SCC.022 Making Sense of Data

Studio Worksheet: Week 1

In this week's studio, we will cover aspects from the first week of the module and build on the skills learnt in "SCC021: Scripting in Python." We will look at some data cleaning approaches and implement them in Python using the Pandas Data Analysis library.

This lab session is split into two parts: (i) group discussions, (ii) practical implementation using Python. The preparation detailed below should be completed before the lab session.

Preparation

Please complete the following preparation items before the lab session.

Step 1: Install Python

We will be implementing and testing our code in Python. You can install Python from AppsAnywhere, or via the Python website if you have problems with AppsAnywhere.

Step 2: Create a directory

When developing programmes, we usually define a "home" directory (or folder) which will house all of our files for a particular project. For example, for this studio, it would be sensible to create a folder called "Wk1" inside another folder called "SCC022" on our H-Drive. This would have the address: "H:\SCC022\Wk1". From this point, we will call this our "home directory" and use this address in our examples. Please keep in mind that your home directory address might be different so, when you see this, you should substitute it for the home folder address.

Step 3: Download the week1studio.zip file

This can be downloaded from the Moodle module page. This contains the files needed for the rest of this worksheet. Save it to your home directory and extract the archive so the files are accessible (right click the file and select "Extract all..." or similar). Move the files to your home directory. This should include the files "requirements.txt", "coronavirus-cases.csv" and "NOTT_2015.csv"

Step 4: Install Pandas and the supporting libraries

We will use “pip” to install these:

1. Launch the command prompt:
    a. If you are using Windows: Open Windows Powershell from the start menu.
    b. If you are using Mac: Open Terminal.app
    c. If you are using Ubuntu: Open the Terminal
2. Go to the folder where the worksheet files are saved by typing “cd H:\SCC022\Wk1”
   NB: If there are spaces in your filename, you may need to use speech marks in your command. E.g. cd “H:\SCC 022\Wk1”
3. Install the required libraries by running “pip install -r requirements.txt” This will install the Pandas, Scipy and Matplotlib libraries.

Task 1: Discussion

In the "What is Data Science?" and "Data Sources" videos, we considered what kinds of data we might want to collect to and how we might do this to understand an underlying issue. We will consider several questions below. For each topic, discuss in your groups:

- What do you want to achieve with that research and how?
- What are the aims, objectives, and questions?
- What kind of data you would need to address those questions?
- What kinds of biases might be present in the data and how might you prevent them?

Topics:

1. The success of recipe box delivery systems
2. The promotion of equality and diversity at university
3. National lockdowns and supermarket shopping.

HINT: There are many different aspects of these these topics that can be considered for research. These could be related to finance, customer satisfaction, environmental impact, etc.

Task 2: Data Import and Plotting

In this task, we will take a real data file, import it using Python and perform some data visualisation. This will introduce you to a several functions in Pandas and things you might need to look out for in your own data cleaning with other files. These are broken into separate steps. You can view these as building blocks that you can reuse or experiment with modifying to try to do something similar. We will use the coronavirus-cases data file from the government website in this task.

Step 1: Run IDLE

We will be using the interactive window in IDLE to run our program step-by-step as we learn the steps. If you want to run it all automatically later, you can write it into a new file window in IDLE and save and run it like other Python programs. If you are doing so, remember to move all the import lines to the top of the script file.

Step 2: Import the libraries

First, we need to import the needed libraries into the program, so type these separate lines into IDLE's interactive window (the one with the >>> prompt):

```
>>> import pandas as pd
>>> import matplotlib as plt
>>> import datetime as dt
>>> import numpy as np
```

Tell the plotting library **matplotlib** to pop-up any graph windows and continue on with the program instead of waiting until the windows has closed to continue. This makes it easier to interactively experiment with.

```
>>> plt.interactive(True)
```

NB: you will not see any feedback from this command.

Step 3: Load the CSV file

Store the home directory address as a string variable and load the CSV file into a Pandas DataFrame using the read_csv function. We can then use that variable to access all the values in the file.

```
>>> hdir = "H:\\scc022\\Wk1\\"
>>> cvd = pd.read_csv(hdir + "coronavirus-cases.csv")
```

NB: If there are any backslash characters in the path then we need to type them twice to escape them so they are stored in the Python string properly - if you forget to do this you will get an error. Note also that we have used the "+" symbol to join together two strings.

The CSV file should load without error. This will be stored in a DataFrame called cvd. We can check this by typing the variable name into the command window and it will show us some of the first and last rows and columns. It is a good idea to do this after each command below so you can see the effects.

```
>>> cvd
```

The number of rows and columns that you see is limited by the Pandas default values. You can change this if needed by modifying the Pandas parameters as in the 2 lines below. After each command, try typing "cvd" to see the effect that each command has.

```
>>> pd.options.display.max_rows = 20      # number if it can
all be shown at once

>>> pd.options.display.min_rows = 20      # number to show if
it exceeds max_rows.
```

NB: in the above code we use the pd variable name since we are editing the Pandas parameters rather than the DataFrame cvd.

Step 4: Clean and tidy the data

Data Format: Converting dates. By default, Pandas imports everything in the file as text if it is not obviously a number. This means dates and times are stored as text. Pandas can do more intelligent things with these dates and times (e.g. put them in chronological order or calculate time differences) if they are stored as dates and times. We therefore want to convert these string values to the correct format:

```
>>> cvd['Specimen date'] = pd.to_datetime(cvd['Specimen
date'], dayfirst=True)
```

NB: What does the component “dayfirst=True” mean? Why do we use it?

Step 5: Inspect and visualise the data

Describe and inspect the data. Pandas has a nice, quick way of showing summary statistics for all columns in a DataFrame:

```
>>> cvd.describe()
```

It is often quite good for sanity checks and allows for an initial inspection of the data. In this case, we can quickly find that the maximum number of cases per day is 72500, which suggests that this file is not limited to only regional cases, as would appear from using “cvd.describe()”.

Plotting. To get more of a visual representation, we can plot one column against another:

```
>>> cvd.plot(x="Specimen date", y="Daily lab-confirmed
cases", figsize=(20, 10))
```

This data file contains data from several regions, meaning that for each day there are multiple values. Let us restrict to plotting the values for London. To do this, let us create a new dataframe containing only the data for London. That is, our new dataframe will contain only the rows where the value in the area name is “London”:

```
>>> cvd_london = cvd[cvd['Area name'] == 'London']
```

This creates a new dataframe called “cvd_london” containing only the data relating to London. Try typing “cvd_london” and “cvd_london.describe()” to inspect this new dataframe. Do you see that the original 153772 rows from cvd has now reduced to 359 in cvd_london?

Let us now plot the figure for London:

```
>>> cvd_london.plot(x="Specimen date", y="Daily lab-confirmed cases", figsize=(20, 10))
```

Take a look at the x-axis. The dates are in reverse chronological order. This is because the order from the dataframe has been retained. To plot in chronological order, we need to sort the data by the specimen date:

```
>>> cvd_london = cvd_london.sort_values(by='Specimen date')
```

Now, try plotting the above graph again to see the difference:

```
>>> cvd_london.plot(x="Specimen date", y="Daily lab-confirmed cases", figsize=(20, 10))
```

Further tasks

Now that you can import, filter, sort and plot the data, you can try a few more tasks. But if time is short, it would be better to move onto Task 3 and try these later:

- Try plotting the figures for a different town
- Try plotting the cumulative cases

Task 3: Data Import, Cleaning and Plotting

In this task, we will use a data file that requires some more complex cleaning to prepare it for data analysis. In this case, we will need to be more careful about (i) what data is being imported, (ii) considering time as well as dates, (iii) the number of columns in the data. This will introduce some additional functions in Pandas. When you are cleaning your own files you might only need to do some of them and not others, depending on what the data looks like. We will use the NOTT data file in this task. If you still have IDLE running from the previous task, you can skip steps 1-2 below; consider why you can skip them.

Step 1: Run IDLE

We will be using the interactive window in IDLE to run our program step-by-step as we learn the steps. If you want to run it all automatically later, you can write it into a new file window in IDLE and save and run it like other Python programs. If you are doing so, remember to move all the import lines to the top of the script file.

Step 2: Import the libraries

First, we need to import the needed libraries into the program, so type these separate lines into IDLE's interactive window (the one with the >>> prompt):

```
>>> import pandas as pd
>>> import matplotlib as plt
>>> import datetime as dt
>>> import numpy as np
```

Tell the plotting library **matplotlib** to pop-up any graph windows and continue on with the program instead of waiting until the windows has closed to continue. This makes it easier to interactively experiment with.

```
>>> plt.interactive(True)
```

NB: you will not see any feedback from this command.

Step 3: Load the CSV file

Store the home directory address as a string variable and load the CSV file into a Pandas DataFrame using the read_csv function. We can then use that variable to access all the values in the file.

```
>>> hdir = "H:\\scc022\\Wk1\\"
>>> nott = pd.read_csv(hdir + "NOTT_2015.csv")
```

NB: If there are any backslash characters in the path then we need to type them twice to escape them so they are stored in the Python string properly - if you forget to do this you will get an error. Note also that we have used the "+" symbol to join together two strings.

If we try that we will get an error that ends with something like

```
pandas.errors.ParserError: Error tokenizing data. C error:
Expected 1 fields in line 3, saw 2
```

This is because this csv file is not in the expected format – it has extra lines at the start that are not expected to be there. If you open the CSV file using Excel or a text editor, you can see that the first 4 lines are not a part of the table. We have two options to address this: (i) we could edit the file manually, or (ii) we can edit our code to factor this in. While the first option is the easier solution, we would need to remember to do it each time we use the file. If we have a large number of files to examine, this could take a significant amount of additional work. The better option is to follow the second solution and incorporate this into our code. Generally, all data cleaning should be done within your programme so it is recorded and automated. We therefore need to tell Pandas to skip these lines when loading

the file. So, we will modify the previous line to tell the function there is a 4 line header at the top of the file that we want to skip:

```
>>> nott = pd.read_csv(hdir + "NOTT_2015.csv", header=4)
```

The CSV file should now load without error. This will be stored in a DataFrame called nott. We can check this by typing the variable name into the command window and it will show us some of the first and last rows and columns. It is a good idea to do this after each command below so you can see the effects.

```
>>> nott
```

The number of rows and columns that you see is limited by the Pandas default values. You can change this if needed by modifying the Pandas parameters as in the 3 lines below. After each command, try typing “nott” to see the effect that each command has. After the third command, you may see that the output is hidden under a label when you try to inspect the DataFrame nott. To view the output, you just need to double-click on the label.

```
>>> pd.options.display.max_rows = 20      # number if it can
all be shown at once

>>> pd.options.display.min_rows = 20      # number to show if
it exceeds max_rows.

>>> pd.options.display.max_columns = 36
```

NB: in the above code we use the pd variable name since we are editing the Pandas parameters rather than the DataFrame nott. When showing a lot of lines, IDLE often hides the output under a label that you need to double-click to expand. You may now want to change the maximum number of columns to 13.

Step 4: Clean and tidy the data

Removing un-needed columns. Often, your data will contain columns that are not needed for your analysis. In this file, there are lots of columns with names starting with "status" or "unit" that we do not need. We can use the “drop” function to remove named columns from the DataFrame in two different ways.

Method 1: Remove specific columns.

With this method, we remove individual columns by giving its name. In the below example, we remove the first column called “unit”. The second argument “1” means that we are removing columns. If we wanted to remove rows, we would replace this with “0”.

```
>>> nott = nott.drop("unit",1)
```

Method 2: Remove multiple columns with a pattern.

The second method is to remove all columns with names matching a pattern, which is helpful for our CSV file. To do this, we use the “filter” function and the wildcard symbol “*” that will match any text. That is, "unit*" refers to any text string that begins with “unit” regardless of the rest of the string. This is a two-step process: we first filter the columns using our pattern, then we give those columns to the drop function to remove. In the below case, we remove all columns starting with “unit” and “status”.

```
>>> nott = nott.drop(nott.filter(regex="unit*"), 1)

>>> nott = nott.drop(nott.filter(regex="status*"), 1)
```

If you display the DataFrame after each command, you will see the number of columns reduce each time. you should be left with only 13 columns after both.

Renaming columns. Often, the column names in data files are not very convenient to work with when programming. For example, in this data file, some column names are long, or

have spaces, or contain HTML fragments. It is therefore often beneficial to rename these columns. We do this by specifying the new names for each column:

```
>>> nott.columns = ['date', 'time', 'pm10', 'no', 'no2',
'n_as_no2', 'nv_pm10', 'nv_pm2', 'oz', 'pm2', 'so2',
'v_pm10', 'v_pm2']
```

Data Format: Converting dates and times. By default, Pandas imports everything in the file as text if it is not obviously a number. This means dates and times are stored as text. Pandas can do more intelligent things with these dates and times (e.g. put them in chronological order or calculate time differences) if they are stored as dates and times. We therefore want to convert these string values to the correct format. We first convert the contents of the "date" column:

```
>>> nott['date'] = pd.to_datetime(nott['date'],
dayfirst=True)
```

We can also convert the time:

```
>>> nott['time'] = pd.to_datetime(nott['time'])
```

If your CSV file contains times from 00:00 to 23:59, this is all you need to do. But for this example, you will get an error since our data file uses "24:00" to mean midnight instead of "00:00" so we must accommodate this. In this case, we need to convert all the "24:00" to "00:00" but we also need to add 1 day to the date of those lines since 24:00 represents midnight at the end of a day but 00:00 represents midnight at the start of a day. Since we now need to change two columns, this is more difficult. We must also be careful when we change dates since adjusting them by +1 day might also change the month and the year. First, we will add 1 day to the appropriate dates:

```
>>> nott.loc[nott['time']=="24:00", "date"] = nott['date'] +
dt.timedelta(days=1)
```

This is a busy line of code. On the left-hand side of the equals sign, we have a common Pandas pattern. The loc function is used to consider certain rows, in this case the rows where the "time" column is "24:00", and return a different column, in this case the "date" column. So, the left-hand side gives us the date values of all the rows which have a time of 24:00. The right-hand side uses the "date" column again to add an extra day using the datetime library (dt). Note that only the rows retrieved by the left hand side are modified.

We then update the times for each of those rows:

```
>>> nott.loc[nott['time']=="24:00", 'time'] =
pd.to_datetime("00:00", format="%H:%M").time()
```

Similarly, we only retrieve the rows with a time of “24:00”, but now we return the "time" column. On the right-hand side, we set each of these entries to midnight. The function to_datetime combines both the date and the time normally, so we ask for only the time portion using .time() at the end.

These are the most complicated lines in the cleaning process for this file because it had a non-standard format. It is often the case when working with real data that we have to work out how to do some conversions. If you search Google for help with such conversions, it is useful to include the name of the library that you are working with, e.g. Pandas. It is also often useful to split your code into multiple programmes to try to simplify the task but in this studio, we will stay with the command window.

Step 5: Inspect and visualise the data

Describe and inspect the data. Pandas has a nice, quick way of showing summary statistics for all columns in a DataFrame:

```
>>> nott.describe()
```

It is often quite good for sanity checks. For example, we can see that a couple of columns have minimum values less than zero. Given that these are volumes or proportions, that doesn't seem physically possible and so we should explore those columns further to confirm if they are errors and to deal with them appropriately (delete row, clamp to zero, shift the range to match zero, etc).

Plotting. We can plot one column against another:

```
>>> nott.plot(x="date", y="pm2", figsize=(20, 10))
```

We can plot multiple columns at the same time by giving a list of column names instead of just one:

```
>>> nott.plot(x="date", y=["pm2", "oz"], figsize=(20, 10))
```

We can also use different kinds of chart:

```
>>> nott.plot(kind="scatter", x="pm2", y="oz", figsize=(10,
10))
```

The list of different kinds of plot can be found in the Pandas plot() documentation. Note that some of them might require multiple columns or specific types of data. The window that pops up with the plot also allows you to save an image of the graph and to zoom in and out to see more detail.

Further Work

Experiment with your own data files... At this point you would move on through the data science process to exploring and analysing your data, which will be covered next week. For this week, try looking for data sets that are relevant to your major subject, load them into Pandas, and clean them up, in order to build some experience with the kinds of pre-processing you commonly need to do on data files. Check the "Data Sources" slides for a list of websites with big selections of data sets which might fit your subject.