Programming for Social Scientists: Concert Scraper `https://github.com/ksuchak1990/concert_scraper/`

Keiran Suchak

January 8, 2018

Contents

1 Aim

The code to which this document is attached has been written for the second assessment of the Programming for Social Scientists module. The scope of the assessment was left relatively open-ended, allowing students to design their own program. The main requirement was that the code produced should:

- Read in some data,
- Process it in some way,
- Display the results, and
- Write the results to a file.

With these points in mind, the aim of the project was to produce a program that documented the concerts that were going on in Leeds and the surrounding area.

Problem Specification

The code produced aims to address a problem faced in a somewhat unrelated facet of the programmer's life. The Music section of The Gryphon (the University of Leeds student newspaper, of which the author is a member) regularly assigns writers to attend and review concerts around the city of Leeds. This is achieved by putting a list of upcoming concerts on an online spreadsheet, to which all of the writers have access. This list is manually assembled on a monthly basis by the section editors, i.e. an editor spends time browsing a range of websites, collecting information on the upcoming concerts. Such a task can quickly become tedious as it requires a significant portion of the editor's time to browse through the hundreds of different concerts; furthermore, the task is subject to human error whereby an editor can incorrectly document a concert (or indeed miss a concert altogether). The aim of this project is therefore to use the Python programming language to automate the collection of such data, and to output it in a format that is usable by the editors. Beyond this, the program will produce visualisations, displaying trends in the concerts around Leeds, as shown in Figures 2, 3 and 4.

2 Implementation

The code presented was designed so as to carry out three discrete tasks:

1. The collection of data regarding the times and locations of concerts — the program achieves this by emulating the human behaviour of manually collecting such data by searching through web-pages. This procedure is known as Web Scraping.

2. The writing of this information to human-readable form.

3. The analysis and visualisation of trends in the concerts.

The software development approach applied for this project was something akin to the Iterative & Incremental Development style. The initial development phase involved constructing the core functionality encapsulated in points 1 and 2. Following on from this, further phases of development focused on iteratively improving software performance, refactoring and fixing bugs, as well as extending functionality to include point 3. This process of extending the code was made smoother as a result of the object-oriented approach that was undertaken (outlined in Section 2).

The program requires no local input files, and writes all output files to the `./output/` directory, with subdirectories allocated to each of the activities.

Object-Oriented Programming

The functionality described above has been implemented through the use of an object-oriented approach to programming. This involves the development of classes — something akin to user-defined data-types with certain attributes and methods associated with them. With this approach in mind, new classes can be built on top of existing classes, with the new subclass *inheriting* from the existing superclass. Such inheritance has been actively woven into the construction of this program, as shown in Figure 1.

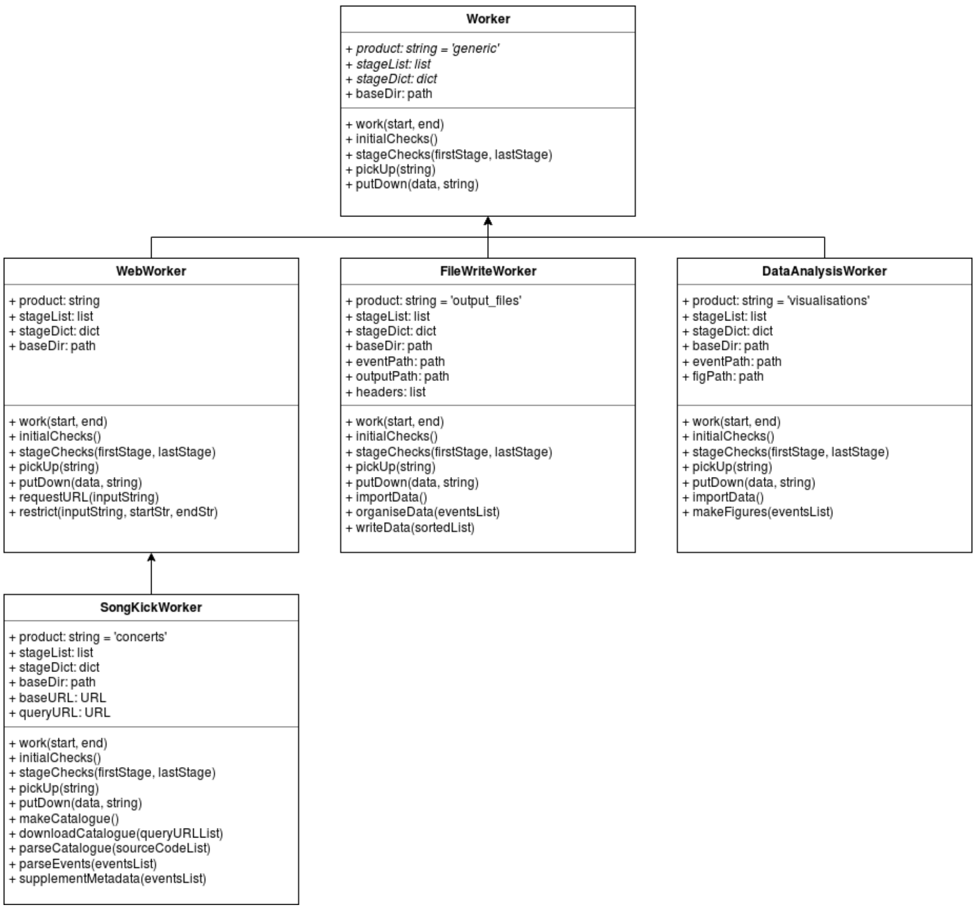

Figure 1: Class diagram showing the inheritance relationships of the Worker classes that have been devised. Italicised attributes and methods from the Worker class are overridden in subclasses.

Worker

The Worker class is the superclass from which all other constructed classes inherit. This class lays out the framework that other subclasses will use to undertake their specific forms of work. Such work is divided up into stages, each captured by a list attribute, allowing for corresponding stage functions to be called in sequence. This is predominantly achieved through three functions:

1. `work(start, end)` — Given start and end stages, this handles the calling of stage functions, reading the required data in preparation and writing the stage's output data.

2. `pickUp(path)` — This handles the aforementioned reading of required data. Crucially, this makes use of the `json` package, thus enabling the preservation of data structures between stages, whilst also allowing the writing function to remain generic.

3. `putDown(data, path)` — This is the counterpart writing function to the aforementioned `pickUp` function.

WebWorker (inherits from Worker)

The WebWorker class inherits directly from the Worker class, adding functions for interacting with web-pages. It has not been directly used in this project, but will likely be of use in future projects involving web scraping.

SongKickWorker (inherits from SongKickWorker)

The SongKickWorker class inherits from the WebWorker, and therefore from the Worker class. It is, in essence, the core of this project. This applies the functionality of the WebWorker class specifically to `https://www.songkick.com/`, which contains listing of concerts at venues in a region. The process of collecting data from this website is broken down into the following stages:

1. `makeCatalogue()`,
2. `downloadCatalogue(queryURLList)`,
3. `parseCatalogue(sourceCodeList)`,
4. `parseEvents(eventsList)`,
5. `supplementMetadata(eventsList)`,

which are documented within the source code.

3 Usage

Please note that prior to usage, the end-user should ensure that all of the pre-requisite packages (as detailed in Appendix A) are installed, using either `pip` or the user's choose of package management system. Each of the classes outlined in Section 2 can then be made use of as shown in Listing 1; line 3 of the listing crucially ensures that the packaged Worker classes are accessible to the script.

```
1  # Imports
2  import sys
3  sys.path.append('./Workers')
4  from Workers import SongKickWorker, FileWriteWorker, DataAnalysisWorker
5
6  # Running
7  s = SongKickWorker()
8  s.work(start='makeCatalogue', end='supplementMetadata')
9
10 fW = FileWriteWorker()
11 fW.work(start='importData', end='writeData')
12
13 dAW = DataAnalysisWorker()
14 dAW.work(start='importData', end='makeFigures')
```

Listing 1: Example code to make use of Worker classes.

4 Future Improvements

Although the program currently achieves the intended goal, there are still a number of ways in which it could be further developed or improved.

Manager Class

If constructed, a manager class would act as a wrapper class that would manage the running and interaction of each of the workers. This would add a further level of abstraction between the code and the end-user, with the aim that the end-user would only need to know the options that needed to be provided to the Manager class, and the Manager would take care of how such options affected the running of the workers. This could also help to apply the program to the collection of information on concerts in region other than Leeds.

Worker subclass to write to Google Sheets

The end goal of this project is to automate as much as possible the work undertaken by the Music editors in collecting and disseminating information of concerts. Whilst much work has been put into automating the collection side of this process, comparatively less has been put into the dissemination side. Thus far, the program only goes as far as to produce a CSV file that holds all of the required information on each of the collected events. However, the final stage of disseminating this information through the shared Google Sheet has not been approached. This could be undertaken in a manner similar to that used to write the FileWriteWorker class, making use of the `gspread` package and the Google Drive API.

Parallel Processing

When considering the time currently required to run all three activities sequentially, the majority of time is taken on the stages that involve requesting web-pages. This could be reduced by applying a parallel processing approach to these stages, allowing multiple pages to be requested at any one time. This will be of particular benefit in the case when the Workers are applied to large cities such as London where a greater number of events are taking place with a larger variety of musical artists. However, such improvements would be limited by the rate of response of the servers holding the required information.

Parallel processing would also allow for the stages undertaken by the FileWriteWorker and DataAnalysisWorker to run as soon as data had been collected by the SongKickWorker.

Handling of User Input

At present, when the program looks up artists on Wikipedia during the metadata supplementation stage, two types of exceptions are handled:

1. Page Errors — when Wikipedia does not appear to have any entries that match the query.

2. Disambiguation Errors — when Wikipedia appears to have multiple entries that that are close (but not exact) matches to the query, and cannot decide which one to use.

In the latter case, if the exception is not caught, the `wikipedia` package can prompt the user to choose between the various options as to which is the most appropriate given the query. This would result in a greater number of successful Wikipedia queries; however, this stage is currently the one that takes the greatest time, and allowing for user input at this stage would simply slow the process even further. It is therefore questionable as to whether the trade-off is worthwhile.

A Required Packages

In order to run the code to which this document is attached, the following packages are required:

- `collections`
- `csv`
- `datetime`
- `genres`
- `json`
- `math`
- `matplotlib`
- `os`
- `pandas`
- `requests`
- `time`
- `wikipedia`

The majority of these are provided upon installation of Python, however packages that the user is unlikely to already have include `wikipedia` and `genres`. The `wikipedia` package grants the ability to search Wikipedia pages, obtaining text content (amongst other forms). The `genres` package identifies keywords in strings that pertain to musical genres.

B Visualisations

Visualisations presented here are indicative of data collected and processed on 02/01/18. Data processed by the user is unlikely to produce the same visualisations.

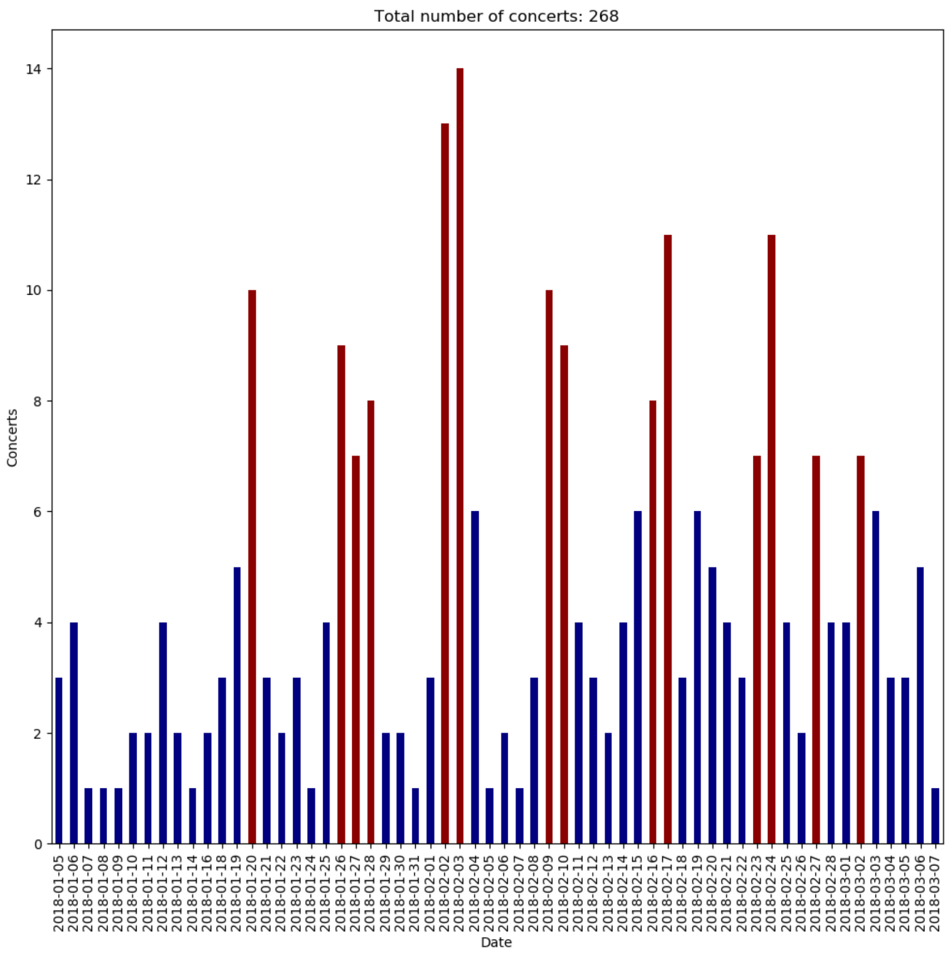

Figure 2: The variation of the number of concerts in Leeds with date. Red: dates with greater than six concerts; blue: dates with less than or equal to six concerts.

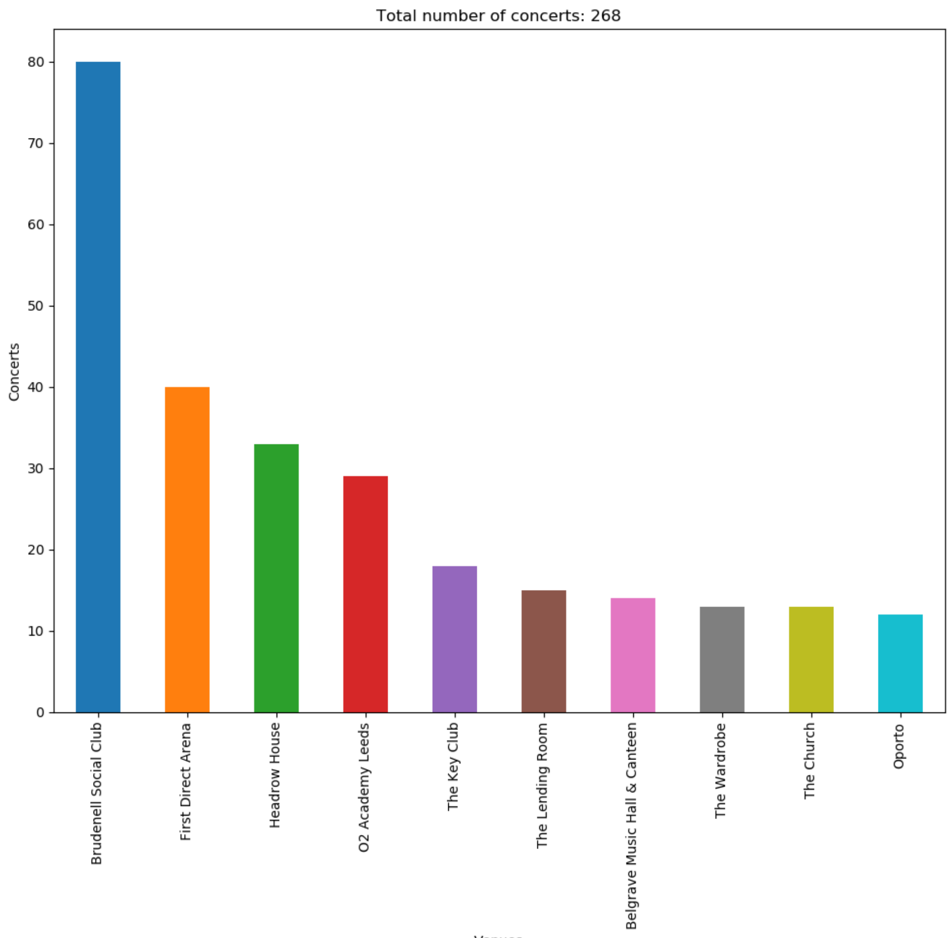

Figure 3: The variation of the number of concerts in Leeds across the 10 most active venues.

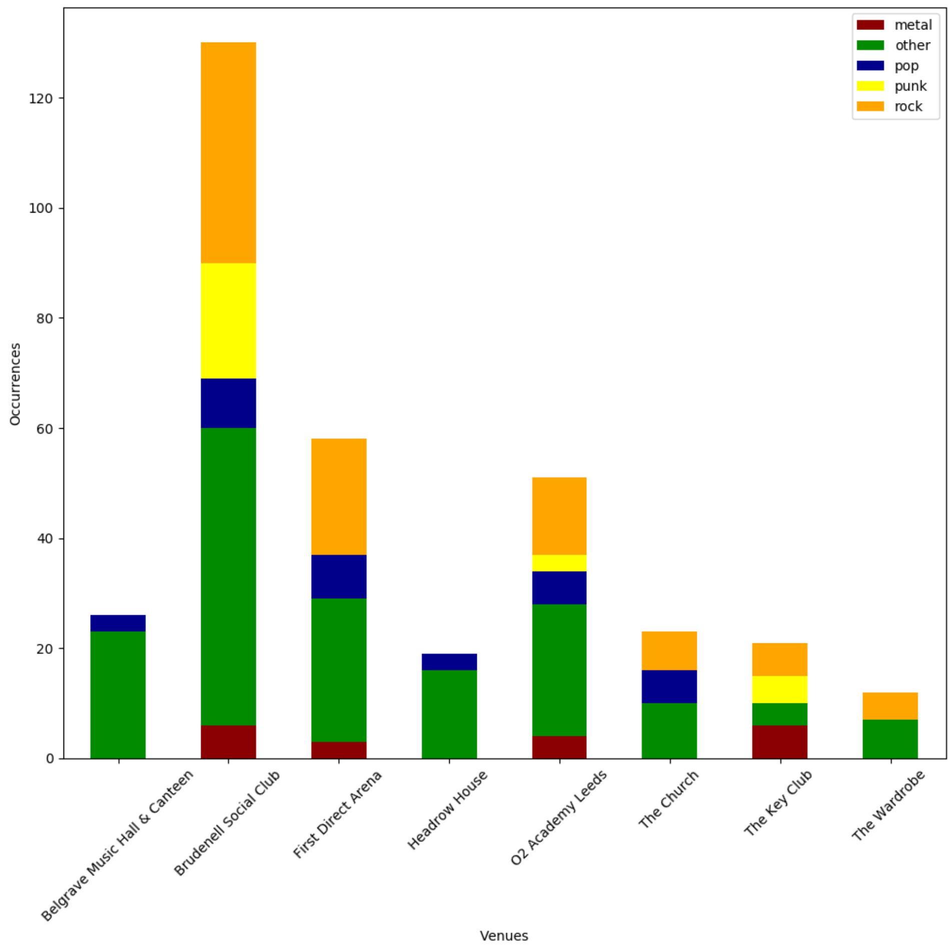

Figure 4: The variation of the genre of artists at each of the most popular concert venues in Leeds. Please note that genre totals at each venue may exceed artist totals as many artists fall under multiple genres.