Let's create an experiment pipeline

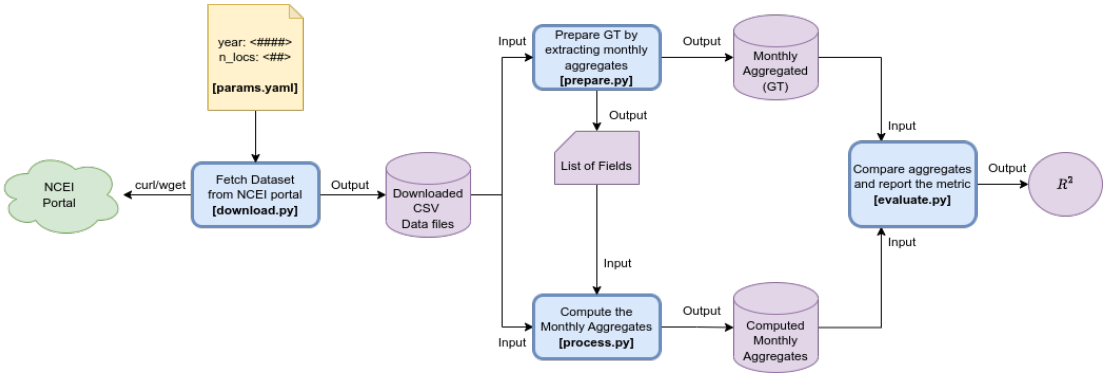
We want to setup a pipeline to verify a dataset's consistency. We will use the same climatalogical dataset that we used in the last assignment.

Our proposed pipeline will acquire the public domain climatological data from National Centers for Environmental Information. The archive contains data collected from over 13400 stations from YYYY=1901 to YYYY=2024. Each year in every station the data is collected multiple times a day on the readings of Altimeter, DewPointTemperature, DryBulbTemperature, Precipitation, PresentWeatherType, PressureChange, PressureTendency, RelativeHumidity, SkyConditions, SeaLevelPressure, StationPressure, Visibility, WetBulbTemperature, WindDirection, WindGustSpeed, WindSpeed, Sunrise, Sunset readings.

The data files (CSV formatted location specific files) also contain aggregate entries for monthly averages. We want to setup our pipeline to extract those monthly aggregates to compare them against the computed monthly averages from day wise data points. We shall use all the fields that have the monthly aggregate information. Note that some fields may not have them[1].

Once the monthly aggregates (ground truth in our experiment) and the estimated monthly aggregates (predicted outcome in our experiment) are ready, we shall compute the R^2 score as our measurement of consistency. If the R^2 >= 0.9, we call them as consistent (C).

The Pipeline



The above pipeline captures the entire task that you are expected to complete. The blue blocks (aka stages) are scripts that you should develop. The yellow block is the parameter file [params.yaml] that you should set ahead of running the pipeline. The purple items are the outputs from different stages which are used as inputs to other stages.

The Process [50 points]

1. Refer to the "02-DVC4ML.pdf" slideshow from slide 21 onwards to recall our discussion on setting pipelines using DVC.
2. Install Git for source control and DVC for source control and pipeline management.
3. Create a blank project in GitHub and check out to a folder. This folder will have the params, source, data and other outputs. You are expected to checkin all the relevant files to source control.
4. Initiate DVC also from the same folder. Now DVC and Git are linked.

1. Use appropriate strategies to handle the missing cases.

5. Setup the pipeline using “dvc stage add --run -v -f” command to add a stage to the pipeline. Every time you add a stage, “dvc.yaml” is created/updated in your folder. Also, another file named “dvc.lock” is also created/updated, which should be tracked in git. [20]
6. Once all the stages are added, use “dvc dag” to visualize the DAG of the pipeline. [5]
7. Run the pipeline using “dvc repro” command. Everytime, you change the parameters, the pipeline will be run again. You may change the ‘n_locs’, while keeping the ‘year’ constant during multiple runs. Alternately, you may introduce a dummy variable, say ‘seed: ####’ to the params file and keep changing only the seed to run multiple rounds. Remember, when the dependencies are unaltered, the pipeline would skip running. [10]
8. Use “dvc exp show” to list the runs. [5]
9. Use “dvc params diff” to compare experiments. [5]
10. Ensure that all the versions of your experiments are correctly checked into DVC and Github. [5]

Important Pointers

1. Ensure you comment your code thoroughly.
2. Ensure you checkin source changes made to your source code on a daily basis to Github.
3. Ensure you checkin the experiment (data, source, config, params) to dvc & git.
4. Here is the allocation of points per task.
    20% for a clean coding style
    50% for the correctness of the implementation
    20% for readable comments
    10% for input arguments’ validation/boundary check
5. Feel free to contact the TAs or me (sudarsun@gmail.com) if you need clarifications.

Best Wishes.