Formative assessment week 7: Snakemake

You have **implicitly** created a pipeline to generate your data already. You expect some original data to be present, and then the scripts are named to be run in a particular order. The scripts expect some of the data as inputs, and creates some output files. Some of those output files are inputs for subsequent scripts. This is a pipeline.

You can use Snakemake to make this pipeline **explicit**. Your objective is to write a Snakefile. In doing so, the snakemake orchestration will serve to

- manage the execution of your pipeline for example
  - checking what needs to be run
  - avoiding re-running things that have already been run
  - making sure things are run in order
  - managing parallel analysis
  - determining if steps need to be re-run if inputs have changed
- log the execution of the processes
- improve reproducibility
- make it easier to share your work with others
- eventually make it easier to run your pipeline on different systems including HPC

By the end of this session you will have a `Snakefile` that can run all 6 of the data management scripts that you have generated in your codebase so far. Going forwards, as you add more scripts to your codebase, make sure that you include additional rules to your `Snakefile` reflect how those new scripts need to be integrated into the pipeline.

1. Getting started

You will need `snakemake` installed within your conda environment. Whenever you change your environment file you can update your conda environment using e.g `conda update --file ahds_formative_environment.yml`. To install snakemake there are a few changes that need to be made to your environment file

- Add the `bioconda` channel
- Add the `snakemake` package as a dependency
- Note that `snakemake` currently works up to python 3.12. You may have python 3.13 installed. So add `python=3.13` as one of your dependencies also.

2. Create a Snakefile

Create a file called `Snakefile` in the home directory of your project. Start by adding a rule that

- Expects the `original/BMX_D.csv` file as input
- Checks the data
- Outputs a log of the data checking script

Run the snakemake pipeline.

3. Try running the snakemake pipeline again

Did it run? Why or why not?

4. Try to get snakemake to run again

Run `touch data/original/BMX_D.csv` to update the timestamp of the file. Then run the snakemake pipeline again. Did it run? Why or why not?

5. Add the accelerometer data check rule

Add a data check rule for the accelerometer data and create another log file as the output.

a) Note that for this rule the script is processing several thousand input files, whereas our previous rule only had a single input file. It is good practice to start with a small number of files to test and develop your pipeline. Start with a single file (e.g. `data/original/accel/accel-31128.txt`) and see if you can get the rule working.

b) Next, try to introduce a wildcard `{pid}` into the rule to allow the rule to work with any of the accelerometer data files. Look back at the advanced snakemake examples of how the `expand` function is used.

c) Finally, try to get the snakemake rule to observe all the accelerometer data files. Hint: You can use this python code to get a list of the accelerometer data files at the start of the Snakefile before specifying the rules:

```
import glob
import re

# Get the list of pid values from the filenames in data/original/accel/
accel_files = glob.glob("data/original/accel/accel-*.txt")
ACC_PID = [int(re.search(r"accel-(\d+).txt", f).group(1)) for f in accel_files]
```

6. Add all other rules required to complete the pipeline

Include:

- `code/3-data-fix-accel.sh`
- `code/4-list-accel-ids.sh`
- `code/5-generate-sample.R`
- `code/6-demo-data-prep.R`