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=3.12` as one of your dependencies also.

Your environment file should look like this:

```yaml
name: ahds_formative
channels:
  - conda-forge
  - bioconda
  - defaults
dependencies:
  - python=3.12
  - r-base=4.4.1
  - r-tidyverse=2.0.0
  - r-janitor=2.2.0
  - snakemake
prefix: /opt/anaconda3/envs/ahds_formative
```

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.

Initial snakefile:

```
rule all:
    input:
        "logs/1-data-check-bm.log"

## Step 1: Checking the data files and making sure they are in a standard file format
# Check the body measures data:
rule check_bm_data:
    input:
        "data/original/BMX_D.csv"
    output:
        "logs/1-data-check-bm.log"
    shell:
        """
        cd code
        bash 1-data-check-bm.sh > ../logs/1-data-check-bm.log
        """
```

To run:

```
snakemake -c1
```

3. Try running the snakemake pipeline again

Did it run? Why or why not?

When `snakemake -c1` is called again it should not run this second time, stating that all rules have been satisfied.

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?

Now the output is more recent than the inputs so Snakemake runs things again.

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]
```

---

a) Simplified version with a single hard coded accelerometer file:

```
rule all:
    input:
        "logs/1-data-check-bm.log"

## Step 1: Checking the data files and making sure they are in a standard file format
# Check the body measures data:
rule check_bm_data:
    input:
        "data/original/BMX_D.csv"
    output:
        "logs/1-data-check-bm.log"
    shell:
        """
        cd code
        bash 1-data-check-bm.sh > ../logs/1-data-check-bm.log
        """

# Check the accelerometer data:
rule check_accel_data:
    input:
        "logs/1-data-check-bm.log",
        "data/original/accel/accel-31128.txt"
    output:
        "logs/2-data-check-accel.log"
    shell:
        """
        cd code
        bash 2-data-check-accel.sh > ../logs/2-data-check-accel.log
        """
```

b) Using a few input files. The `ACC_PID` variable is then used in the rules to generate the lists of files to monitor and create.

```
ACC_PID = [31128, 31129, 31131, 31132, 31133, 31134, 31137]

rule all:
    input:
        "logs/1-data-check-bm.log",
```

```
        "logs/2-data-check-accel.log"


## Step 1: Checking the data files and making sure they are in a standard file format
# Check the body measures data:
rule check_bm_data:
    input:
        "data/original/BMX_D.csv"
    output:
        "logs/1-data-check-bm.log"
    shell:
        """
        cd code
        bash 1-data-check-bm.sh > ../logs/1-data-check-bm.log
        """

# Check the accelerometer data:
rule check_accel_data:
    input:
        "logs/1-data-check-bm.log",
        expand("data/original/accel/accel-{pid}.txt", pid=ACC_PID)
    output:
        "logs/2-data-check-accel.log"
    shell:
        """
        cd code
        bash 2-data-check-accel.sh > ../logs/2-data-check-accel.log
        """
```

c) A more advanced solution would require a little bit of python code to create a list of all the input accelerometry files. For example this code would read a directory and find a list of all the PIDs.

```
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]
```

This improves on the previous version because it automatically generates the list and includes all the files.

6. Add all other rules required to complete the pipeline

Include:

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

```
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]

# To keep things a bit simpler, we can hard-code a small list of participant IDs instead of reading in
# This will mean that if any of the accelerometer data files change or some are added or removed then s
# ACC_PID = [31128, 31129, 31131, 31132, 31133, 31134, 31137]

rule all:
    input:
        "data/derived/body_measurements.csv",
        "data/derived/sample.csv",
        "logs/1-data-check-bm.log",
        "logs/2-data-check-accel.log",
        "logs/3-data-fix-accel.log",
        "logs/4-list-accel-ids.log",
        "logs/5-generate-sample.log",
        "logs/6-demo-data-prep.log"


## Step 1: Checking the data files and making sure they are in a standard file format
# Check the body measures data:
rule check_bm_data:
    input:
        "data/original/BMX_D.csv"
    output:
        "logs/1-data-check-bm.log"
    shell:
        """
        cd code
        bash 1-data-check-bm.sh > ../logs/1-data-check-bm.log
        """

# Check the accelerometer data:
rule check_accel_data:
    input:
        "logs/1-data-check-bm.log",
        expand("data/original/accel/accel-{pid}.txt", pid=ACC_PID)
    output:
        "logs/2-data-check-accel.log"
    shell:
        """
        cd code
        bash 2-data-check-accel.sh > ../logs/2-data-check-accel.log
        """

# Fix the accelerometer data:
rule fix_accel_data:
    input:
        "logs/2-data-check-accel.log",
        expand("data/original/accel/accel-{pid}.txt", pid=ACC_PID)
    output:
        expand("data/derived/accel/accel-{pid}.txt", pid=ACC_PID),
        "logs/3-data-fix-accel.log"
```

```
    shell:
        """
        cd code
        bash 3-data-fix-accel.sh > ../logs/3-data-fix-accel.log
        """

## Step 2: Generating a sample file

# Our sample file contains a binary variable that indicates whether a participant is in our sample vers
# A participant is included in our sample if they have accelerometer data and a BMI value.
# First we create a list of participant IDs for those with an accelerometer file:
rule make_pid_list:
    input:
        "logs/3-data-fix-accel.log",
        expand("data/derived/accel/accel-{pid}.txt", pid=ACC_PID)
    output:
        "data/derived/accel/pids-with-accel.txt",
        "logs/4-list-accel-ids.log"
    shell:
        """
        cd code
        bash 4-list-accel-ids.sh > ../logs/4-list-accel-ids.log
        """
# Then we derive a sample file:
rule make_sample:
    input:
        "logs/1-data-check-bm.log",
        "data/derived/accel/pids-with-accel.txt",
        "data/original/BMX_D.csv"
    output:
        "data/derived/sample.csv",
        "logs/5-generate-sample.log"
    shell:
        """
        cd code
        Rscript 5-generate-sample.R > ../logs/5-generate-sample.log
        """


# Preparing and merging the demographics data

# Preparing and merging demographics variables into the body measurements data.
# It also merges in the sample file prepared previously and saves the combined body_measurements.csv fi
# Variable names are standardised to lower snake case.
rule merge_data:
    input:
        "data/original/BMX_D.csv",
        "data/original/DEMO_D.XPT",
        "data/derived/sample.csv"
    output:
        "data/derived/body_measurements.csv",
        "logs/6-demo-data-prep.log"
    conda: "ahds_formative"
```

```
    shell:
        """
        cd code
        Rscript 6-demo-data-prep.R > ../logs/6-demo-data-prep.log
        """
```