# Report – Python Script 1 & 2

## Chosen datasets for Script 2

It was surprisingly difficult to find 3 linked datasets. We originally had hoped to link a homelessness report for one month to datasets describing regional employment and regional house prices (i.e. use the region as the linking method). Unfortunately, this was not possible as the data was not available on data.gov.ie. The most recent data on house prices we were able to find was 2015 and this only provided a breakdown of big cities and ‘other’ so could not be linked by region. This dataset was also collected 5 years before the homeless dataset was collected so it would not be possible to provide valuable insights. We found good employment data for Dublin only and once again we encountered the same problem of the datasets being collected years apart.

This led us to the decision to use 3 monthly (January, June & December) datasets examining the homelessness situation in Ireland in 2020. We were aiming to reveal developments and examine any differences in the different regions.

These datasets contained information describing the homeless situation in 9 regions of Ireland: Dublin, Mid-East, Midlands, Mid-West, North-East, North-West, South-East, South-West and West. The rest of the data contained counts of the total number, gender, age-bracket of homeless adults in each region. There was also counts of the number of times different types of temporary or emergency accommodation had been accessed and counts of the number of homeless families, dependents, single-parent families in each region.

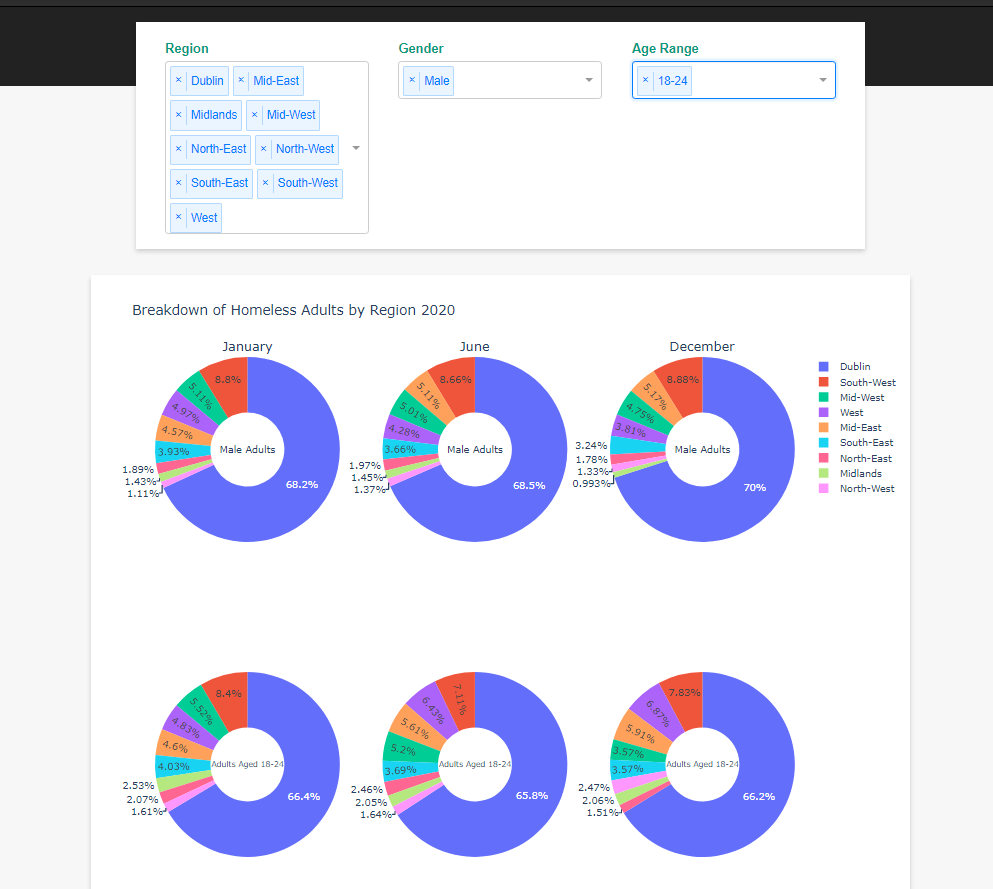
Mid-way through the assignment we realized these datasets might not be the best as there was not enough information to filter the data in some cases, for example it was not possible to tell how many males were in the 18-23 age bracket. However, it was possible to filter the data by region and month. We also got creative and created filters for gender and age, whereby if a gender was specified the count of the specified gender would be graphed instead of the total adults count and if the user wanted to only examine certain age brackets extra graphs would be created in figure 1.



Figure 1: Extra row of donut pie-charts generated as age bracket has been selected

We read in the files as csv files directly from the data.gov.ie site. We were able to read in the JSON returned from the data.gov.ie API, but we decided not to use this method as it was more complicated, and we believed it reduced the risk of errors during execution. Just to show we could read in the data from the API we still included the code that reads in the January dataset from the API and created a Panda’s data frame from the returned JSON, but this data frame is not used anywhere in our app.

## The methodology used for Script 1 and Script 2

Ensuring the programs ran smoothly was our number one priority. This meant that we had a lot of error handling. This error handling was primarily done in the form of try-except blocks. When an error was discovered we outputted a useful and easy to understand error message to the user. We imported the Python library sys. The exit function of this library was used when an error was caught. After the error message was outputted to the terminal we would use sys.exit() to terminate the running of the program. It should be noted that this will work perfectly if the whole script is run at once, but problems will arise and messy error messages will be displayed if the program is being run chunk by chunk (i.e. Selecting code and using Fn + F9). This should not be a problem for a user but will be a minor annoyance for someone trying to debug the program chunk by chunk (the error message is still outputted but theres just a lot of error traces outputted too).

Both scripts were broken down into logical operational chunks. These chunks were clearly described in comments and the start and end of chunks was also clearly defined using multiple ‘#’. For script2 these chunks include reading in the data, cleaning the data, creating the app etc.

Comments were frequently used which hopefully makes all the code understandable.

Git version control was also used throughout development of the two scripts to ensure that code was not lost and we could go back to previous versions if we made a bags of it. This was particularly important for script 2.

For script 2 a local CSS file was used to style the dashboard. This was done as it saved time as we could create blanket style classes and wrappers with ease. It also helped to ensure consistency in our styling. Should any changes need to e made to the styling it is quick and easy to do so with a local CSS file.

## Difficulties Encountered

### Script 2

The biggest trouble we had with this assignment was choosing linked datasets for script 2. We realized a bit too late that we had shot ourselves in the foot with our choice of datasets as there was not enough information to filter the data as we would have liked. We were also limited in what we could graph, and Dublin (unsurprisingly) dominated every graph.

We used pivot tables to create graphs 2, 3 & 4. It took a bit too much googling and reading through documentation to realize that these could be created using pivot tables. It took me a couple hours of debugging to figure out how to change an example I had seen in Plotly documentation to work with our dataset and do what we wanted it to.

Script 2 was quite time consuming compared to script 1. It took a long time to figure out how to get things to work and to make changes.

I spent a bit too long trying to deploy our app to the cloud platform Heroku. Eventually I gave up as the documentation was hard to follow and I realized it wasn’t that important.

After this I tried to create a regional heatmap of Ireland, but this stuff went way over my head and I quickly gave up as there was not much time left until submission.

### Script 1

I was a bit confused how to go about Script 1 at first, but it wasn’t that bad in the end. The most difficult thing about this script was that the user could read in any datasets and its simply not possible to test every dataset out there. Because of this we knew we would have to have good error handling and rely on verification (of the read in df) from the user.

The file explorer was relatively easy to set up and the error handling catches every exception we could think of. The hardest part about it was reading in the JSON and the JSONStat. We couldn’t find a one-size fits all solution for reading in the JSON files so it worked for some of the files we tested but other files will contain unparsed JSON in some columns. If I had more time, I’d try do some sort of check to see if we had unparsed columns and then parse them. As JSON files are sometimes saved as .txt files we checked if the txt files contained valid JSON and if they did, we proceeded and if they didn’t any errors were caught, and the program execution was terminated.

Since JSONStat files do not have a universal extension this added an extra level of difficulty as we had to have an option for (‘\*.\*’) all files in our file explorer GUI, but we were able to handle this by examining the end prefix of the file path of any read in files and using multiple try-except statements.

For a csv file a user is asked if they want to specify a delimiter, if they do not the default ‘,’ is used. This means the user may select the wrong delimiter, but this is later dealt with by displaying descriptive statistics and the head of the read in Pandas data frame is displayed to the user. At this stage the user is asked if they want to continue. If they do, they can then specify a filename and if they enter nothing or an invalid filename the data is saved as “data+<unixTimeStamp>” in their local directory. The filepath and filename are then outputted to the user.