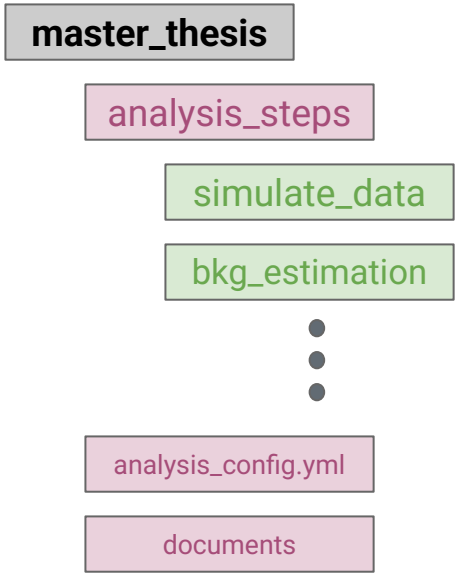
Suggested guidelines for your analysis setup/code

Disclaimer

These are only suggestions which are mainly inspired by the way i code myself and are meant to give you an idea how one could structure his code.

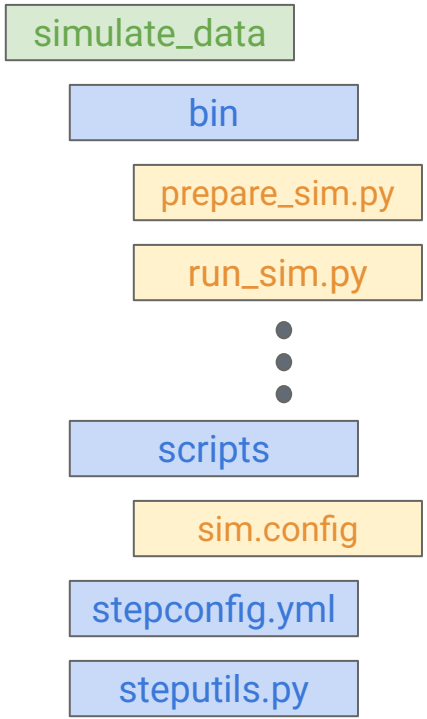
1) Folder Structure

- i would recommend one main analysis folder
  (e.g. your bachelor/master thesis which is of course gitted)
- think about the steps of your analysis and create a folder for each of these
  (git submodules if you want)



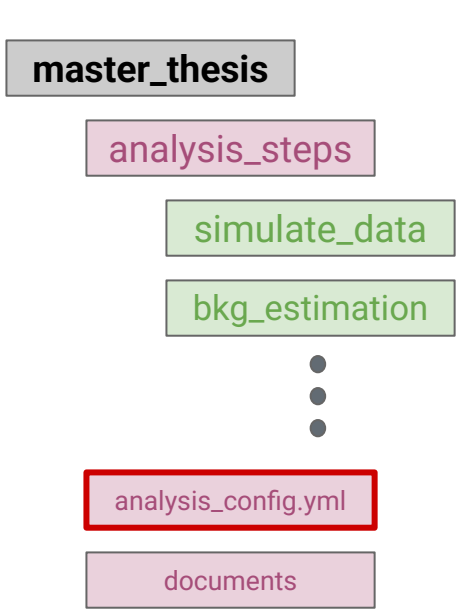
1.1) Step Folder Structure

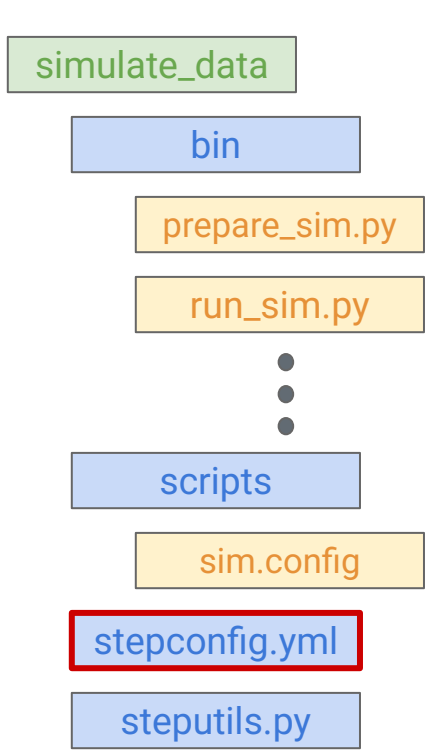
- feel free to change the naming of the bin/scripts folders. I also don't really like them, it is some old ATLAS standard i think
- idea: use **bin** for your **main scripts** that you have to execute in order to perform the step and **scripts** for any **additional file** needed for the main scripts



2) Use configs

- instead of hardcoding paths and parameters in each of your scripts, you can e.g. use yaml configs to set them centrally and then load them in each script that needs them
- i always have a ***stepconfig*** holding all the parameters/paths for the respective step and an ***analysis_config*** holding the analysis-wide parameters and paths, relevant for multiple steps





analysis_config.yml M

```yaml
# general
name: tqy differential
history_file: ~/Desktop/tqy_analysis/logs/history.txt

# analysis specific values
run_name: Run 2
center_of_mass_energy: 13
integrated_luminosity: 140
luminosity:
  mc20a: 36646.74
  mc20d: 44630.6
  mc20e: 58791.6

# plots
colors:
  tqy_prod:    [234, 153, 153]
  tqy_dec:     [100, 100, 100]
  tty_prod:    [164, 194, 244]
  tty_dec:     [162, 196, 201]
  Zyjets:      [160, 190, 160]
  Wyjets:      [182, 215, 168]
  prompt:      [180, 167, 214]
  hy_fakes:    [200, 200, 200]
  ey_fakes:    [229, 214, 192]
  Zjets:       [150, 200, 222]
  bernstein:   [214, 39, 40]
  Wjets:       [222, 90, 106]
  tt:          [140, 140, 140]
labels:
  tqy_prod:    '$tq\gamma$ (prod)'
```

stepconfig.yml

analysis_steps > egamma_fakes > stepconfig.yml

```yaml
# general
name: egamma_fakes
description: |
  This step is used to calculate scale factors for e->gamma fakes in CRs
  which shall correct the difference between MC and data.

  Required steps: ntuple_production, fastframes_processing.
  Required tools: TRExFitter.
  Manual:
    1. Do initialisation steps and create necessary folders. (init_step)
    2. Create histograms for the Bernstein polynomials. (create_bernstein_hists)
    3. Estimate scale factors with TRExFitter (do_fits)
    4. Calculate systematic uncertainties and create final SFs (calc_sfs_with_unc)
    5. Create validation plots (create_validation_plots)
    6. Create plots for the scale factors (comparison to old SF and systematic com

# paths
analysis_config_path:       '~/Desktop/tqy_analysis/analysis_config.yml'
step_path:                  '~/Desktop/tqy_analysis/analysis_steps/egamma_fakes/'
fit_result_path:            '~/Desktop/tqy_analysis/analysis_steps/build/egamma_fa
fit_summary_path:           '~/Desktop/tqy_analysis/analysis_steps/build/egamma_fa
trex_config_path:           '~/Desktop/tqy_analysis/analysis_steps/build/egamma_fa
replacement_path:           '~/Desktop/tqy_analysis/analysis_steps/build/egamma_fa
sfs_with_unc_path:          '~/Desktop/tqy_analysis/analysis_steps/build/egamma_fa
validation_path:            '~/Desktop/tqy_analysis/analysis_steps/build/egamma_fa
sf_plots_path:              '~/Desktop/tqy_analysis/analysis_steps/build/egamma_fa

# binnings
inv_mass_binning:
  nbins: 40
  low: 71
  high: 111
gamma_eta_bins: [[0, 0.3], [0.3, 0.6], [0.6, 1.0], [1.0, 1.37], [1.52, 1.81], [1.8
gamma_convtype_bins: [[-0.5, 0.5], [0.5, 1.5], [1.5, 2.5], [2.5, 3.5], [3.5, 4.5],
```

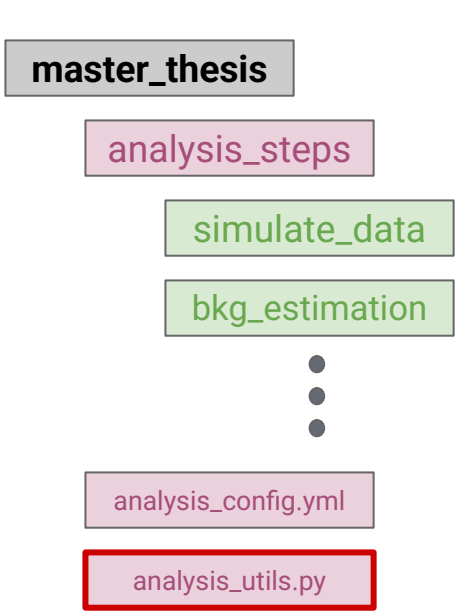
3) Export common functions

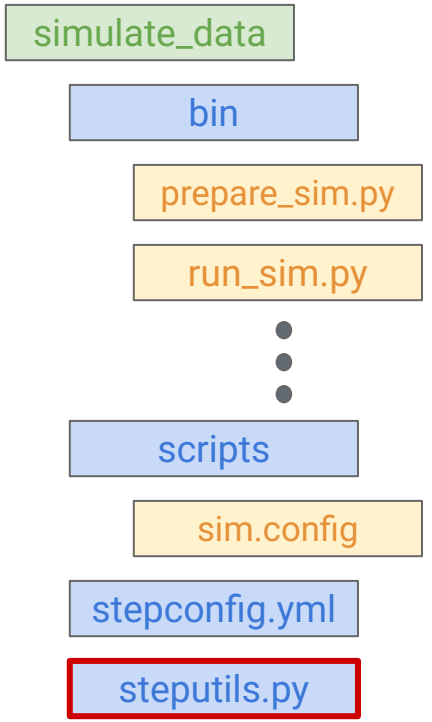
- same as with the configs: i use these python utils files to define functions commonly used in the analysis/step
  (e.g. a function to load the stepconfig)

steputils.py

analysis_steps > ntuple_production > steputils.py > ...

```python
import os
import yaml

from navigator.utils import AutoPath


def load_step_config():
    config_path = os.path.dirname(__file__) + '/stepconfig.yml'

    with open(config_path, 'r') as configfile:
        config = yaml.safe_load(configfile)

    return config
```





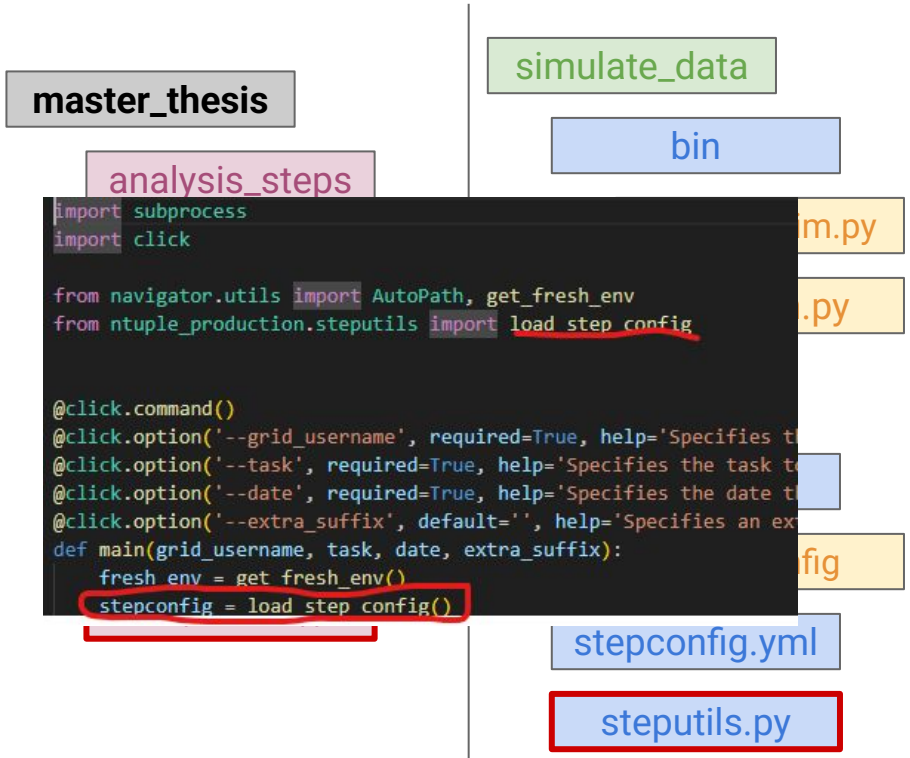
3) Export common functions

- same as with the configs: i use these python utils files to define functions commonly used in the analysis/step
  (e.g. a function to load the stepconfig)

steputils.py

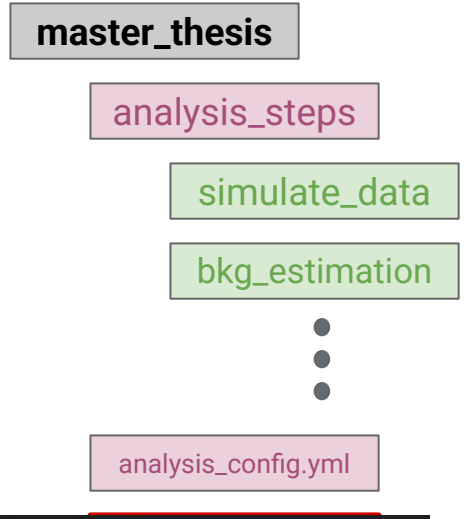
analysis_steps > ntuple_production > steputils.py > ...

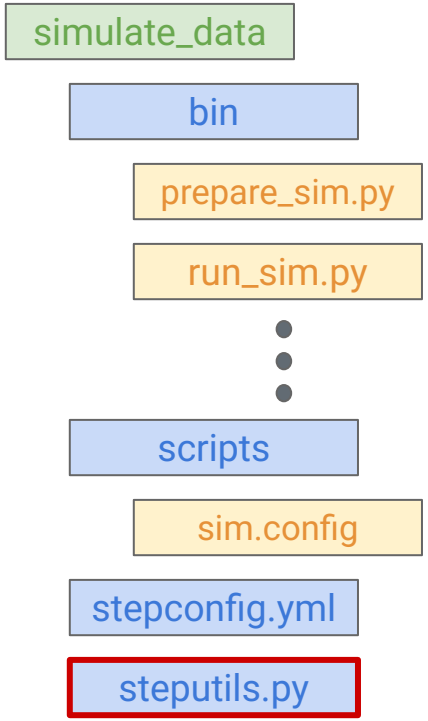
```python
import os
import yaml

from navigator.utils import AutoPath


def load_step_config():
    config_path = os.path.dirname(__file__) + '/stepconfig.yml'

    with open(config_path, 'r') as configfile:
        config = yaml.safe_load(configfile)

    return config
```



3) Export common functions

- same as with the configs: i use these python utils files to define functions commonly used in the analysis/step
  (e.g. a function to load the stepconfig)
- to be able to always import functions from these files you have to add them to your python path, e.g. by adding something like this to your **.bashrc:**

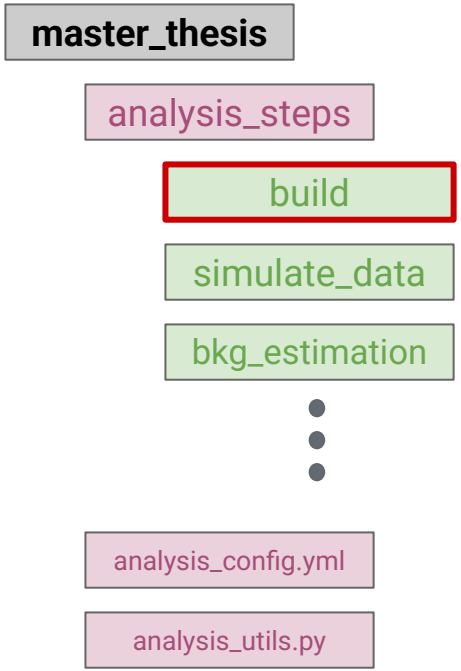





```
# To be able to import step specific python files from the analysis
export PYTHONPATH="/nfs/homes/lcremer/Desktop/tqy_analysis/analysis_steps:$PYTHONPATH"
export PYTHONPATH="/nfs/homes/lcremer/Desktop:$PYTHONPATH"
```

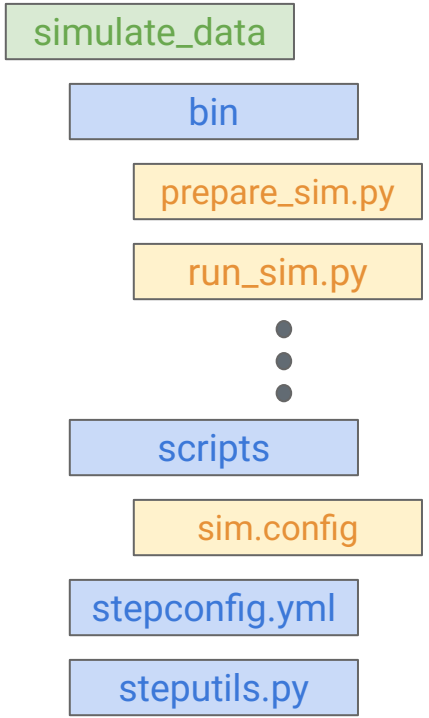
4) Use a build folder

- to keep the folders with your scripts clean and keep an overview which files are actually created by your scripts and which one are required to run the analysis, you can use a **build** folder
- idea: but everything create by your scripts inside the **build** folder
  (you should probably still implement sub folders in your build folder or it will get really messy, e.g. i have a sub folder for each analysis step inside the build folder)
- also makes it easy for the **gitignore**
- **VScode bonus hint:** you can configure the file explorer to hide certain files, so if you are annoyed by e.g. cache files polluting your file overview google or ask ChatGPT how to hide files in the explorer



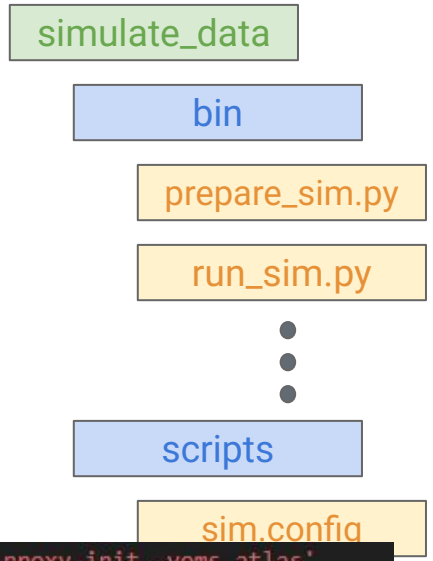
5) Python scripts

- for my main scripts i like to use always python
  (don't get scared by ATLAS, you can also "use python" to run e.g. ATLAS tools)



5) Python scripts

- for my main scripts i like to use always python
  (don't get scared by ATLAS, you can also "use python" to run e.g. ATLAS tools)
- you can use the **subprocess** library in python to run basically anything, by passing a series of terminal commands to be executed (e.g. including sourcing of ATLAS software)
  (you have to a bit careful to have a fresh environment for your subprocess, feel free to ask me if you have any questions on that)




```
setup_rucio = r'source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh -q && lsetup rucio && voms-proxy-init -voms atlas'
cmd = f'{setup_rucio} && python {script_path} {grid_username} {task} {date} {extra_suffix}'

subprocess.run(cmd, shell=True, env=fresh_env)
```

5) Python scripts 2.0

- i am using **click** to steer my python scripts by passing command line options

```
import click

from navigator.utils import AutoPath, get_fresh_env
from ntuple_production.steputils import load_step_config


@click.command()
@click.option('--grid_username', required=True, help='Specifies the grid user
@click.option('--task', required=True, help='Specifies the task to download f
@click.option('--date', required=True, help='Specifies the date the task was
@click.option('--extra_suffix', default='', help='Specifies an extra suffix w
def main(grid_username, task, date, extra_suffix):
    fresh_env = get_fresh_env()
    stepconfig = load_step_config()
    script_path = AutoPath(stepconfig['step_path']) + 'scripts/download_grid.
    setup_rucio = r'source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh -
```

5) Python scripts 2.0

- i am using **click** to steer my python scripts by passing command line options
- for my longer scripts i am using classes to avoid *variable passing madness* and ensure a clear structure

```
@click.command()
@click.option('--mode_train_weights', default='absolute', help
def main(mode_train_weights):
    analysis_config = load_analysis_config()
    step_config = load_step_config()

    preprocesser = Preprocesser(analysis_config, step_config)
    preprocesser.load_data()
    preprocesser.calc_train_weights(mode_train_weights)
    preprocesser.train_val_test_split()
    preprocesser.save_dataset_stats()
    preprocesser.scale_features()
    preprocesser.save_datasets()
```

```
class Preprocesser:
    def __init__(self, analysis_config, step_config):
        self.path_mini_ntuples = AutoPath(analysis_conf
        self.nn_variables = step_config['nn_variables']
        self.weight_name = step_config['mini_ntuples_we
        self.samples = step_config['samples']
        self.datasets_stats_path = AutoPath(step_config
        self.scaler_path = AutoPath(step_config['scaler
        self.preprocessed_datasets_path = AutoPath(step

    def load_data(self):
        samples_df = []
        for sample_name, sample_config in self.samples.
            print(cyan(f'Loading sample: {sample_name}'
            for subset in sample_config['subsets']:
                print(f'\tLoading subset: {subset}')
                files = self.path_mini_ntuples.rglob(f'
                for file in list(files):
```

6) Classics

- in general try to code in a way that someone else can also understand what is happening in your code
- use comments to explain what you are doing
- use meaningful variable and file names
- try to be consistent in how you name things and also in your code structure