Industrial Programming Coursework B

Duncan Cameron
Software Engineering
Heriot-Watt University
dac31@hw.ac.uk
H00153427

December 1, 2016

Contents

Chapter 1

Introduction

For this coursework task I created a simple data analysis tool for a document tracker built in python.

1.1 Assumptions

One slight difference from the spec in my implementation is that I have combined task 2a and 2b into the same screen. When the task (-t) from the command line is either 2, 2a, or 2b the screen showing both views by country and views by continent will be displayed. Views by browser for that document are also displayed as an additional piece of functionality.

I also used an additional command-line argument to specify the datafile: -f

I also assumed that for all tasks other than 3 that I would only take in instances which have and event_type of read or pagereadtime. For task 3 I use every item in the datafile as specified. However, if this functionality is accessed through the GUI it will show for only the reads and pagereadtimes.

For section 5 I did not create specific functions as specified in a) and b). I obtain all of the related documents within 1 argument using a nested loop. I did this as it made the task more efficient as I could track the views and read-times as I iterated through.

Section 5 also mentions that a visitor id is required in addition to the document id but does not specify how this should be used. The way I am using it by having the algorithm not use that user to find the documents, so it is fully based off of the interests of other users who have viewed the document.

1.2 Report Summary

This report will cover:

- A requirements checklist
- A breakdown of the design considerations I made when implementing the project
- A user guide
- A developer guide
- Details on my testing process

- Conclusions from developing the project

Chapter 2

Requirements' Checklist

I completed all of the tasks required for the project:

2.1 Implementing in python

The program was implemented in python 3.5.2

2.2 Views by Country/Continent

For any given document the program will display a histogram showing both the views by country and by continent

2.3 Views by Browser

The program can show a histogram of the views by browser for every entry in the data-file. This includes an unsorted display of all different user-agents and sorted by browser family

2.4 Readership Profiles

The program will display the top 10 users with most time spent viewing documents

2.5 Also likes Functionality

For any document a list of up to 10 documents that will be 'also liked' - other documents read by users who have read this document. This can be sorted by either readership profile or number of views. In the implementation, any function can be passed in to sort it differently. It also allows for a user to be passed in, meaning that that user is ignored in finding related documents (as if it was being recommended to that user)

2.6 GUI Usage

There is a full GUI to allow a user to browser through the various functionalities that the system offers.

2.7 Command Line Usage

The program can be started using command line parameters which allows the program to be launched in certain ways for each task.

Chapter 3

Design Considerations

3.1 Data Representation Classes

I created 3 classes to represent the data that was read in from the input file: A Document class, a Reader class and a View class. I decided on this structure as it allowed for a simple, natural way to explore the dataset and the Documents and Readers could be linked by views.

3.1.1 Document

The Document Class stores information about the document, including the document UUID and a list of the views of the document. It contains functions to get the data required for creating relevant histograms for the document, as well as an 'also likes' function.

Also Likes

3.1.2 Reader

The Reader class contains the UUID and views of a reader (visitor). There is also a function to get the users total view time.

3.1.3 DocumentView

The DocumentView class links the Document and Reader class as well as holding additional information about each view - the time the page was viewed for, the user-agent for detailing the browser that was used for the view and the country from which it was viewed. I decided to store the user-agent and country here as opposed to the Reader class as it is possible for a reader to view from different countries or using different browsers.

3.1.4 DataLoader

To load the data I created a class called DataLoader used to load the data in from a JSON file. When created it takes in a filename and will access try to load it and create the relevant classes based on the data.

It contains 2 fields for accessing the readers and documents. They are both dictionaries in which the key is the UUID and the value is the object. I chose this approach as it means that if a key is given the related document can be accessed in O(1) time.

The data loader is also able to either load in certain types of data based off of the event type of each instance in the file by taking in an optional boolean parameter in its constructor. If the value is true it will use all of the event types (as is required by task 3). Otherwise it will only take in 'pageread' and 'pagereadtime' instances as they represent actual reads of the document.

I used the data loader as a class instead of a series of global variables and functions so that multiple files could be loaded and independently managed. While this is not done in the program, it would be possible by simply creating another instance of the DataLoader class. This is also a likely extension of the program as it would be useful for situations such as comparing multiple datasets.

3.2 GUI Classes

To implement the GUI I used the tkinter library. My implementation involved creating a few classes. The Controller class was used to control what page was viewable. The NavigationPage class was used to represent the main page from which many things could be accessed, the GraphPage was used to represent the page displaying the histograms and the HeaderFrame was used for the top half of each graph page.

3.2.1 Controller

The main function of the Controller is to control the GUI pages, including making sure they can access the relevant data that they need. It extends the Tk class to give it the functionality to switch out which frame is active and where the frame should be placed (in this case, the whole window). To give them access to the data it creates a DataLoader object. It then creates stores both the GraphPage and the NavigationPage and methods to load each page.

3.2.2 NavigationPage

The NavigationPage will perform all of its functionality when it is initialised. It will create the required labels and buttons, which have their commands specified using lambda expressions.

3.2.3 GraphPage

The GraphPage controls the page on which the histograms are displayed. Each time this page is loaded, all of the items on it are cleared and the new items are created and added.

3.2.4 HeaderFrame

The HeaderFrame represents the top part of the GraphPage. It displays a button to return to the NavigationPage, a heading displaying the name of the document and a list of also liked documents.

3.3 GUI Design

I wanted to make the GUI easy to use so added extra functionality than required. In the main page I added a list of the most viewed documents and how many times they were viewed. There is also two text entry field that can take in a document uuid and a user uuid.

When viewing the data for a document, it is shown on a different page and will display 3 histograms: The views by country, by continent and by browser. Along the top of the screen the name of the document is displayed and a list of buttons for also liked documents - when the button is clicked that pages data will be shown.

3.4 Command Line Management

To effectively manage parameters passed into the program from the command line I used python's "getopt" method. This paired up the option and argument for all of the arguments (e.g. '-t' and '2a'). I could then assign the arguments to the correct variable without having to do much micromanagement. Then, using these variables I could run the correct task. When the GUI was required the code would launch the GUI and pass in a parameter which would make sure that it navigates to the correct page when launched.

3.5 Use of Advanced Language Features

Throughout the code for the project I made use of many of python's advanced language features to create cleaner and faster code. One thing I commonly used was lambda expressions for defining anonymous functions. This was often used for sorting. I also used in-line generators which allowed me to gather a sequence of items from 1 list. One time this was done in a nested fashion in `DataLoader.get_global_view_data` to get every view from every visitor.

3.6 Coding Style

I made an attempt to ensure that my code style followed the official python style guide[1]. This was aided by using the PyCharm IDE[2] which notified me of any style infringements.

In addition to how the code was laid out, I used other means to ensure that my code was clean, readable and maintainable, mostly by following guides from the book Clean Code[3]. This included aspects to creating high quality code that I followed.

One thing is variable naming. I made sure that all of my variable names were smart so anyone looking at a variable could tell what it is. I tried to avoid names like 'x' or 'val'. There were occasions where I created more abstract code that means a variable could be one of several things, which meant occasionally using names such as 'item'.

Another important aspect was appropriately defining functions. I tried to keep my functions both smartly named and small. This helps keep functions stick to the single responsibility principal and makes it easier to read: its hard for a programmer to understand what is going on if they look at a long stream of code; breaking these up into functions make it clear what each part is doing.

I also made appropriate use of comments, by using them sparingly and only when my code fails to simply describe itself. Normally, good variable and function names can eliminate the need for lots of comments.

Chapter 4

User Guide

4.1 Launching the Program

The main gui for the program can be launched by running ./cw2 from a linux command line. If it fails either python is not installed or there may be need for certain extensions to be installed. Running pip install -r requirements.txt will install all required extensions. It may be a good idea to set-up a virtual environment[4] to do this.

4.2 Viewing a Document's Data

To view the data for a document, either click it from the "Most viewed" list or enter its uuid in the "Document Search" field and press the search" button.

4.3 Readership Profile

When the GUI is launched, a readership profile of the top 10 readers is displayed on the right of the screen.

4.4 Global Browser Views

To view data on the global browser views, click the "global browser views" button on the main page. To view for all entries and for each individual user-agent, the command line must be used (see 4.5)

4.5 Command Line Tasks

4.5.1 General Usage

The command line parameters will either display the required information or launch the program and navigate to the appropriate screen. For general usage, launching the command line can be used as follows:

```
./cw2 -t task_id -d doc_uuid -u user_uuid -f file_name
```

4.5.2 Views by Country and Continent

To see the views by country and continent for a specific document use the following command where <doc_id> is a document UUID:

```
./cw2 -t 2 -d <doc_id> -f data.json
```

4.5.3 Global Browser Statistics

To view the global browser statistics for each individual user-agent use the following command:

```
./cw2 -t 3a -f data.json
```

To view the global browser statistics for each browser family use the following command:

```
./cw2 -t 3b -f data.json
```

4.5.4 Readership Profile

To see the readership profile of the top 10 readers use the following command:

```
./cw2 -t 4 -f data.json
```

4.5.5 Also likes Functionality

The also likes functionality can be used for two different sorting methods (by number of views and by the readership profile). To sort by readership profile, where <doc_id> is a document UUID, enter:

```
./cw2 -t 5d -d <doc_id> -f data.json
```

For sorting by number of readers of the same document enter:

```
./cw2 -t 5e -d <doc_id> -f data.json
```

The also likes functionality is also able to accept a user. For example, the following would sort by number of viewers where <user_id> is a visitor UUID:

```
./cw2 -t 5e -d <doc_id> -u <user_id> -f data.json
```

Chapter 5

Developer Guide

The project is split among 5 files:

- cw2
- countryinfo.py
- data_loader.py
- gui_controller.py
- view_data.py

5.1 cw2

cw2 is the main file for launching the application. It handles parameter inputs and launches the program in the way required.

5.2 Countryinfo

The countryinfo file simply contains two dictionaries - one linking country code to continent codes and one linking continent codes to continent names.

5.3 Data Loader

The data_loader file manages the loading of the data. All of its functionality is contained within the DataLoader class. The constructor takes in a filename which it will open and read. The data will then be converted into the data structures that the program (in view_data 5.5). The loaded documents are users are stored in the fields documents and visitors, each of which is a dictionary using the uuid as a key.

5.4 Gui Controller

The gui_controller file contains all of the user interface management. The main classes in this file are Controller, NavigationPage, GraphPage and HeaderFrame.

Controller manages the active displays and has a DataLoader to manage the data accessed by the GUI. It initialises a NavigationPage and GraphPage and contains functions to load them.

NavigationPage controls the main page from which a user can navigate to either the document graph pages or the global browser page. All of its functionality is set-up when it is initialised. A grid layout is used for this page.

GraphPage represents the page on which the histograms are displayed. Each graph is represented in a sub_plot so the histogram from matplotlib is embedded within the GUI. The graph page uses a pack layout as it is more dynamic for cases where there are many items of different sizes. The top of the page is handeled in a different frame: the HeaderFrame.

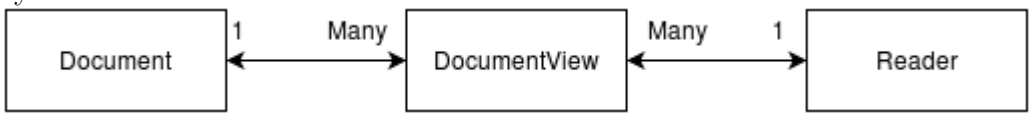
Header frame manages the top of the graph page. It contains a button to go back, a title and displays links to the "also likes" pages. A pack layout is also used here.

5.5 View Data

view_data.py contains the classes that represent the users, documents and views.

The Reader class represents a document. Its fields simply contain its UUID and a list of the users views. There is also a property to retrieve the users total time spent viewing documents.

The Document class contains more. In addition to storing the UUID and views, it contains many more functions.



The get_views_by_key function is used to retrieve a dictionary showing the number of views when a given property is a certain value. The key parameter is a function that will get the value from the view. There are other functions that use this function to show the views by country, continent and browser for easier access by defining the key in the function using lambda expressions.

The also_likes function finds the documents which are also read by users who have viewed this document. The function builds a dictionary for each document and tracks the number of views from unique users and the total read-time from all of the related users. It also takes in a function sort, which is used to sort the items.

The DocumentView class represents each view. the document and visitor variables relate to the document being viewed and the reader viewing the document. It also stores the time the document was viewed for, the country it was viewed from and the user agent. The country and user agent are stored here instead of in the user as a user may view from multiple browsers or country. With regards to the user agent, only the string is actually stored. The browser family is found using a property which will first check if the string is in a global dictionary variable called parsed_user_agents. If it is there it will use that, other-wise the string will be parsed using ua_parser and added to the dictionary. This approach is faster than parsing each one individually which can take quite a long time for large data sets. This way, any user agents which are the same as one already created are instantly accessed through the dictionary.

Chapter 6

Testing

To test my program I attempted to perform various tasks and ensure that they had the expected output. While I did not perform any formal testing, I was continuously making sure that any new functionality I added worked as expected and did not break other parts of the program.

While I was unable to make the program crash, there were areas that I could have improved on:

- When an invalid document UUID is entered in the GUI, there is no error message. While an exception is thrown and can be seen in the console, the GUI does not crash and will continue to function as expected.
- I also could have done some usability testing as my GUI was not very clear or usable.

Chapter 7

Conclusions

I believe that my program was well designed and implemented with a high quality of code. I met all the requirements and I was not able to detect any bugs.

I was happy with my overall code design and quality. I exploited the use of many advanced language features to make my program shorter and more reliable. For example, the Document get views by key function was able to be reused for different purposes easily instead of duplicating the code to provide the slightly different functionality.

I was also proud of my GUI as it embedded the histograms into the GUI and made use of multiple pages. It also showed the also likes with links so a user can navigate through them.

There are still some improvements I could have done to the project.

Unit testing would have been a good thing for me to set-up so I could have a proper testing infrastructure to ensure that all parts of my program are working as expected. Currently I have to manually test everything to make sure it works as expected.

My code could also have been made more stable and reliable by including more try-except statements. This would mean that my program would be more protected against bugs that I was unable to detect and would stop the program from completely crashing and I could handle how it should act should an error occur.

I would also have liked to see if I could improve my code even more using advanced features of python that I did not use, such as decorators

Bibliography

[1] PEP 8 – Style Guide for Python Code. https://www.python.org/dev/peps/pep-0008/.

[2] PyCharm. https://www.jetbrains.com/pycharm/.

[3] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, Upper Saddle River, NJ, 1 edition edition, August 2008.

[4] Virtual Environments — The Hitchhiker's Guide to Python. http://docs.python-guide.org/en/latest/dev/virtualenvs/.