CPE 486/586: Machine Learning for Engineers

02 Tools for Machine Learning

Fall 2025

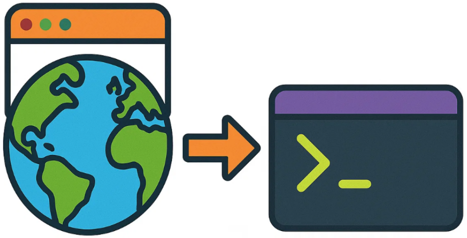
Rahul Bhadani

Outline

Command Line Tools and Linux

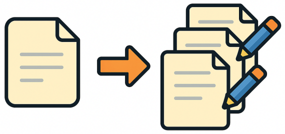
Why Command Line Tools and Linux

1. Download data from another location, webpage or server



Why Command Line Tools and Linux

2. File operation, renaming, copying, editing, etc, in bulk.



3. Logging into remote server
4. To integrate with other languages and tools.
5. Command Line provides scalability, extensibility, and can automate the entire pipeline for machine learning and data science.

Command Line

```
refulgent@refulgent:~$
```

1. Terminals in Linux and Macbook
2. Windows. Several Options:
   1. Git Bash. Install guide: https://youtu.be/SsdpuprzRE0
   2. Cygwin. Install guide: https://youtu.be/_j0Prs7aggo
   3. WSL.Install guide: 1. https://youtu.be/GMhV5Uqd8R8, 2. https://youtu.be/NPuIUT_6NeM?si=g0N39WfsBf3kIKCx

Common Linux Commands

My recommendation is to use Windows Subsystem for Linux, as it provides full linux funtionality to Windows.

A good tutorial to get started is **Getting started with Linux and Bash** at https://learn.microsoft.com/en-us/windows/wsl/tutorials/linux

1. Installing Software:

   ```
   sudo apt-get install vim git
   ```

2. Get the working directory path:

   ```
   pwd
   ```

3. Change directory:

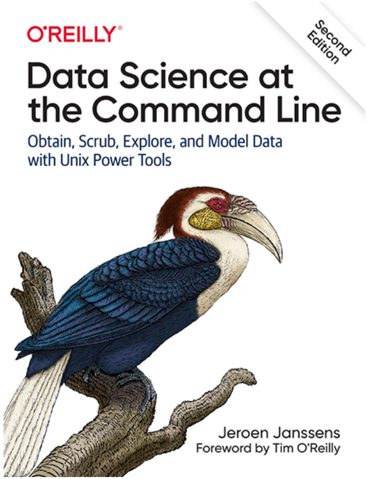
   ```
   cd /data/ch02
   ```

4. Create a directory or folder:

   ```
   mkdir data
   ```

An in-depth tutorial is at

https://jeroenjanssens.com/dsatcl/chapter-1-introduction


O'REILLY
Second Edition
Data Science at the Command Line
Obtain, Scrub, Explore, and Model Data with Unix Power Tools
Jeroen Janssens
Foreword by Tim O'Reilly

Remote Logging

If you do not have a powerful machine, you can remotely logging into UAH Engineering Linux Server using your UAH ID.

```
    ssh rkb0022@blackhawk.ece.uah.edu
```

Windows users should use WSL Terminals.

Python

Installing Python

Most machine learning libraries and codebase are written Python. Python is supported by a strong open-source community.
Windows users download from https://www.python.org/downloads/
Mac users download from https://www.python.org/downloads/macos/
Linux users install from command line

```
    sudo apt-get install python3.12
```

or

```
    sudo apt-get install python3.12-full
```

Recommended version of Python for this course is Python 3.12.

Git and GitHub

Version Controlling your Development

It is must that any of your machine learning project, and in general any coding project should use Git.
You should already have Git installed in your system through Git, otherwise install.

1. Create a GitHub account and a user name.
2. Create a new repository.
3. Clone to local machine:

```
git clone https://github.com/rahulbhadani/TestRepo TestRep
```

Commonly used Git Commands I

1. Initialize an existing local folder as a git repo.

```
git init
```

2. Add remote to your local git folder (Only needs to do once per repo).

```
git remote add origin https://github.com/rahulbhadani/TestRepo.git
```

3. Add files for commit.

```
git add . # add all files
git add somefile.txt # Add specific file(s)
```

4. Commit changes with a message before you push to the remote.

```
git commit -m "Added some files" # add a meaninful message
```

Commonly used Git Commands II

5. Push to the remote.

```
git push  # shortcut command
git push -u origin # push to a specific remote named origin
git push -u origin2 HEAD:master # push to specific origin's specific branch
```

6. Pull updates from a remote.

```
git pull  # shortcut command
git pull origin master # pull from specific remote
```

7. Git undo add before commit

```
git reset filename.txt
```

8. Git undo add after commit but before push and keep your changes

Commonly used Git Commands III

9.
```
        git reset --soft HEAD~1
```
10. Git undo add after commit but before push and unstage
11.
```
        git reset HEAD~1
```
12. Git undo add after commit but before push and discard every changes
13.
```
        git reset --hard HEAD~1 # use n for undoing n multiple commits
```

Development Environment

Integrated Development Environment (IDE)

Why use IDEs?

1. Provides integrated view of editor, file explorer, command lines.
2. Some IDEs provide intellisense for autocomplete.
3. IDEs are also integrate with compiler/inteprerter – one click to run your code.
4. IDEs are beautiful, rich colors and fonts over boring plain text editors.
5. Many IDEs add support for additional features such as visualization, remote log in, data wranling, etc.

VS Code

Many IDEs are available such PyCharm, Atom, VS Code.

For this course, I recommend VS Code.

Visual Studio Code

VS Code supports several extensions for Python, and other necessary tools.

Installing VS Code

Download: https://code.visualstudio.com/download
On Linux, downloaded deb file can be install at command line:

```
    sudo dpkg -i code_1.103.1-1755017277_amd64.deb
```

VS Code Extension

Recommended VS Code Extension:

1. Jupyter (Microsoft)
2. Jupyter Notebook Renderers (Microsoft)
3. Python (Microsoft)
4. Pylance (Microsoft)
5. Python Debugger (Microsoft)
6. Python Environments (Microsoft)
7. Remote – SSH (Microsoft)
8. Remote – SSH: Editing Configuration Files (Microsoft)
9. Remote Explorer (Microsoft)

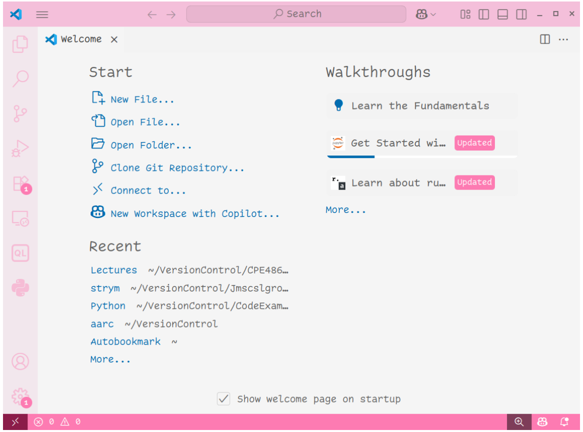
Connecting to Remote Machine in VS Code

When connecting to remote machine on VS Code, environment variables of the remote machine takes into effect.

To connect with `blackhawk.ece.uah.edu` from outside the Engineering building, we need VPN.

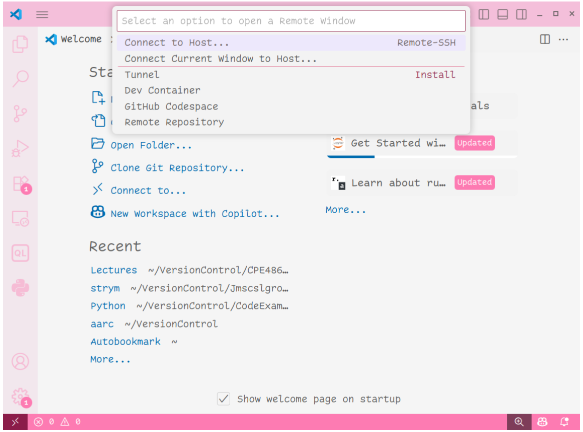
Download VPN: `https://chargerware.uah.edu/all-software/secure-access-vpn`

Connecting to Remote Machine in VS Code


Search
Welcome
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Connect to...
New Workspace with Copilot...
Walkthroughs
Learn the Fundamentals
Get Started wi... Updated
Learn about ru... Updated
More...
Recent
Lectures ~/VersionControl/CPE486...
strym ~/VersionControl/Jmscslgro...
Python ~/VersionControl/CodeExam...
aarc ~/VersionControl
Autobookmark ~
More...
Show welcome page on startup

After connecting to campus VPN, click **Connect to ...**

Connecting to Remote Machine in VS Code


Select an option to open a Remote Window
Connect to Host... Remote-SSH
Connect Current Window to Host...
Tunnel Install
Dev Container
GitHub Codespace
Remote Repository
Welcome
Open Folder...
Clone Git Repository...
Connect to...
New Workspace with Copilot...
Get Started wi… Updated
Learn about ru… Updated
More...
Recent
Lectures ~/VersionControl/CPE486…
strym ~/VersionControl/Jmscslgro…
Python ~/VersionControl/CodeExam…
aarc ~/VersionControl
Autobookmark ~
More...
Show welcome page on startup

Click **Connect to Host ...**, enter the address `username@blackhawk.ece.uah.edu`. Use your own user name.

Jupyter Notebook

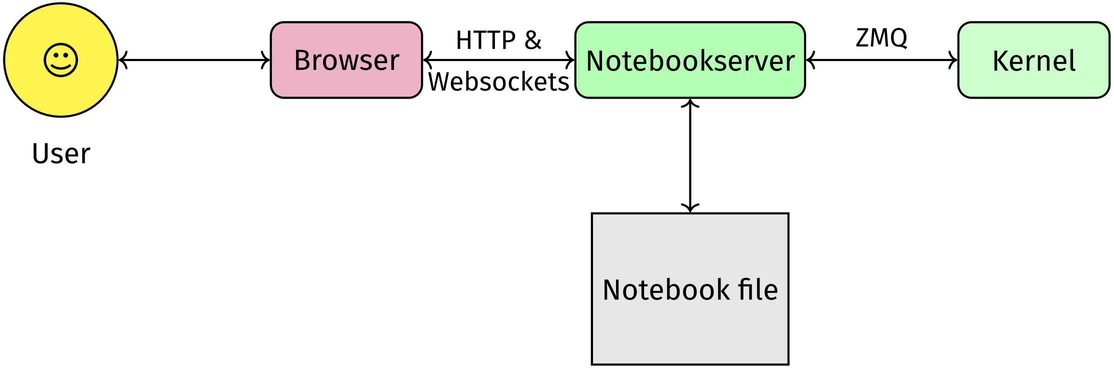
What is Jupyter Notebook?

Jupyter Notebook is an open source web application which can be used all sorts of data science tasks. The highlight of Jupyter Notebook is reproducibilit and shareability that can combine Markdown, Latex, and plain text for documentation, comments, math, and equations, along with code. In addition, Jupyter also supports Widgets for interactivity, and provides support for multiple programming languages. Some tasks that are possible are:

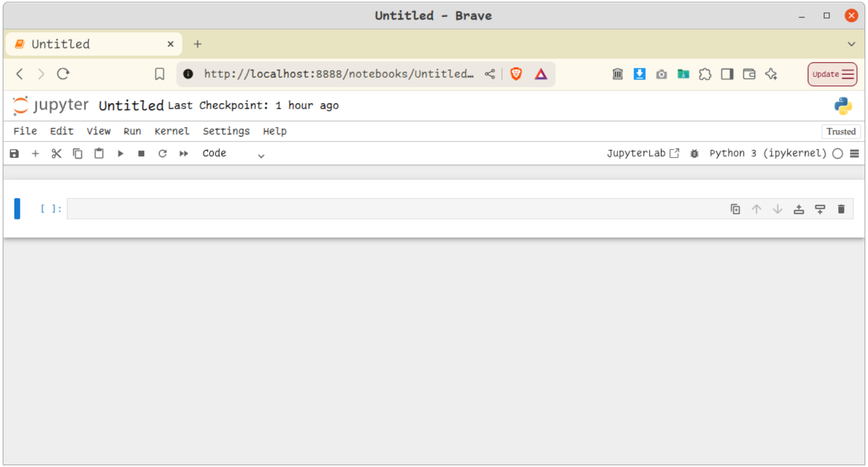
1. Data cleaning and transformation
2. Numerical simulation
3. Exploratory data analysis
4. Data visualisation
5. Statistical modeling
6. Machine learning
7. Lab notebook
8. Document sharing
9. Process documentation
10. Teaching and learning

How does Jupyter Notebook Work?

It can run locally on your own computer for lightweight use. It can also run on big servers, which we access remotely through the browser, for heavy computation.



Jupyter Nootebook


Untitled - Brave
Untitled
http://localhost:8888/notebooks/Untitled...
Update
jupyter Untitled Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[ ]:

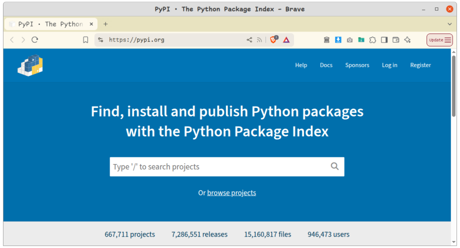
Packages and Package Manager

Package / library management – Intro

- Python has a vast number of packages and libraries
- Installation from **official** Python Package Index
  - Always try to install from the official package manager if possible
- The **pip** tool helps to find and install packages

Python Packages

- Python, as a programming language, is rather lean in terms of its built-in functions.
- With the help of packages, however, the range of functions can be expanded almost infinitely.
  - This is the official third-party software repository for Python.
  - It's also known as the Cheese Shop (`https://youtu.be/Hz1JWzyvv8A`).



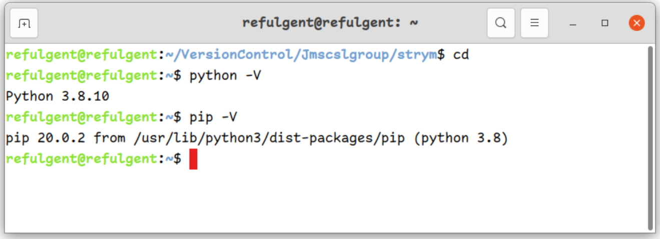
Find, install and publish Python packages with the Python Package Index

667,711 projects 7,286,551 releases 15,160,817 files 946,473 users

Figure: A screenshot of the Python Package Index homepage.

Python Package Tool – pip

Check your versions, you should have v3.0 for both, python & pip



```
refulgent@refulgent:~/VersionControl/Jmscslgroup/strym$ cd
refulgent@refulgent:~$ python -V
Python 3.8.10
refulgent@refulgent:~$ pip -V
pip 20.0.2 from /usr/lib/python3/dist-packages/pip (python 3.8)
refulgent@refulgent:~$ 
```

Figure: Check for PIP version

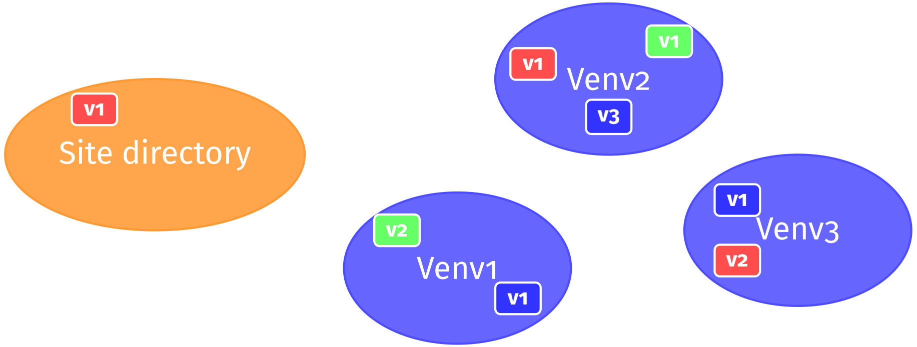
Installing Python packages through PIP

```
pip install numpy # Install specific packages
```

```
pip install -r requirements.txt  # Install packages from a list of files
```

Virtual environments

1. Virtual environments isolate the version of the Python interpreter (Python binary) and all its packages in aseparated directory
2. Virtual environments are useful when you want to isolate one development activity from another, as some package versions might conflict with another versions.



Introduction to UV

UV is a Python package and project management tool.

Once upon a time ...

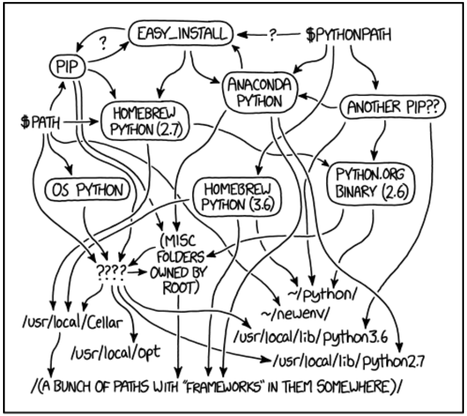


Figure: https://xkcd.com/1987/

Objectives

1. **Manage packages and dependencies:** `pip`
2. **Each project in its own environment:** `venv`, `poetry`, `hatch`, `pdm`, `rye`...
3. **Each tool in its own environment:** `pipx`
4. **Manage different Python versions:** `pyenv`

> `uv` can potentially replace them all !!!

Limitation of `uv`

1. Not an alternative to Conda.
2. Can only install Python packages
3. However, easy to integrate with RUST programming language

Installation

1. Linux/Mac/WSL on Windows:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

2. Windows:

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

3. Doesn't require admin rights

Python version management

1. **List installed and available Python versions:**

```
uv python list
```

2. **Install a Python version:**

```
# Install latest version
uv python install
```

```
# Install a specific version
uv python install 3.12
uv python install 3.11.10
```

Python version management

Run Python:

1. **Latest version:**

```
uv run python
```

2. **Specific version:**

```
uv run --python 3.11 python
```

3. **System version:**

```
python
```

Note:

```
uv doesn't modify the PATH.
```

Python version management

Pin a Python version in a directory:

```
# Fix default Python version
cd python_project
uv python pin 3.12
uv run python
```

If a requested Python version is not installed, `uv run` will install it automatically:

```
uv run --python 3.9 python
```

Remove an installed Python version:

```
uv python uninstall 3.9
```

Dependencies

Run a Python interpreter with dependencies:

```
# One dependency
uv run --with numpy python
```

```
# Several dependencies
uv run --with numpy --with matplotlib python
```

Run a command from a dependency:

```
# Launch a jupyter lab instance
uv run --with jupyterlab jupyter lab
```

```
# Launch a jupyter lab instance with numpy installed under Python 3.12
uv run --python 3.12 --with jupyterlab --with numpy jupyter lab
```

Script dependencies

Suppose we have the following `test.py` script:

```
import numpy as np
print(np.random.rand())
```

We can run the script in a temporary virtual environment with `numpy` with:

```
uv run --with numpy test.py
```

Script dependencies

We can also add the dependency directly in the script:

```
uv add numpy --script test.py
uv run test.py
```

Dependencies have been added as inline script metadata (PEP 723):

```
# /// script
# requires-python = ">=3.12"
# dependencies = [
#     "numpy",
# ]
# ///
import numpy as np
print(np.random.rand())
```

A script with inline metadata can be run with `uv` on any machine: required python version and dependencies will automatically be installed in a temporary virtual environment.

Projects

More complex projects must have their own persistent virtual environment.

```
# Create a project in a new folder
uv init my_project
# Create a project in the current folder
uv init .
```

`uv init` will:

- create a `pyproject.toml` and a `.python-version` file
- initialize a git repository
- create sample `hello.py` and `README.md` files

Projects

The `pyproject.toml` file contains the project metadata and dependencies. It can be edited manually, or via `uv` commands.

```
# Add a dependency to current project
uv add numpy
# Add a dependency to current project with a minimum version
uv add "polars>=1.6"
# Add a dependency to current project with a specific version
uv add "polars==1.5"

# Remove dependency from current project
uv remove polars

# Add a development dependency
uv add --dev black
```

If your project is itself a Python package, it will be installed in the virtual environment as an editable dependency.

Projects

Dependencies constraints are resolved in the `uv.lock` lockfile.

- `uv lock` ensures that the lockfile is still consistent with the dependencies constraints
- `uv sync` synchronizes the virtual environment with the lockfile, installing / removing dependencies if needed

Note

- `uv lock` and `uv sync` are automatically run after `uv add`, `uv remove` or `uv run`.
- `uv lock` doesn't upgrade packages if new versions are available. Add `--upgrade` argument to do it.

Note

The virtual environment is created in `.venv` in the project folder. It can still be activated with `source .venv/bin/activate`.

Projects

`uv run` in a project folder (or one of its subfolders) runs the command inside the project's virtual environment.

```
# Run Python in home folder with no dependencies
cd ~
uv run python
```

```
# Run Python in a project folder. Its dependencies will be available.
cd my_project
uv run python
```

```
# Run a script in the project environment
uv run main.py
```

Projects

If an `uv` project is cloned from a git repository or copied to another machine, running `uv sync` will install the required Python version and dependencies from its `uv.lock` lockfile.

Running `uv run` to launch scripts or commands will also automatically lock the dependencies and sync the environment.

Tools

Some Python packages are installed for the application they provide:

- `ruff`
- `black`
- `ipython`
- `py-spy`
- `snakemake`

It is highly recommended to install these tools each in their own virtual environment.

Tools

`uvx` launches a tool in a temporary environment without installing it:

```
uvx pycowsay "Hello !"
```

`uv tool` allows to install the tool in its own environment and make its binary available in the PATH:

```
uv tool install pycowsay
pycowsay "Hello !"
```

Pay close attention to the package name when running or installing a tool. A wrong name could run a malicious command from a supply chain or typosquatting attack.

Tools

Commands to manage Python tools:

```
# List installed tools
uv tool list

# Install a tool
uv tool install radian

# Upgrade a tool
uv tool upgrade radian
# Upgrade all tools
uv tool upgrade --all

# Uninstall a tool
uv tool uninstall radian
```

Tools

(Temporary) caveats

- If the default `uv` Python version is changed, tools need to be reinstalled.
- Some tools need workarounds to be installed completely, for example:

```
# To install Snakemake
AR=/usr/bin/ar uv tool install snakemake
```

```
# To install Ansible
uv tool install ansible-core --with ansible
```

Creating and Installing a Package with UV

```
# Create a uv project
uv init --lib --package myawesomepackage --build-backend maturin --author-from git
cd myawesomepackage
# install python
uv python install 3.12
# create a virtual environment
uv venv --python 3.12
# activate virtual environment
source .venv/bin/activate
```

Creating and Installing a Package with UV

Let's look at the structure of the package:

```
|-- Cargo.toml
|-- pyproject.toml
|-- README.md
\|-- src
    |-- lib.rs
    \|-- myawesomepackage
        |-- _core.pyi
        |-- __init__.py
        |-- py.typed
```

Inside `src/myawesomepackage` create a folder called `arithmetic`.

```
mkdir arithmetic
cd arithmetic
```

Creating and Installing a Package with UV

Now we will create a module in `arithmetic` subpackage.

Module

A module is a file containing Python definitions and statements. A module can contain executable statements as well as function definitions. Read more at https://docs.python.org/3/tutorial/modules.html

```
touch __init__.py # a subpackage must contain __init__.py
code series.py
```

and enter the following code by copying from https://gist.github.com/rahulbhadani/a95e5f57bd7bb906755f443a91d7b0c2

Creating and Installing a Package with UV

Add the following code to `__init__.py`.

```
"""
Arithmetic subpackage containing mathematical progression functions.
"""

from .series import arithmetic_progression, arithmetic_sum

__all__ = ['arithmetic_progression', 'arithmetic_sum']
```

Creating and Installing a Package with UV

Now, we build the package. Change the directory to the root directory of your `myawesomepackage`, and type

```
uv build
```

This will generate a .whl file and a .tar.gz file in `dist/` folder. You can upload this to GitHub release to be installed later.
Alternatively, you can upload PYPI. You will need to create an account on PYPI and note down your secret code. However, to upload on PYPI, your package need to have a unique name. When you upload on PYPI, you can install from PYPI directly using `uv add packagename` or `uv pip install packagename`

```
uv add twine
twine upload dist/*
```

Note:

You may need to rename your .whl file so that it ends in `...cp39-abi3-manylinux_2_28_x86_64.whl`

Creating and Installing a Package with UV

Installing the package

Let's install package in another UV environment. Open a new terminal, and type

```
uv init arithmetic_test
cd arithmetic_test
uv venv --python 3.12
source .venv/bin/activate
uv add https://github.com/rahulbhadani/CPE486586_FA25/releases/download/0.1/myawesomepackage-0.1.0-cp39-abi3-linux_x86_64.whl
```

If you uploaded your package to PYPI, you can install using

```
uv add myawesomepackage
```

or

```
uv pip install myawesomepackage
```

Using Installed Package with UV

We will demonstrate how to use Install Package in UV jusing Jupyter and VS Code. Navigate to the project folder you created and activate the virtual environment:

```
cd arithmetic_test
source .venv/bin/activate
uv add --dev jupyter
uv add --dev ipykernel
# create a kernel for jupyter
uv run ipython kernel install --user --env VIRTUAL_ENV $(pwd)/.venv --name=arithmetic_test

# Launch Jupyter Notebook
jupyter notebook
```

It will launch the Jupyter Notebook in your browser. There you can create a new .ipynb file select kernel as **arithmetic_test** as it is the name we gave to the kernel above.

Using Installed Package with UV

Now, in the notebook, add the following code and execute

```
# Option 1: Import the function directly
from myawesomepackage.arithmetic.series import arithmetic_progression
term_10 = arithmetic_progression(5, 2, 10)
term_10

# Option 2: Import from the subpackage (due to \_\_init\_\_.py)
from myawesomepackage.arithmetic import arithmetic_progression

# Option 3: Import the module
from myawesomepackage.arithmetic import series
result = series.arithmetic_progression(5, 2, 10)
result
```

Setting Python Virtual Environment using Anaconda

Alternatively, we can use Anaconda instead of UV for virtual environment as well, however, I recommend using UV for this course.

Download: `https://www.anaconda.com/download/success`

Creating Virtual Environment in Python using Anaconda

```
conda create --name pythonenv python=3.11
conda activate pythonenv
```

Installing Jupyter Kernel in Your Anaconda Environment

The following code snippet install python kernel for use with Jupyter Notebook within your Anaconda virtual environment.

```
conda install ipykernel
ipython kernel install --user --name=pythonenv
conda install jupyter
jupyter notebook
```

Clound-based Solution

Google Colab

https://colab.research.google.com/

- No local setup needed.
- Packages can be installed as needed.
- GPU can be bought if needed.

Other Notable Tools

Tools helpful for Machine Learning and Data Science

1. R Language and R Studio
2. Julia Programming Language
3. Tableau
4. Power BI
5. Structured Query Language (SQL)
6. Statistical Analysis System (SAS)

The End