea3" on Document UUIDs	List of documents that the viewer has read.	List of documents that the viewer has read.	
Viewer UUID = "b2a24f14"	Message saying this receives no output.	Message saying this receives no output.	
Viewer UUID = ""	Message saying that no Viewer UUID has been supplied.	Message saying that no Viewer UUID has been supplied.	
Viewer UUID = "Test"	Message saying this receives no output.	Message saying this receives no output.	
Viewer UUID = "140206010823-b14c9d966be950314215c17923a04af7" On Document UUIDs	List of documents that the viewer has read.	List of documents that the viewer has read.	
Viewer UUID = "!£\$%^&*()_+="-	Message saying this receives no output.	Message saying this receives no output.	

Overall testing went well and resulted in no errors. Although I could have handled the Document and Visitor UUID better by taking their size into account as all of the Document UUIDs and all of the Viewer UUIDs are the same length and then could use this information to output a more detailed analysis of what was wrong.

## Reflections on programming language and implementation

Python was a simple language to learn and use, especially since I have limited experience using it as I have only used it for 1 year. The simplistic style that Python uses make it easy for anyone to read as there are less brackets everywhere in comparison to languages like Java which allows for such ease in picking up where you left off after a break. Some disadvantages to using Python were that errors only appeared at runtime which could become an issue when you have implemented a lot of new code at once as it causes a lot of stopping and starting to find and fix these errors.

From my experience developing software on a Linux system was better than developing on a Windows system. On Windows, I utilised Visual Studio which is a much nicer application to use for development compared to Geany which I used to create the Python programs on Linux. But I found Linux easier to run overall and since I get to run the program on the same servers as required, it means there are less errors in the transfer of the programs while with windows there were changes to be made to make sure compatibility wasn't a problem and I still finished that project with an error that would only occur on the university servers that I could not figure out in the time that I had left myself.

## Conclusions

In conclusion, I liked using the language python to create this project as it was easy to find any errors in the code and allowed for easy creation of the various functions. Although, with more time, I would have liked to have fixed the reused code and put the reading from the json file into its own function so it could be called by the various other functions. I would have liked to fix the way some of the histograms are displayed as well as in certain situations such as showing the top 10 readers, the data can overlap such as the visitors UUID making it hard to read.