Practical 7. Working with Global Health Data

Melanie Stefan

IBI1, 2019/20

1 Learning objective

- Analyse a public health dataset in python

2 Introduction

In 2020, the world is facing a pandemic outbreak of COVID-19. You have probably seen reports in the news that used plots to show how the COVID situation is developing. In this practical, you will learn to make such plots yourselves. The data comes from a public repository of COVID data[1]. In order to work with this data, you will be introduced to a new Python object: the dataframe. You will also make use of your Unix skills, as well as your Python skills for working with variables, handling numbers, and plotting.

This is a rather difficult practical - please use the Discussion Forum on Learn to discuss your solutions, and also to ask questions. If you find it difficult, don't give up. If you find it easy, help your classmates.

3 Importing a dataset

- First things first, you will need to import a few python libraries that will help you in this assignment.

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

What do those libraries do? You probably have already encountered `matplotlib.pyplot` for plotting and `numpy` for useful mathematical functions. The other two may be new: `os` allows us to work with files and directories, and `pandas` is for working with dataframes. We will explain this in more detail below.

- Download the data file `full_data.csv` onto your computer. Make sure you know what directory you downloaded it to, including the full directory path. (Look back at Practical 2: Introduction to Unix if you need a reminder of how to do this.)

- In your python script, change the working directory to where your `full_data.csv` file is stored. You can do this using the `chdir` function provided by the `os` library. Here is what it may look like, but of course you will have to change it to the correct path!

```
os.chdir("/home/melanie/Work/IBI1/Practical7")
```

[1] Max Roser, Hannah Ritchie and Esteban Ortiz-Ospina (2020) - "Coronavirus Disease (COVID-19) Statistics and Research". Published online at OurWorldInData.org. Retrieved from: `https://ourworldindata.org/coronavirus-source-data` [Online Resource]

- The `os` library also provides two other useful functions to check that what you have done is correct:
  - `getcwd()` is equivalent to the `pwd` function in a Unix command line
  - `listdir()` is equivalent to the `ls` function in a Unix command line

  Try both. What do you see? What does this tell you?

- Now we are finally ready to import the actual dataset.

  Let's have a quick chat about the file format of the data set (`full_data.csv`) first. You can have a look at the file in some spreadsheet programme such as Excel, Numbers, or Libreoffice Calc. What you will see is that the data is neatly organised in six columns, starting with a column for "date". Each row is a particular observation. For instance, the second row shows the numbers of new cases, new deaths, total cases, and total deaths recorded in Afghanistan on 31 December 2019. (The very first row just contains the titles of all columns). Once you have had a look at your dataset, close the spreadsheet programme again, but make sure you haven't made any changes to it before you do. (If you have changed something, download the file from Learn once more and start fresh.

  What is a .csv file anyway? The ending (.csv) indicates that the file is stored as "**c**omma-**s**eperated **v**alues". This means that internally, the file is stored as a list of things with commas in between. You can see this if you open the file in a normal text editor (such as notepad, text edit, emacs, vim, or similar) - as you can see, it's just many rows of words and numbers, with commas between them. Software such as your spreadsheet app knows that whatever is between two commas belongs to a cell in the spreadsheet. It also knows that line breaks mean the start of a new row. A lot of software packages know how to read .csv files, and it is therefore a very useful way to store tables of data.

- In python, we use the `pandas` library to read the content of the .csv file into a dataframe object. We call our dataframe `covid_data`.

  ```
  covid_data = pd.read_csv("full_data.csv")
  ```

4 Working with dataframes

- What is a dataframe? It is a table where columns correspond to types of measurements taken and rows to individual data points. Let's have a look!

- You can see the beginning of the data frame using the head command. Try:

  ```
  covid_data.head(5)
  ```

  What does the number 5 do? Change it to find out!

  You will notice that similar to your spreadsheet programme, python has recognised that things between commas should be read as separate things. It has also recognised that the first row contains no data, but instead the names of the columns.

- Let's take a closer look. You have learnt in week 6 that variables in python come in types. What type are the data points in `covid_data`? What are the columns called? How many rows are there? You can answer all these questions using the `info()` command:

  ```
  covid_data.info()
  ```

- The last useful whole-dataframe command is `describe()`. For each numeric column, this shows the number of entries, mean, standard deviation and a number of quantiles. (If you don't remember what quantiles are, now would be a good time to read up on it and remind yourself!)

  Use the `describe()` function to answer the following questions:

- What was the mean of new cases (across all rows of the dataframe)? What is the median?
- What is the range of total deaths (minimum and maximum)?

- What if you want to see only specific values in your dataframe? This is where the fun begins! We will look at two functions, `iloc` and `loc` - one is to access values by row and column number, the other to access values by row number and column name.

  Let's start simple. The syntax is always the same: after `iloc` or `loc`, there are square brackets. In the square brackets there is a comma. Before the comma you have the rows you are looking at, after the comma the columns. For instance, this shows you what's in the first row, second column (as always in Python, counting begins at zero).

  ```
  covid_data.iloc[0,1]
  ```

  Can you verify from looking back at the "head" command that this is really true?

- What if I want to show several rows and columns? Play around with the following commands (try them, make up a hypothesis about how they work, change them, see what happens) to understand what they do:

  ```
  covid_data.iloc[2,0:5]
  covid_data.iloc[0:2,:]
  covid_data.iloc[0:10:2,0:5]
  ```

  How would you show all rows, and every third column between (and including) 0 and 15?

- There is another weird thing we can do with iloc. Instead of numbers, we can use a *Boolean* to access entries. Let's say, I wanted to see the first three rows, but only the first, second, and fourth column. I could do this using numbers:

  ```
  covid_data.iloc[0:3,[0,1,3]]
  ```

  But instead, I could also do the following thing, which may seem a bit strange at first:

  ```
  my_columns = [True, True, False, True, False, False]
  covid_data.iloc[0:3,my_columns]
  ```

  What happened here? You will notice that `my_columns` is a list of Booleans that is the same length as the number of columns in the dataframe. If I use it to read columns with `iloc`, Python will go through the list and show me the columns where the value is True. It's almost like asking: Do you want to see the first column? Do you want to see the second column? Do you want to see the third column? Etc.

  What happens if `my_columns` *is shorter than the number of columns of your data frame? What happens if it's longer?*

  This may seem an unusual way of doing things, but we will find out later that it can be really useful.

- Let's look at `loc` next. While `iloc` uses the row and column **i**ndex to **loc**ate items, `loc` uses column names.

  For instance, try

  ```
  covid_data.loc[2:4,"date"]
  ```

Why are we still using row numbers? Well, rows are different from columns. Columns have names, but rows don't. But we *can* actually look for rows that interest us without having to know the row numbers. This next bit is a bit difficult to get right, but stick with it, because once you master it, it is basically a superpower!

- Let's say I am interested in all total cases in Afghanistan. Using loc, it's fairly easy to get the column part right.

  ```
  covid_data.loc[blabla,"total_cases"]
  ```

  But what goes in the "blabla" bit?

  Of course you could look at your dataframe and count, and you would figure out that rows 0 to 81 are for Afghanistan.

  ```
  covid_data.loc[0:81,"total_cases"]
  ```

  This is relatively easy in this situation, because the dataframe is ordered alphabetically by country and Afghanistan is first. But you can easily see that this is not a good solution for big dataframes that may not be ordered by your variable of interest - you really want a way to do this automatically. So, what you want is something like

  ```
  covid_data.loc[every row where location is "Afghanistan","total_cases"]
  ```

  But how do you find every row where the location is "Afghanistan"?

  In fact, you already have all the tools you need to do this! Try on your own, or follow these steps:

  1. How do you read just the "location" column, but all the rows from `covid_data`?
  2. How can you create a Boolean that is True when the "location" is "Afghanistan", but false otherwise? (Recall from lecture 6.1 how to check whether something is equal to something else)
  3. How do you use that Boolean to find exactly the rows you need in your dataframe?
  4. Is the result the same as when earlier, when you did

     ```
     covid_data.loc[0:81,"total_cases"]
     ```

  You have now completed the most difficult part of the practical!

5 Examining the worldwide situation

- Now, you can use your new superpower of retrieving values from dataframes to answer a few interesting questions. For instance, what can we learn from looking at new cases worldwide?

  You will notice that one of the locations in the "location" column is "World". This has the aggregated data for the entire world. In the next bit, we are interested in only the parts of `covid_data`, where the location is "World", and in two columns "date" and "new_cases". When you read out specific rows and columns and intend to use them for several different things, it is sometimes useful to save them as separate objects. For instance, I made an object called `world_new_cases` to store only the data on new cases for the entire world. This is not necessary, but can make your code easier to read (and easier to fix if it doesn't work!)

- Use `numpy` to compute both the mean and the median for new cases around the world. Are they similar or different? What does that tell you?

- Plot the new cases around the World as a box plot. What does it look like? Does this fit with what you know about the mean and median? What do you think is going on?

- Let's plot the data over time. For this, I made another object called `world_dates`. This may look a little bit different for you, depending on whether you also made objects like that and what you called them. But you should be able to edit the following function as needed:

  ```
  plt.plot(world_dates, world_new_cases, 'b+')
  ```

  What does `'b+'` *do? Change it to something different, e.g.* `'r+'` *or* `'bo'` *and see what happens. Make a hypothesis of what other command might work here and test it!*

- Maybe you are also interested in new deaths worldwide? Plot new cases in one colour and new deaths in another colour. Refer back to the lecture and practical from week 6 for help with titles, axis labels etc.

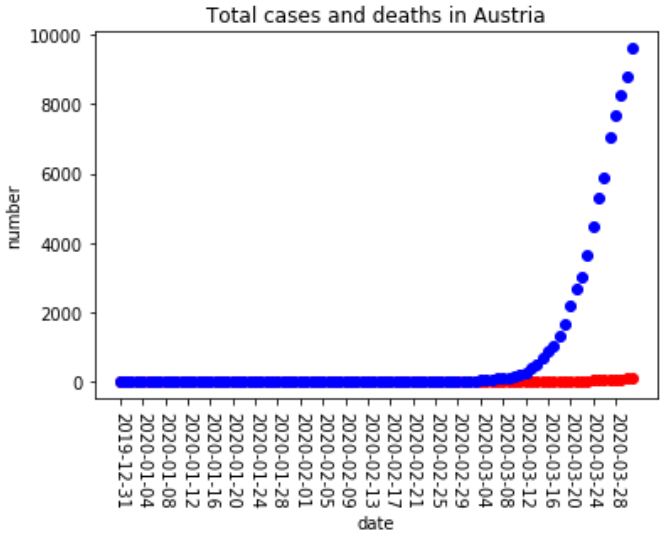
  I also use the following command to make my plot nicer:

  ```
  plt.xticks(world_dates.iloc[0:len(world_dates):4],rotation=-90)
  ```

  Why? What does it do? Change some of the numbers and see what happens if you are unsure.

6 Asking one other question

- Now you have all the tools you need to draw graphs like the ones you have seen in the news. This allows you to answer specific questions. For instance, I was interested in the total number of deaths and cases in my home country, Austria.



- Ask one question that can be answered using this dataset and a plot or some simple summary statistics. Make a text file called question.txt where you (briefly!) state the question. In the same .txt file, also provide the line number for where in your main python file the code to answer your question starts, and briefly discuss the result (E.g. is it what you expected, can you explain reasons for why the data might look the way it does?)

  You are completely free to pursue any question that interests you. But if you feel uninspired, pick one of the following suggestions:

  - How have new cases and total cases developed over time in Spain?
  - South Korea, Kenya, and Colombia are three countries in different parts of the world, but with similar populations. Plot the total number of COVID cases over time in all three.

- What proportion of cases have died (as of the latest information) in Germany? What proportion of cases have died in the UK?
- Are there places in the World where there have not yet been more than 10 total infections (as of 31 March)? If so, where are they?
- Plot a boxplot of total case numbers in different countries on 14 March 2020.
- ...

7 For your portfolio

The markers will look for and assess the following:

File covid.py

- The code for importing the .csv file works
- There is correct code for showing all rows, and every third column between (and including) 0 and 15
- You have successfully used a Boolean to show "total_cases" for all rows corresponding to Afghanistan.
- You have correctly computed the mean and median of new cases for the entire world.
- You have successfully created a boxplot of new cases worldwide.
- You have successfully plotted both new cases and new deaths worldwide.
- There is code to answer the question stated in file `question.txt`
- The code to answer the question runs without errors
- The code to answer the question does what it is meant to do
- All plots are clearly labelled

File question.txt

- A question is clearly stated
- A line number is provided that corresponds to where the question is addressed in the `covid.py` file
- The result is discussed.

You can add or edit things at any time before the portfolio deadline. We do not look at the commit date, we just want it all to be there!