Setting up the Coding Environment

Charles Martineau

2024-09-09

Table of contents

1 Poetry

We will work with poetry to manage the dependencies of the project. Poetry is a tool for dependency management and packaging in Python. It allows you to declare the libraries your project depends on and it will manage (install/update) them for you.

To install Poetry with brew for **Mac** users, run the following command in the Terminal app:

```
brew install poetry
```

For **Window** users, you can install Poetry by running the following command in the command prompt:

```
pip install poetry
```

By default, Poetry installs Python for each project in the `~/Library/Caches/pypoetry/virtualenvs/` directory. I prefer to have it in the project directory. That way if, I delete the directory, then the environment is deleted as well, which prevents accumulating virtual environments for discarded projects. To enable this, run the following command:

```
poetry config virtualenvs.in-project true
```

To create a poetry environment, first clone the repository and navigate to the root directory of the project. Then run the following command:

```
cd ~/codes/  # where codes is the directory where you store the cloned repositories
poetry init
```

Once the project is created, you can add the dependencies:

```
poetry add pandas numpy scipy matplotlib seaborn statsmodels scikit-learn linearmodels pya
```

You can always add more dependencies later by running the same command with the additional dependencies.

This step updates the `pyproject.toml` file and creates a `poetry.lock` file, which contains the exact version of each dependency. This file is used to make sure that all collaborators use the same version of each library. Note that because our dependencies are built on top of other libraries, Poetry will also install the dependencies of our dependencies.

To activate the environment in the terminal, run the following command:

```
poetry shell
```

Usually, after opening your GitHub repo through VS Code, the terminal will show the name of the Python environment in the prompt that was installed via Poetry. You won't need to run `poetry shell` in this case.

If the `poetry.lock` file is already present when you clone the repo, after install poetry, simply do

```
poetry install
```

2 .env file

We will use a `.env` file to store the environment variables. This file will be used to store the API keys and other sensitive information such as the root of the data directory. We will store the data on a server (and maybe some on Dropbox) and the directory root to the data is different for each user. The `.env` file should be in the root directory of the project. The

.env file should not be committed to the repository. To prevent this, we will add .env to the .gitignore file. The .env file should look like this:

```
API_KEY=your_api_key
DATA_DIR=/path/to/data
FIG_DIR=./results/figures/
TBL_DIR=./results/tables/
TMP_DIR=./tmp/
```

To read the environment variables, we will use the python-dotenv library. To install it, run the following command in terminal:

```
poetry add python-dotenv
```

To read the environment variables, the following code will be added to the main.py file such that the environment variables are read when the script is run:

```
from dotenv import load_dotenv
from pathlib import Path
import os

load_dotenv()

api_key = os.getenv("API_KEY")

datadir_path = os.getenv("DATADIR")
if not datadir_path:
    raise ValueError("DATADIR environment variable not set")

data_dir = Path(datadir_path)
download_dir = data_dir / "download_cache/"
open_dir = data_dir / "open/"
restricted_dir = data_dir / "restricted/"
clean_dir = data_dir / "clean/"
tmp_dir = Path(os.getenv("TMP_DIR", "tmp/"))

fig_dir = Path(os.getenv("FIG_DIR", "./results/figures/"))
tab_dir = Path(os.getenv("TBL_DIR", "./results/tables/"))
```

3 Hydra

We will use Hydra to manage the configuration of the project. Hydra is a framework for elegantly configuring complex applications. It is used to manage the configuration of the project. Hydra allows you to define a configuration file with default values and then override these values with command-line arguments. This is useful when you want to run the same script with different parameters. To install Hydra, run the following command in the terminal:

```
poetry add hydra-core
```

To use Hydra, you need to create a configuration file. The configuration file should be in the `conf` directory. The configuration file should be a YAML file. The configuration file should look like this:

```yaml
matplotlib:
  font:
    family: serif
    sans_serif:
      - Helvetica
    serif:
      - "Computer Modern Roman"

download_data:
  crsp: false
  compustat: false

process_raw_data:
  ibes_sue: false

database:
  # panel
  build_panel_db: false
  save_panel_db: false
  load_panel_db: true

tasks:
  # ---- Figures ----
  long_short_cumul_ret_fig: false
```

To read the configuration file, the following example code will be added to the `main.py` file such that the configuration file is read when the script is run:

```python
import hydra
from omegaconf import DictConfig, OmegaConf

@hydra.main(version_base=None, config_path="./conf", config_name="config")
def my_app(cfg: DictConfig):

    configure_pyplot(
        font_family=cfg.matplotlib.font.family,
        font_serif=cfg.matplotlib.font.serif,
        font_sans_serif=cfg.matplotlib.font.sans_serif,
    )

    download_files(
        crsp=cfg.download_data.crsp,
        compustat=cfg.download_data.compustat,
    )

    # Process raw files
    process_raw_files(
        ibes_sue=cfg.process_raw_data.ibes_sue,
    )

    if cfg.database.build_panel_db:
        logging.info("Build panel daily database")
        panel_db = create_panel_dataset(
            open_dir=open_dir,
            restricted_dir=restricted_dir,
            start_date="2012-01-01",
            end_date="2022-12-31",
        )
    if cfg.database.save_panel_db:
        logging.info("Saving panel daily database")
        panel_db.to_parquet(clean_dir / "panel_db.parquet")
```

4 Code directory

The code directory in Git should be structured as follows (with .py examples):

```
main_code/
    conf/
        config.yaml
    database/
        __init__.py
        download_data.py
        process_raw_data.py
        build_panel_db.py
    figure_codes/
        __init__.py
    table_codes/
        __init__.py
        regression_table_1.py
        regression_table_2.py
    utils/
        __init__.py
        common_functions.py
    tests/
        __init__.py
        test_download_data.py
        test_process_raw_data.py
        test_build_panel_db.py
```

5 Data directory

The data directory in our shared drive should be structured as follows:

```
data/
    download_cache/
    open/
    restricted/
    clean/
```

The `download_cache` directory is used to store the raw data files downloaded from the internet. The `open` directory is used to store the raw data files that are open to the public. The `restricted` directory is used to store the raw data files that are restricted. The `clean` directory is used to store the cleaned data files.

We will not save the raw and processed data on Git. We will use our shared dropbox folder.

6 Saving data

We should save our data using parquet format.

7 Naming convention

For default variables, like model parameters we use capital letters. E.g., N_LOOPS = 40 for the default number of loops. For others, we use small cap. E.g., n_loops = 40 for the number of loops in a specific case. We will store default variables in a separate file called `parameters.py` in the `utils` directory.

8 File formatting on saving

We will use the black formatter to format our code. To install black in VS Code add the extension `Black Formatter` to format the code automatically. To enable this, go to the settings and search for `format on save` and check the box.