Discrete Structures!

CMPSC 102
Setting Up Projects
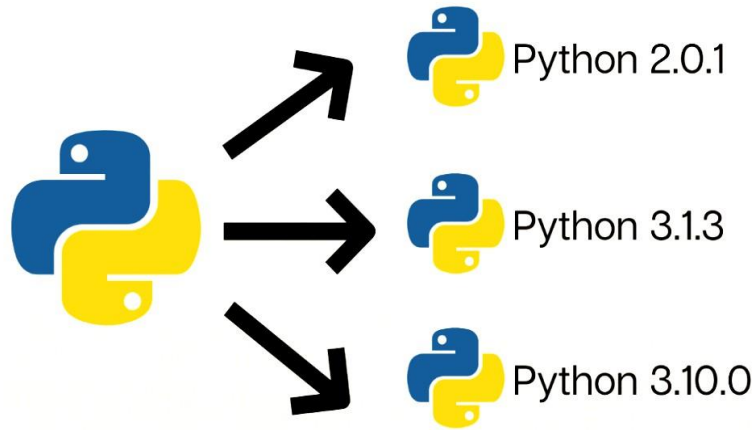
Key Questions and Learning Objectives

- How do I use virtual environments like Venv and Poetry, along with tools like Typer and other resources, to create a professional Python project?

- To learn how to use libraries and dependencies for development with Python code and programming techniques to create the foundations for a professional project.
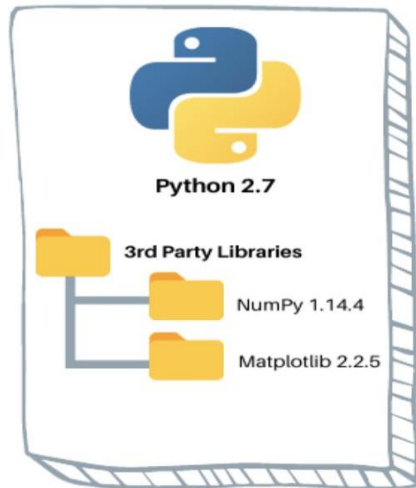
Virtual Environments

- Projects' dependencies and specific versions of libraries.
- Not all projects require the same dependencies; How do you mix projects on your computer?
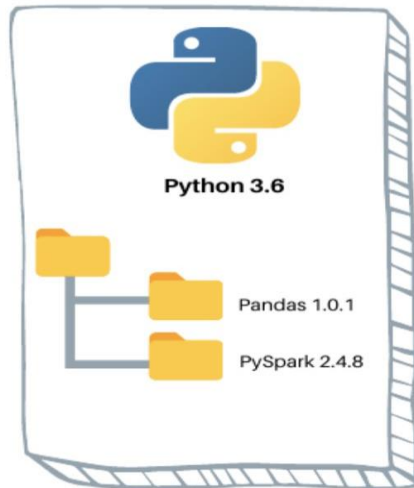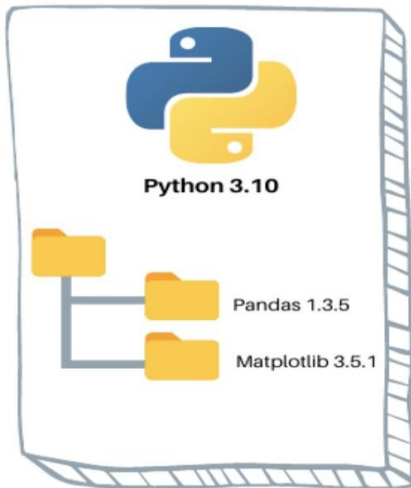
Go Virtual

Virtual Environment 1

Python 2.7

3rd Party Libraries

NumPy 1.14.4

Matplotlib 2.2.5

Virtual Environment 2

Python 3.6

Pandas 1.0.1

PySpark 2.4.8

Virtual Environment 3

Python 3.10

Pandas 1.3.5

Matplotlib 3.5.1

- Virtual environments maintain specific libraries and dependencies projects
- Shipping software: build an exact copy of development environment on client's machine to use software.

Regression Analysis Project

- Decide on the purpose and composition of the project

    - Our project: a regression analysis demonstration from SciKit-Learn

        - https://scikit-learn.org/stable/supervised_learning.html#

    - No command line parameters
    - No output, other than screen printing
    - Execution: Program complete regression analysis of random values
    - One function in project: main()
    - **Dependencies**: scikit-learn, numpy, seaborn

Setting Up Virtual Environment

- Create a project directory

```
mkdir projects
cd projects
```

- Create virtual environment using Python

```
python3 -m venv myenv
# see the file tree
find . -not -path '*/\.*'
```

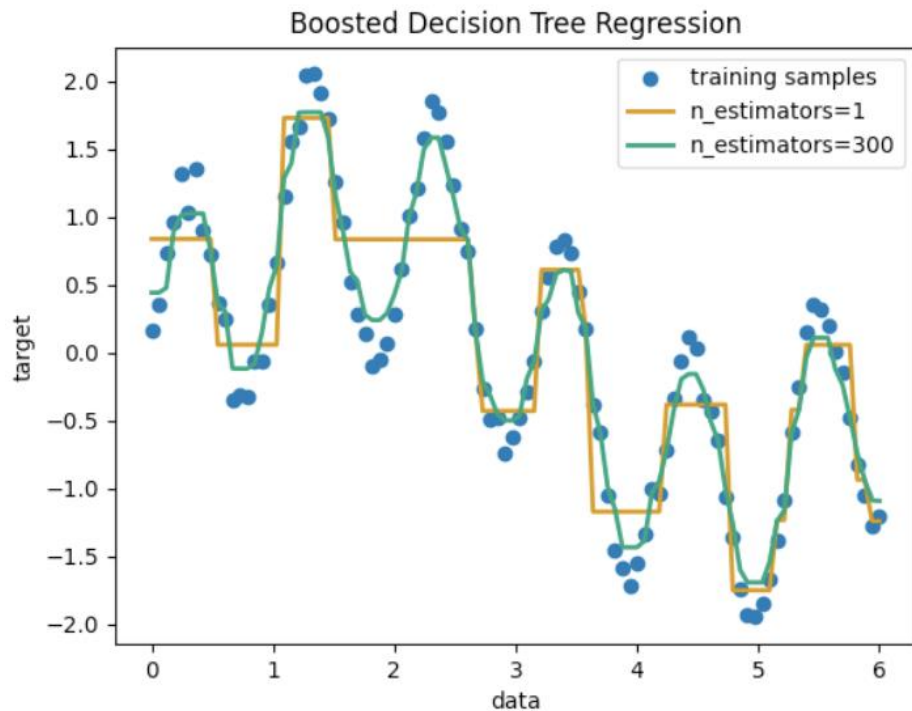- Activate myenv the virtual environment

```
source myenv/bin/activate  # macOS/Linux
myenv\Scripts\activate     # Windows
```

- Activate myenv the virtual environment

```
pip install numpy
pip install seaborn
pip install scikit-learn
# or try: pip install sklearn
```

Output From Executing the Script

- Execute given Script in myenv : Python3 sciKitDemo.py

Boosted Decision Tree Regression

Black: a Python Script formatter

Got Clutter?

- How to maintain *readable* code?
- How to reduce white-space in code to improve readability?

- A code formatting resource
- https://black.readthedocs.io/en/stable/getting_started.html

Black: a Python Script formatter

- Install Dependencies

  ```
  pip install black
  ```

- Linting example: As a String to Printed Line

  ```
  black --code "print ( 'hello, world'      )"
  ```

- Linting example: Standard Input to File

  ```
  echo "print ( 'hello, world' )" | black - > out.txt
  ```

  https://black.readthedocs.io/en/stable/usage_and_configuration/the_basics.html

- We will use this with Poetry (up next)

We Need Poetry!!

Work without Hope, by Samuel Taylor Coleridge Lines Composed 21st February 1825

*All Nature seems at work. Slugs leave their lair The bees
are stirringbirds are on the wing
And Winter slumbering in the open air, Wears on his
smiling face a dream of Spring! And I the while, the
sole unbusy thing,
Nor honey make, nor pair, nor build, nor sing.*

*Yet well I ken the banks where amaranths blow,
Have traced the fount whence streams of nectar flow.
Bloom, O ye amaranths! bloom for whom ye may, For
me ye bloom not! Glide, rich streams, away!
With lips unbrightened, wreathless brow, I stroll: And
would you learn the spells that drowse my soul? Work
without Hope draws nectar in a sieve,
And Hope without an object cannot live.*

A Bigger Virtual Environment

PYTHON PACKAGING AND DEPENDENCY MANAGEMENT MADE EASY

Poetry

*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. Poetry offers a **lockfile** to ensure repeatable installs, and can build your project for distribution.*

Python Resource - Poetry

PYTHON PACKAGING AND DEPENDENCY MANAGEMENT MADE EASY

Poetry

https://python-poetry.org/

- Management support for Python and its resources
- Environments: manage dependencies in isolation
- Package: create a stand-alone executable application
- Publish: expedite and simplify the release of program to PyPI

Python Resource - Typer

https://typer.tiangolo.com/
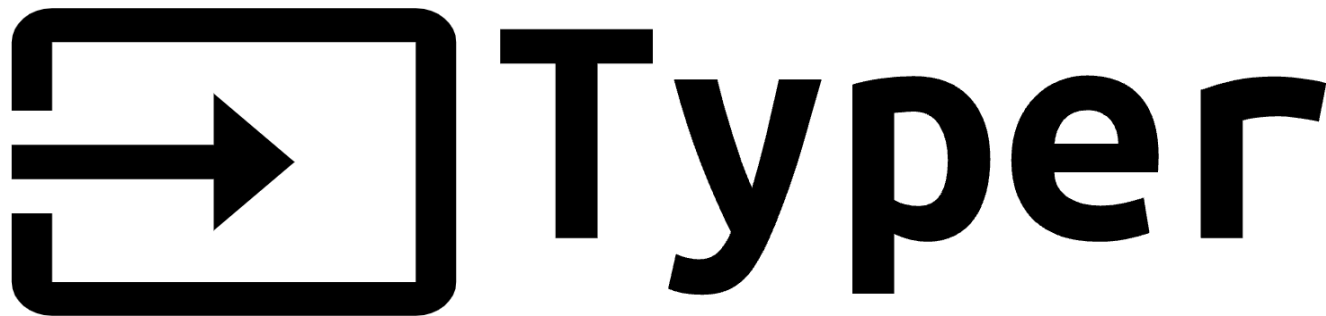
- Command line interface support for program inputs and parameters
- Annotations: assigns types to functions that accept arguments (parameters)
- Productivity: types aid in the creation of the interface
- Checking: Confirm that inputs match expected types.

A Hello User Program

- Decide on the purpose and composition of the project

  - Our project: Say hello to the user

  - Parameters firstName, middleName and lastName as parameters

  - No output, other than screen printing

  - Execution: Program greets user by full name

  - One function in project: main( )

Setup Steps

- Make a working directory

```
mkdir projects
cd projects
```

- Use Poetry to create new project

```
poetry new hello_user
cd hello_user
```

- Add Project Dependencies

```
poetry add typer
poetry add rich
```

- Add Project Development Dependencies

```
poetry add -D black mypy
```

Mypy: http://mypy-lang.org/

Setup Steps

- Add File: projects/hello user/hello user/ init .py

    """"Required docstring for an init file."""

    __version__ = "0.1.0"

- Add File:projects/hello user/pyproject.toml

    [tool.poetry] ...

    ```
    [tool.poetry.scripts]
    hello_user = "hello_user.main:cli"
    ```

    ```
    [tool.poetry.dependencies] ...
    ```

- Update Poetry

    ```
    poetry install
    ```

Add File: projects/hello user/hello user/main.py - File located in sanbox: main.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from rich.console import Console
import typer

# create a Typer object to support the command-line interface
cli = typer.Typer()
@cli.command()
def main(first: str = "", middle: str = "", last: str = ""):
          """Say hello to the person having a name of first, middle and last name"""
          console = Console()
          console.print(" Hello to;")
          console.print(f"\t First = {first}")
          console.print(f"\t Middle = {middle}")
          console.print(f"\t Last = {last}")
# end of main()
```

Basic Reformatting with Black

poetry run black hello_user tests

Execute Project

What do you see?

```
# run from projects/hello_user/hello_user
poetry run python3 main.py --help
```

```
Usage: main.py [OPTIONS]

Say hello to the person having a name of first, middle and last name

 Options
 --first                 TEXT
 --middle                TEXT
 --last                  TEXT
 --install-completion    [bash|zsh|fish|powershell|pwsh]  Install completion for the specified shell. [default: None]
 --show-completion       [bash|zsh|fish|powershell|pwsh]  Show completion for the specified shell, to copy it or customize the installation. [default: None]
 --help                                                   Show this message and exit.
```

Execute Project

- What do you see?

<pre>
run from projects/hello_user
poetry run hello_user
</pre>

- Without parameters

<pre>
poetry run hello_user
Hello to;
            first =
            middle =
            last =
</pre>

Execute Project

- What do you see?

    # run from projects/hello_user poetry run hello_user

    --first Darth

    --middle Cornelius

    --last Vader

- Without parameters

    Hello to;

    first = Darth
    middle = Cornelius
    last = Vader