Getting Started

Nadia Ahmed

Overview

In this lecture we will cover:

- Good repository structure
- Tooling Setup
- Interacting with AWS cloud storage to retrieve data

Organizing the Repository

Suggested Starter Directory Structure

- LICENSE
- README.md
- .gitignore (place data/ here)
- data
  - raw
  - processed
  - interim
  - external
- docs
- models
- notebooks
- environment/setup
- src
  - data
  - features
  - models
  - visualization
- tests
- references

```
├── LICENSE
├── Makefile           <- Makefile with commands like `make data` or `make
train`
├── README.md          <- The top-level README for developers using this
project.
├── data
│   ├── external       <- Data from third party sources.
│   ├── interim        <- Intermediate data that has been transformed.
│   ├── processed      <- The final, canonical data sets for modeling.
│   └── raw            <- The original, immutable data dump.
│
├── docs               <- A default Sphinx project; see sphinx-doc.org for
details
│
├── models             <- Trained and serialized models, model
predictions, or model summaries
│
├── notebooks          <- Jupyter notebooks. Naming convention is a number
(for ordering),
│                         the creator's initials, and a short `-`
delimited description, e.g.
│                         `1.0-jqp-initial-data-exploration`.
│
├── references         <- Data dictionaries, manuals, and all other
explanatory materials.
│
├── reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
│   └── figures        <- Generated graphics and figures to be used in
reporting
│
├── requirements.txt   <- The requirements file for reproducing the
analysis environment, e.g.
│                         generated with `pip freeze > requirements.txt`
│
├── setup.py           <- makes project pip installable (pip install -e .)
so src can be imported
├── src                <- Source code for use in this project.
│   ├── __init__.py    <- Makes src a Python module
│   │
│   ├── data           <- Scripts to download or generate data
│   │   └── make_dataset.py
│   │
│   ├── features       <- Scripts to turn raw data into features for
modeling
│   │   └── build_features.py
│   │
│   ├── models         <- Scripts to train models and then use trained
models to make
│   │   │                 predictions
│   │   ├── predict_model.py
│   │   └── train_model.py
│   │
│   └── visualization  <- Scripts to create exploratory and results
oriented visualizations
│       └── visualize.py
│
└── tox.ini            <- tox file with settings for running tox; see
tox.readthedocs.io
```

*Cookie Cutter Data Science

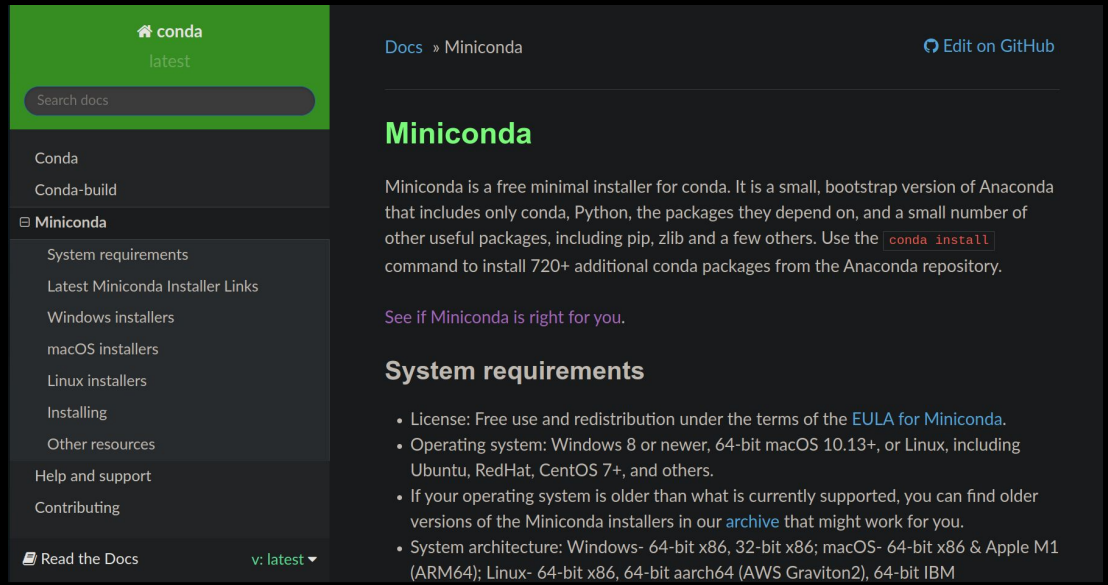
Environment: Tools

Setting up a Conda Environment: Advantages

- **Reproducibility**
  - team members use the same versions of Python and necessary packages for the project
- **Dependency Management**
  - Conda makes it easy to manage dependencies, such as different versions of Python, libraries, and packages.
  - Can create and update environment with the required dependencies for a specific project or configuration (e.g. cpu vs gpu)
  - yaml files are easy to share
- Isolation
  - Conda environments are isolated
  - Packages installed in one environment won't affect other environments.
- Flexibility
  - Conda allows for creating environments with specific versions of packages that are compatible with the project
  - If a package version changes, the environment can be updated or recreated to match the new version.
- **Easy setup**
  - Setting up a conda environment is relatively easy and can be done using a configuration file (.yaml) that lists the required packages

Environment: Tools

```
name: gpu_cs175
channels:
  - pytorch
  - nvidia
  - conda-forge
  - defaults
dependencies:
  - python=3.10.6
  - pip=22.2.2
  - pytorch-cuda=11.7
  - pytorch=2.0.0
  - torchvision=0.15.1
  - torchaudio=0.13.0
  - pandas
  - pip:
    - pytorch-lightning==1.7.7
    - tensorboard==2.10.1
    - tabulate>=0.8.9
    - tqdm>=4.62.3
    - pillow>=8.0.1
    - notebook>=6.4.5
    - jupyterlab>=3.2.1
    - matplotlib>=3.4.3
    - seaborn>=0.11.2
    - ipywidgets>=7.6.5
    - boto3
    - pyprojroot
    - scikit-learn
```



Recommended IDE: VSCODE

```
code --install-extension aaron-bond.better-comments
code --install-extension chouzz.vscode-better-align
code --install-extension eamodio.gitlens
code --install-extension ecmel.vscode-html-css
code --install-extension esbenp.prettier-vscode
code --install-extension mechatroner.rainbow-csv
code --install-extension ms-python.isort
code --install-extension ms-python.python
code --install-extension ms-python.vscode-pylance
code --install-extension ms-toolsai.jupyter
code --install-extension ms-toolsai.jupyter-keymap
code --install-extension ms-toolsai.jupyter-renderers
code --install-extension ms-toolsai.vscode-jupyter-cell-tags
code --install-extension ms-toolsai.vscode-jupyter-slideshow
code --install-extension ms-vscode-remote.remote-containers
code --install-extension ms-vscode-remote.remote-ssh
code --install-extension ms-vscode-remote.remote-ssh-edit
code --install-extension ms-vscode-remote.remote-wsl
code --install-extension ms-vscode-remote.vscode-remote-extensionpack
code --install-extension ms-vscode.cpptools
code --install-extension ms-vscode.remote-explorer
code --install-extension ms-vsliveshare.vsliveshare
code --install-extension ms-vsliveshare.vsliveshare-audio
code --install-extension ms-vsliveshare.vsliveshare-pack
code --install-extension P-de-Jong.vscode-html-scss
code --install-extension snakemake.snakemake-lang
code --install-extension VisualStudioExptTeam.intellicode-api-usage-examples
code --install-extension VisualStudioExptTeam.vscodeintellicode
code --install-extension vscode-icons-team.vscode-icons
code --install-extension Zignd.html-css-class-completion
```

VSCODE Live

- Live Terminal and Text Editor sharing allows for fast collaboration
- Mob timer allows developers to use “Ensemble Programming” best practices
- Extensions allow for use of tools like Github Copilot
- IDE alternative to CLI Github interactions

Interacting with Cloud Storage

- Cloud storage services such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Azure from Microsoft allow commercial users to store and access their data remotely in the cloud for a fee
- This storage can be used to store
  - files
  - images
  - videos
  - databases
  - backups
- The BPS Microscopy Dataset is hosted on AWS as part of their public data registry

Considerations with Cloud Storage

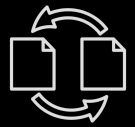
- Cloud storage buckets do not have directories
  - flat structure where all objects (i.e., files) are stored as blobs with unique object keys.

- Simulates a directory structure by including "/" characters in object keys.
  - For example, an object with the key "data/images/image1.jpg" could be interpreted as being located in a directory called "data/images/".

- To traverse directories use API provided by the cloud storage service's SDK or API.
  - Filter files based on their object keys
  - Filter by a common prefix (i.e., the same directory), you can effectively traverse the directory structure.

Ways to Interact with Cloud Storage

- **Web Based:** Use the cloud provider's web-based console to browse and download data files.
- **Command Line:** Use command-line tools such as awscli, gcloud, or az to interact with the data programmatically from a terminal.
- **APIs and SDKs:** Use APIs and SDKs provided by the cloud providers to interact with the data programmatically from within code. For example, AWS provides the **boto3** Python library, GCP provides the google-cloud-storage Python library, and Azure provides the azure-storage-blob Python library.
- **Use third-party tools:** such as Cyberduck or FileZilla to transfer data between the cloud repository and a local machine.
- **Use specialized tools:** and frameworks such as DVC or Pachyderm to version and manage data in cloud repositories as part of a machine learning workflow.



Streaming the Data

Using an API:

- Can retrieve individual files from a specific storage bucket blob as a **BytesIO** object to avoid saving the file in your local machine
- To do this you need:
  - Instantiate the cloud server client with the correct configuration
  - Provide the bucket name
  - Provide the “file path”
- By streaming you
  - Retrieve the file as a buffer object
    - Acts like a file object
    - Useful for manipulating binary data that is being generated or consumed by a program

Downloading the Data Locally

Using an API or CLI:

- Can select data to download based on metadata/filename
- Will require standardization of directory naming to avoid awkward path dependencies amongst team members
- Local copies are faster for the DataLoader to retrieve when we get to modeling stage

Why Metadata is Important

- Provides important data attributes
  - timestamps
  - filenames with extensions
  - labels
- Filename is important to access data in batches by Dataset objects used in deep learning frameworks like:
  - PyTorch
  - Tensorflow
  - JAX
- Metadata attributes provide interpretability of data
- You can subset metadata information to do data versioning