# Analysing COVID-19 data

Helping our epidemiologist friend at WHO

This assignment asks you to create a number of general functions to process a JSON file. You will also need to create some tests and commit your work as you progress in a git repository.

The test file provided is not real and should work without difficulty, however real files will be provided that include different levels of difficulty to the exercise (e.g., missing data, different number of elements on some fields).

Your job consists of modifying/creating the functions so that the code works with the "simple" provided file, while also being general enough to process other files that share the same schema but vary in data quality. You'll also need to write some specific tests and demonstrate your ability to use git version control and GitHub.

The exercise will be semi-automatically marked, so it is very important that your solution adheres to the correct file and folder name convention and structure, as defined in the rubric below. An otherwise valid solution which doesn't work with our marking tool will **not** be given credit.

First, we set out the problem we are solving. Next, we specify in detail the target for your solution. Finally, to assist you in creating a good solution, we state the marks scheme we will use.

## 1 Our epidemiologist friend and his data problem

Jim is an epidemiologist at WHO. Lately he's been having a lot of data to analyse from different regions around the world. However, he never expected he would have to learn that much about computers. Thankfully, his good friend Carmen is amazing at writing scripts with Python and is afraid of no JSON file (however big or complex they are!). There's an additional difficulty though: WHO computers are very locked down! As the data they handle contains personal information, the computers have limitations on what software can be run on them and what data can be transferred to and from them. This means that Carmen will have to help Jim write some scripts in plain Python (i.e., we don't have access to NumPy or Pandas!), with matplotlib being the only exception allowed. "Not to worry", Carmen thinks, "if there's anywhere I'm stuck my friends at MPHY0021 can help me to solve these problems".

Jim has been able to generate a fake sample file that he can get out of the WHO computers and send to Carmen, so she can generate the functions that Jim can then load from a Jupyter notebook.

Inspecting the sample file, Carmen finds that it contains geographic and demographic information about a particular region in the world, information about some age binning, and daily evolution data for many measurements such as the number of people that have been hospitalised, tested or deceased. It also contains information about the weather of that day and different actions taken by the government in that region.

Based on Jim's needs, Carmen has created the skeleton of some functions and a notebook with how Jim is expected to use them.

# 2 Your mission!

You are required to modify the provided Python file (`process_covid.py`) so that Jim's notebook (`Jim.ipynb`) works as expected. You will also need to add some tests (within `test_process_covid.py`) using `pytest` and save it all in a git repository.

## 2.1 Reading JSON files

The first step we will need to solve is how to load the JSON files Jim is working with. Your goal is to create a function that loads the content of these files into a Python object. You are free to change the structure of the data after loading it if that makes any of the later easier. The function needs to be general enough to read files with different names and on different locations in the file system.

Make sure to check that the input file follows a valid schema (i.e., it's similar to the sample file), and throw an error if it doesn't.

## 2.2 Evolution of confirmed cases by age groups in terms of the total population

Jim needs to compare how the virus is spreading across the population for different age groups (i.e., the total number of confirmed cases). To show the impact on these population groups it's better to calculate these values as a percentage of each population group.

There are two problems though. The first one is that not all regions are providing the confirmed cases broken down into age groups. When this happens, we need to signal it properly so Jim understand what the problem is. The second difficulty is that the age ranges provided to indicate the population distribution and the number of cases confirmed are not always the same. To overcome this problem, we will rebin the data from one dataset into the other, but only when it's possible. For example, if we have dataset `A` and dataset `B` with the following ranges:

```
A = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-']
B = ['0-19', '20-39', '40-']
```

Then we are certain that we can rebin `A` with the same bins as `B` by summing together the corresponding ranges. However, when the ranges do not match and we can't rebin them, we should **throw** an error with a meaningful message. The following snippet illustrates a case when we won't be able to rebin the data:

```
A = ['0-14', '15-29', '30-44', '45-']
B = ['0-19', '20-39', '40-']
```

Jim requires the output of this to be a dictionary where the age groups are the keys, and each value is a list of tuples containing the date and the percentage of confirmed cases for that group (relative to the total population of that group).

```
result = {'0-19': [('2020-05-23', 3.3), ('2020-05-24', 3.4), ...]
          '20-39': [(...)]
          ...}
```

Or if you prefer, with type hinting:

```
def cases_per_population_by_age(input_data: "?") -> Dict[str, List[Tuple[str, float]]]:
```

## 2.3 New hospitalisations and new cases

Continuing with the confirmed cases, Jim wants to obtain the ratio of new admissions in hospitals compared to the new confirmed cases by day.

The required output in this case differs from the previous one. Jim needs a tuple with two lists, one with the date, and the other with the ratios.

```
result = (['2020-05-23', '2020-05-24', ... ],
          [0.3, 0.1, ... ])
```

or in the typing jargon

```
def hospital_vs_confirmed(input_data: "?") -> Tuple[List[str], List[float]]:
```

If you encounter missing data that doesn't allow you to calculate the ratio for a particular day, then that date should not be returned.

## 2.4 Preparing data to plot

Jim spends too much time rewriting code to make plots. To help with that, Carmen has decided to design a function that, depending on its arguments, generates the data to be plotted. As Jim's main area of work is the analysis of confirmed cases, we will start with that for now.

Sometimes, the number of confirmed cases comes broken down by age (as we've seen already) and on some occasions it also comes separated by sex. Additionally you may get the accumulated total number of cases, or the new ones detected on that day.

For this Carmen has left us with a very basic docstring on how the function should work:

```
def generate_data_plot_confirmed(input_data, sex, max_age, status):
    """
    At most one of sex or max_age allowed at a time.
    sex: only 'male' or 'female'
    max_age: sums all bins below this value, including the one it is in.
    status: 'new' or 'total' (default: 'total')
    """
```

and a half-baked wrapper function to plot sex and age groups:

```
def create_confirmed_plot(input_data, sex=False, max_ages=[], status=..., save=...):
    # FIXME check that only sex or age is specified
    fig = plt.figure()
    # FIXME change logic so this runs only when the sex plot is required
    for sex in ['male', 'female']:
        # FIXME need to change `changeme` so it uses generate_data_plot_confirmed
        plt.plot('date', 'value', changeme)
    # FIXME change logic so this runs only when the age plot is required
    for age in max_ages:
        # FIXME need to change `changeme` so it uses generate_data_plot_confirmed
        plt.plot('date', 'value', changeme)
    # TODO add title with "Confirmed cases in ..."
    # TODO Add x label to inform they are dates
    # TODO Add y label to inform they are number of cases
    # TODO Add legend
    # TODO Change show to save it into a '{region_name}_evolution_cases_{type}.png'
    #      where type may be sex or age
    plt.show()
```

The `generate_data_plot_confirmed` should produce all the data required so that the `plt.plot` within `create_confirmed_plot` understands it and the following is produced:

- When visualising the cases for males and females:
  - female should be represented by a purple line and male by a green line (in line with The Telegraph).
  - the legend should say "{status} {sex}" for each colour and line style.
- When visualising the cases by age:
  - the colour scheme has to follow: green for 25 or younger, orange for 50 or younger, purple for 75 or younger and pink for the rest.

  - the legend should say "{status} younger than {age}" where the age should be the upper limit of the age included.
- The linestyle for either case should be solid when visualising the total cases and dashed when it's the new ones.

The `create_confirmed_plot` function should let you choose whether to show the plot or save it to a file. This is controlled through the `save` argument, which should accept boolean values.

To successfully complete this mission, you'll need to read the matplotlib documentation to understand the parameters that are allowed in `plot`. Figure 1 shows an example of how the output plot should look when using `max_ages` with multiple values.

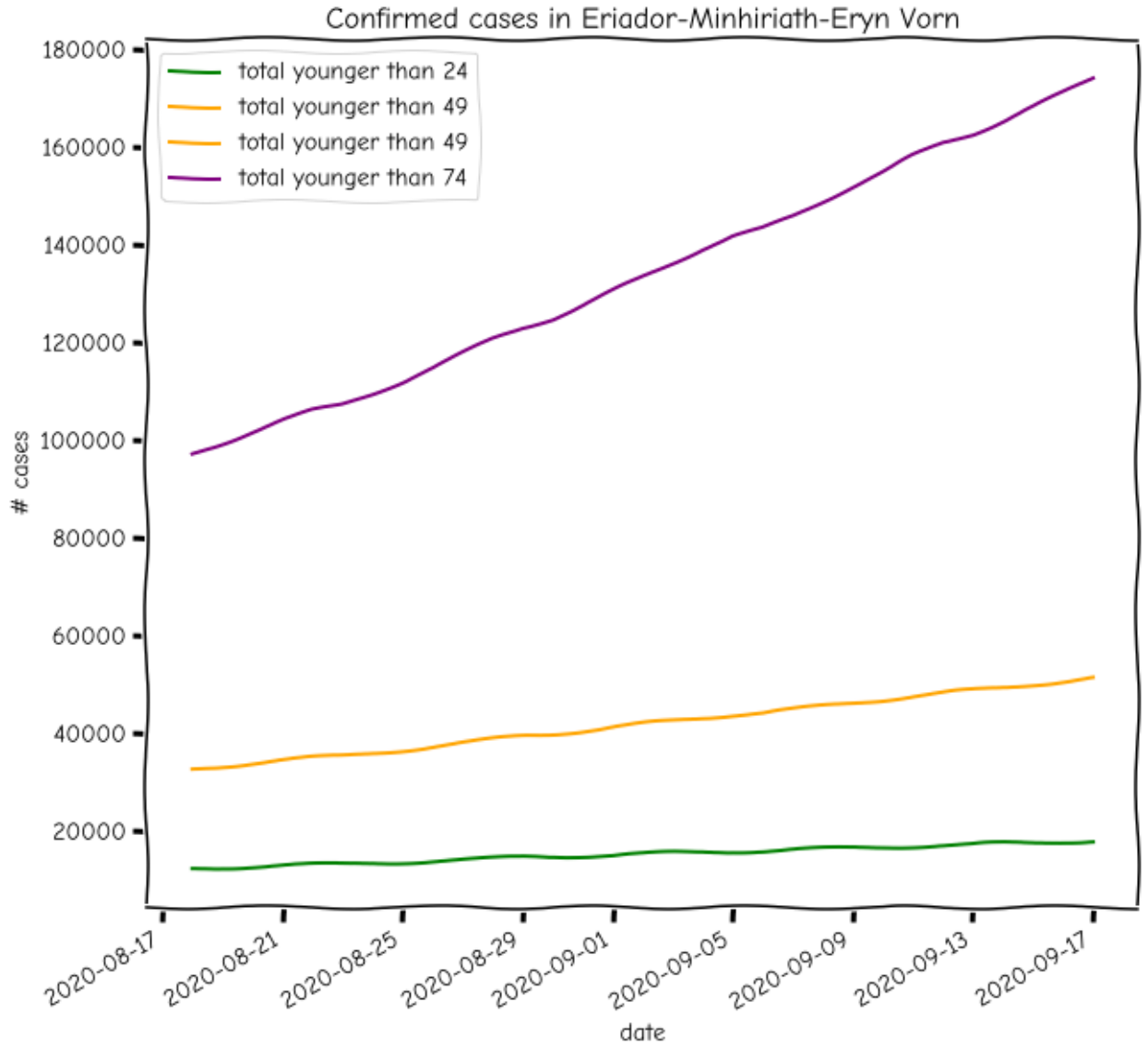


Figure 1: Example of the output of `create_confirmed_plot` for the case of multiple max ages, where two of the requested ages correspond to the same bin (e.g., `max_ages=[15, 25, 37, 55]` where the age bins are `['0-24', '25-49', '50-74', '75-']`).

## 2.5 The effect of weather

Jim is very interested to see whether the weather has any effect on the number of tests carried out every day. The question he wants to answer is: How many days that it rains (or rains more than the day before) coincide with a decrease in the number of tests?

To obtain an approximation to this question, Carmen suggests the following: 1) smooth the number of tests to remove any weekly variation 2) use a simple "derivative" (today's value minus yesterday's) to find the increases and decreases of the data 3) count the number of days when the derivative for the rain is positive and at the same time the derivative of the number of tests is negative.

The functions needed are then:

1. `compute_running_average` has to accept a list of numbers (assuming they are evenly distributed, e.g., one per day) and an odd window size. The output data also has to be a list of the same size as the input, however, where the average could not be computed, it should have the value `None`. For example:

```
>>> input_data = [0, 1, 5, 2, 2, 5]
>>> compute_running_average(input_data, 3)
[None, 2.0, 2.666, 3.0, 3.0, None]
```

   Additionally, when one of the values is `None`, it should calculate the average of the valid inputs. For example:

```
>>> input_data = [2, None, 4]
>>> compute_running_average(input_data, 3)
[None, 3.0, None]
```

2. `simple_derivative` also needs to work with a list of numbers of any length as an input, and returns a list with the same number of elements as the input. Affected edges and values that involve `None`s should be taken into consideration, returning `None` when the derivative can't be calculated. For example:

```
>>> input_data = [None, 1, 2, None, 4]
>>> simple_derivative(input_data)
[None, None, 1, None, None]
```

   The first `None` in the output is because the derivative can't be computed on the first position. The remaining `None`s are because the matching input values include `None`.

3. `count_high_rain_low_tests_days` should accept the input data as used in the previous sections and, using the two functions created above, return the ratio of

$$\frac{\text{number of days when rain has increased and a decrease in tests has been observed}}{\text{total number of days that rain has increased}}$$

Certainly, both Jim and Carmen are aware that this approach is not the most precise but they are curious to see the result.

## 2.6 Making sure your functions work as expected

Carmen will write most of the tests, but she would be happy if you could at least write the following unit tests:

- Any example shown above that could be converted into a test;
- `load_covid_data` throws a meaningful error if the structure of the file doesn't match with what's expected.
- `hospital_vs_confirmed` produces the expected output even when some values are missing;

- `generate_data_plot_confirm` throws a meaningful error when an input argument is not correct (e.g., `sex=4`)
- `compute_running_average` works as expected with different window sizes (with even and odd sizes)

Of course, you can create more tests if you'd like to, which can also help you check your work!

## 2.7 Keeping track of your moves

Carmen will keep working on the code on top of where you leave it, and, like the good developers we are, we should give her a repository with clear commit messages so she can inspect the changes you made, if needed.

You should create a Git repository for your code, and use `git` throughout your work. You should also create a repository on GitHub to hold your code (more details about this below). When making changes, focus on small pieces of work and feel free to create issues and branches as you see fit (though they are not a requirement for this assignment).

## 2.8 Directory structure

You must submit your exercise solution to Moodle as a single uploaded gzip format archive. (You must use only the tar.gz, not any other archiver, such as .zip or .rar. If we cannot extract the files from the submitted file with gzip, you will receive zero marks.)

To create a tar.gz file you need to run the following command on a bash terminal:

```
tar zcvf filename.tar.gz directoryName
```

The folder structure inside your tar.gz archive must have a single top-level folder, whose folder name is your student number, so that on running

```
tar zxvf filename.tar.gz
```

this folder appears. This top level folder must contain all the parts of your solution (the repository with the two Python files). You will lose marks if, on extracting, your archive creates other files or folders at the same level as this folder, as we will be extracting all the assignments in the same place on our computers when we mark them!

Inside your top level folder, you should have a `repository` directory. Only the `repository` directory has to be a git repository. Within the `repository` directory, the files `process_covid.py`, `test_process_covid.py` and `Jim.ipynb` have to be there.

Thus, if you run Jim's notebook from within the repository directory, this line will work:

```
from process_covid import load_covid_data
```

If you cannot do this, you will receive zero marks.

You should `git init` inside your `repository` folder, as soon as you create it, and git commit your work regularly as the exercise progresses. Due to our automated marking tool, only work that has a valid git repository, and follows the folder and file structure described above, will receive credit.

Due to the need to avoid plagiarism, do not use a public github repository for your work - instead, use the repository that you'll get access to by accepting this invitation: https://classroom.github.com/a/XZT6YbDk which, after you choose your UCL student number and accept the permissions, will create a repository named `MPH0021-2020-covid-data-<gh_username>`.

In summary, your directory stucture as extracted from the `studentNumber.tar.gz` file should look like this:

```
studentNumber/
└── repository/
```

```
├── .git/
├── process_covid.py
├── test_process_covid.py
└── Jim.ipynb
```

Remember to not include artefacts files (like for example: `.pyc`, `.DS_Store`) in your submission; their presence will remove points to your final mark. There's a way to make `git` ignore these files (you should include that in your submission too).

# 3 Mark Scheme

Note that because of our automated marking tool, a solution which does not match the standard solution structure defined above, with file and folder names exactly as stated, may not receive marks, even if the solution is otherwise good. "Follow on marks" are not guaranteed in this case.

You can add more functions or files (e.g., fixtures) if you consider it's appropriate. However, you should not change the name of the provided files and functions.

- **Version control with git (15%)**
  - Senssible commit sizes (5 marks)
  - Appropriate commit messages (5 marks)
  - Artefacts not included in any commit (5 marks)
- **Code structure (50%)**
  - Load the data (4)
    - ⋆ Using the right library to read it. (1 mark)
    - ⋆ Checks the file follows the right schema. (2 marks)
    - ⋆ Returns a Python object to be used in the other functions. (1 mark)
  - Evolution of cases by age and population distribution (8)
    - ⋆ The return object is as required. (1 mark)
    - ⋆ It works when the age bins are equal. (2 marks)
    - ⋆ It works when the age bins are different but can be grouped. (3 marks)
    - ⋆ It throws a meaningful error when the bins are incompatible. (2 marks)
  - New hospitalisations vs new cases (4)
    - ⋆ The return object is as required. (1 mark)
    - ⋆ It works as expected when all data is available in the given range. (1 mark)
    - ⋆ It doesn't generate output values for the dates on which new hospitalisations or new confirmed cases are missing. (2 marks)
  - Generate plot data (10)
    - ⋆ It checks the validity of the combination of the arguments used. (1 mark)
    - ⋆ It returns the right Python object for being used within `create_confirmed_plot`. (4 marks)
    - ⋆ It produces the right colours (by gender and age). (2 marks)
    - ⋆ It produces the correct line style depending of the status requested. (2 marks)
    - ⋆ And it creates the required legend labels. (1 mark)
  - Plot wrapper (8)
    - ⋆ Checks the input arguments used and throws an error if the inputs are invalid. (1 mark)
    - ⋆ Changed the logic of the provided sample to produce what's requested. (1 mark)

    - ⋆ Modified the `changeme` labels with the appropriate call to `generate_data_plot_confirmed` so that the line works. (2 marks)
    - ⋆ Create title and axis labels as required. (2 marks)
    - ⋆ Add a legend to the plot. (1 mark)
    - ⋆ Change logic to allow to visualise the graph on the screen or save it in a file. (1 mark)
  - Running average (6)
    - ⋆ Checks the validity of the inputs. (1 mark)
    - ⋆ Returns a list as required with the correct results when there's no missing data. (2 marks)
    - ⋆ Returns a list as required with the correct results when the input contains missing data. (3 marks)
  - Simple Derivative (4)
    - ⋆ Returns the correct data with the expected type and size when there's no missing data. (2 mark)
    - ⋆ Produces `None`s when needed. (2 marks)
  - Rain and tests (6)
    - ⋆ Using the output from `load_covid_data`, extracts the right values for tests and rain. (1 marks)
    - ⋆ Calculates the running average of tests carried over with a window size of a week. (1 mark)
    - ⋆ Produces the derivatives needed. (2 marks)
    - ⋆ Returns the ratio as requested. (2 marks)

- **Tests (25%)**
  - Examples shown here are converted as tests. (6 marks)
  - One negative test that checks that `load_covid_data` throws a meaningful error when using a file with a different schema. (5 marks)
  - One test that checks that `hospital_vs_confirmed` produces the right output when there's missing data. (5 marks)
  - One negative test that checks that `generate_data_plot_confirm` throws a meaningful error when called with wrong inputs. (5 marks)
  - One or multiple (parameterised) tests of `compute_running_average` that check windows of different sizes. (4 marks)

- **Style (10%)**
  - Good variable, functions and classes names. (3 marks)
  - Clean and readable structure of the code and tests files. (7 marks)

# 4 COVID JSON summary format

## 4.1 Description

Each file used in this service should respect the following schema. If it does not, please discard it. The file name should be `{region_name}_{start_date}_{stop_date}.json` where the `start` and `stop` dates corresponds to the date of the extract.

## 4.2 Schema

```
{
  metadata: {
    time-range: {
      start_date: String,
      stop_date: String
    },
    age_binning: {
      hospitalizations: [String],
      population: [String]
    }
  },
  region: {
    name: String,
    key: String,
    latitude: Decimal,
    longitude: Decimal,
    elevation: Decimal,
    area: {
      total: Decimal,
      rural: Decimal,
      urban: Decimal
    },
    population: {
      total: Integer,
      male: Integer,
      female: Integer,
      age: [Integer],
      rural: Integer,
      urban: Integer
    },
    open_street_maps: Integer,
    noaa_station: Integer,
    noaa_distance: Decimal
  },
  evolution: {
    <date>: {
      hospitalizations: {
        hospitalized: {
          new: {
            all: Integer,
            male: Integer,
            female: Integer,
            age: [Integer]
          },
          total: { <idem new> },
          current: { <idem new> }
        },
        intensive_care: { <idem hospitalized> },
```

```
        ventilator: { <idem hospitalized> }
      },
      epidemiology: {
        confirmed: {
          new: { <idem hosipitalizations/hospitalized/new> },
          total: { <idem hosipitalizations/hospitalized/new> }
        },
        deceased: { <idem confirmed> },
        recovered: { <idem confirmed> },
        tested: { <idem confirmed>}
      },
      weather: {
        temperature: {
          average: Decimal,
          min: Decimal,
          max: Decimal
        },
        rainfall: Decimal,
        snowfall: Decimal,
        dew_point: Decimal,
        relative_humidity: Decimal
      },
      government_response: {
        <various_parameters>: Integer,
        stringency_index: Decimal
      }
    }
  }
}
```