Dependency management

With a focus on Python: pip, venv and Conda

Contents

- Intro to problem
- venv
- Editable installs: the ‘proper’ way to code
- conda
- Workshop

The problem(s)

- Downloading and installing many packages `pip`
  - Ensuring they, and all their dependents, are all cross-compatible
  - This is called **dependency management**
- Being able to work on multiple projects on the same machine
  - Even if the projects use different library versions
- Making it easy for other people to run the code you wrote with the same outcome `pip`
  - Avoiding ‘it works on my machine’

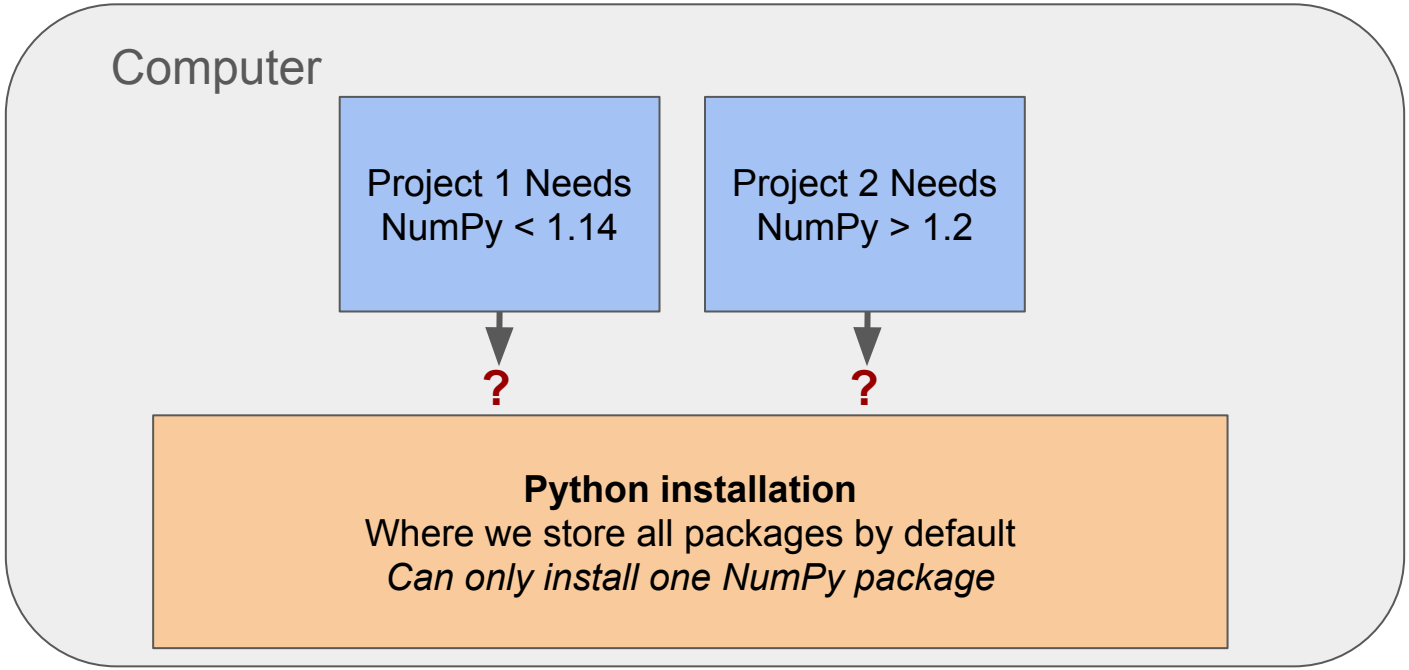
The problem(s)

- Downloading and installing many packages `pip`
  - Ensuring they, and all their dependents, are all cross-compatible
  - This is called **dependency management**
- Being able to work on multiple projects on the same machine `venv`
  - Even if the projects use different library versions
- Making it easy for other people to run the code you wrote with the same outcome `pip`
  - Avoiding ‘it works on my machine’

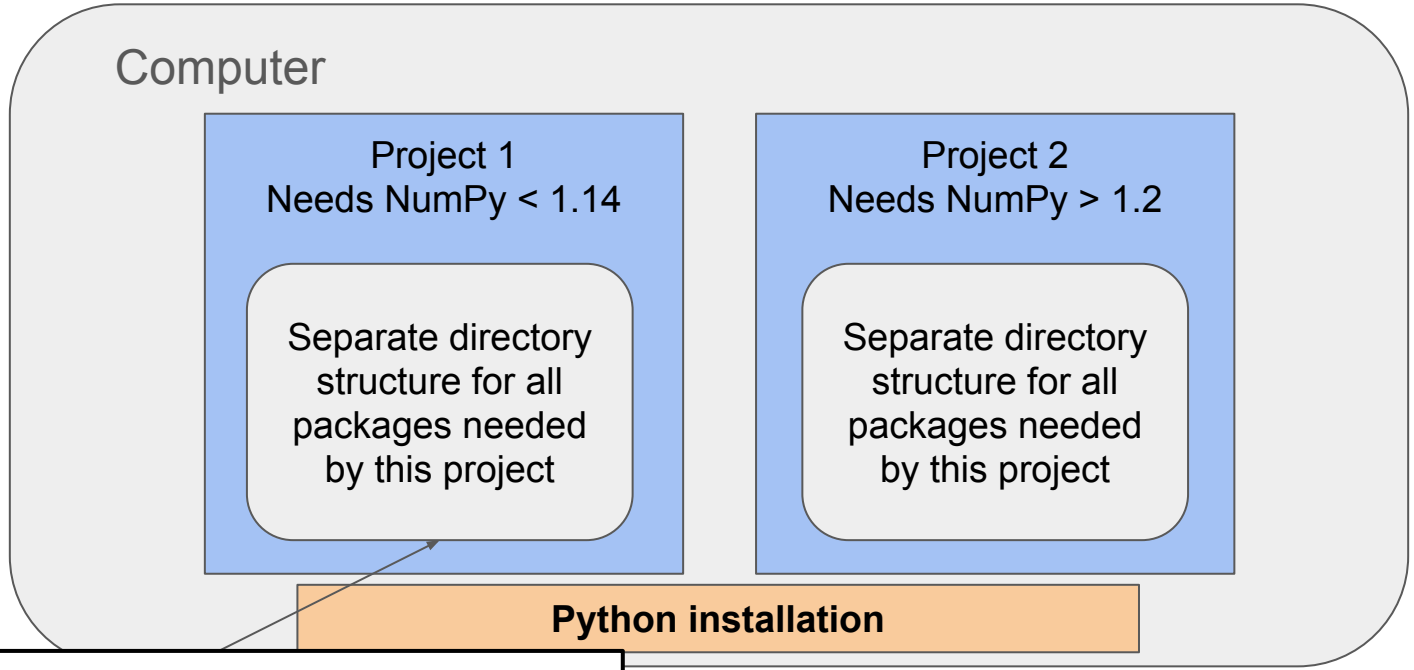
The problem(s)

- Downloading and installing many packages `pip` `conda`
  - Ensuring they, and all their dependents, are all cross-compatible
  - This is called **dependency management**
- Being able to work on multiple projects on the same machine `venv` `conda`
  - Even if the projects use different library versions
- Making it easy for other people to run the code you wrote with the same outcome `pip` `conda`
  - Avoiding ‘it works on my machine’

What happens without virtual environments



What happens with virtual environments/conda



This is a virtual environment!

Using venv

- Python only projects
- Easy, lightweight, unlikely to go horribly wrong
- Creates a sandbox of versions so one environment does not interfere with another

```
python -m venv venv
```

Command Flag Module Directory

```
source ./venv/bin/activate
```

On windows

```
source ./venv/Scripts/activate
```

```
pip install -r requirements.txt
```

Make your own with

```
pip freeze > requirements.txt
```

Bonus slide: Editable installs and easier importing

- Classic problem in Python importing, sibling directory import:
  - src
    - utils
      - util.py
    - preprocessing
      - clean.py

Needs

- Set up code to be installable, involves some minor arrangement of folders and a simple config file
  - https://packaging.python.org/en/latest/tutorials/packaging-projects/
- **`pip install -e .`**
- Can now easily import from sibling directories
  - `from src.utils.util import blabla`

Using conda

- Create environment
  - Preferably as many packages at once

```
conda create -n <name> [list of packages]
```

- Activate environment

```
conda activate <name>
```

- Act as normal/using a virtual environment
- To stop using (useful!)

```
conda deactivate
```

Jargon buster

- Channel
  - Where you look for and download packages from
- Condaforge
  - Unofficial common channel for packages
- Miniforge
  - Alternative version of conda (very confusing)
- Anaconda
  - Preconfigured set of packages managed by conda; typically for datascience
- .condarc
  - If you don’t know - don’t touch
  - If you are annoyed by conda always starting up in every terminal you can add:

```
auto_activate_base: false
```

Common problems with Conda

- Using multiple environments at once
  - If your prompt looks like `(base)(conda_env)` you are going to have a bad time
  - Simply deactivate all environments one by one, until no prompt.
- Confusing pip and conda usage
  - They are ok to use together - but be careful!
  - Activate environment before pip install
  - Don’t use venv + conda
- Super slow
  - Use miniforge
    - Under the hood uses faster version of conda called mamba

Workshop