Python Virtual Environments

Alex Lyttle | Skills Session | 14 May 2021

Overview

1. Prerequisits and definitions
2. Why should I use a virtual environment?
3. How do I create a virtual environment?
4. Will they work with Jupyter notebooks?
5. How do I manage different Python versions?
6. Live demonstration

Prerequisites

This skills session assumes you are familiar with the following,

- Python >= 3.3
- Conda >= 4.6 (optional)
- IPython >= 6.0 (optional)
- Python 2 (optional)

What is Python?

Python is an interpreted programming language. To run Python code we need to install an *interpreter*. You may then access the interpreter with the `python` command in your *terminal* application.

- Your computer searches the `PATH` for the first script named `python`.
- Depending on your system or installation, this could correspond to Python 2 or 3.
- You may need to type `python3` if both versions are installed.

What is `pip`?

You can install many packages (available locally, or remotely via PyPI) through the Python module `pip`. Using the `pip` command in your terminal,

- Your computer searches the `PATH` for the first script named `pip`.
- This may not correspond to the version of Python you want to use!
- If unsure, use `python -m pip` where `python` is your chosen version.
- Use `python -m pip --user` to install packages to your user only.

What is Conda?

Conda is a package and environment management system. Unlike `pip`, Conda can be used to manage environments with multiple programming languages.

If you installed Python with Anaconda, it is managed by the `base` Conda environment. You can use the `conda` command in the Anaconda Prompt, or in your terminal after you use `conda init` to configure your `PATH`. Restart your terminal to access the `python` command.

> Help! I updated a Python package and now my code is broken!

Example

We want to run a script `myscript.py`

- Imports some packages called `foo` and `bar` available on PyPI
- `bar` *depends* on a specific version of `foo`

We type the following into our terminal,

```
python -m pip install foo bar  # Installs script dependencies
python myscript.py             # Runs the script
```

Example

Later, we update `foo` to use a new feature,

```
pip install --upgrade foo
```

We run `myscript.py` again, but now there's an error! The `bar` package doesn't work with the updated `foo` package.

Why a virtual environment?

When you install `foo` and `bar` they are put in the `site-packages` directory associated with your Python interpreter. This is where Python accesses the package when you run code.

When you install a package, it may have *dependencies* — i.e. other required packages. Sometimes dependencies must be a *particular version* in order for a package to work. **Therefore, if you update one package, it could break another.**

What is a virtual environment?

- Allows you to keep dependencies required by different projects separate
- Updates your `PATH` to prioritise a specific Python interpreter
- Has its own isolated `site-packages` directory
- Updates your `sys.path` so that Python looks for packages installed within the isolated `site-packages` directory only

How do I make a virtual environment?

Depends on how you installed Python

- Anaconda/Miniconda uses `conda` to manage environments
- Otherwise
  - `venv`
  - `virtualenv` / `virtualenvwrapper`

Virtual environments with `conda` - Create

- Can be created in Anaconda Navigator
- Or, can be created in Terminal or Anaconda Prompt (see below)

To create an environment named `myenv` with access to the latest version of Python,

```
conda create --name myenv python
```

Virtual environments with `conda` - Use

To *activate* and use the environment,

```
conda activate myenv
# Example usage:
conda install foo
python myscript.py
```

To *deactivate* the environment,

```
conda deactivate
```

Virtual environments with `venv` - Create

- The simplest way to get started without `conda`
- Only works with Python 3 (for Python 2 see `virtualenv`)
- No installation needed

To create the environment,

```
python -m venv path/to/myenv
```

Virtual environments with `venv` - Use

To activate and use the environment,

```
source path/to/myenv/bin/activate
# Example usage:
pip install foo==1.0  # installing version 1.0 of a package
python myscript.py    # running a script
```

To deactivate the environment and go back to system Python,

```
deactivate
```

Virtual environments with `virtualenv` - Install

- Compatible with Python 2 and `virtualenvwrapper`
- More features (see the docs for examples)

To install `virtualenv`,

```
python -m pip install virtualenv
```

where `python` is your system python.

Virtual environments with `virtualenv` - Create

To create the environment,

```
virtualenv path/to/myenv
```

Activating and using the environment is the same as with `venv`.

Virtual environments with `virtualenvwrapper` - Install

- Provides memorable, easy to use commands
- Requires a bit more setup (see e.g. the docs)
- Need to use virtualenvwraper-win for Windows OS

To install `virtualenvwrapper`,

```
python -m pip install virtualenvwrapper
```

See here for more information on installation.

Virtual environments with `virtualenvwrapper` - Setup

For example, on Unix-like OS, add these lines to your shell startup file (e.g. `.bashrc`, `.profile`, `.zshrc`)

```
export WORKON_HOME=$HOME/.virtualenvs  # Path to virtual environments folder
export PROJECT_HOME=$HOME/Projects     # Path to your projects folder
source /usr/local/bin/virtualenvwrapper.sh  #
```

then reload the startup file,

```
source path/to/startup/file
```

Virtual environments with `virtualenvwrapper` - Create

To make an environment,

```
mkvirtualenv myenv
```

To activate and use the environment,

```
workon myenv
```

Deactivate in the same way as before with the `deactivate` command.

Virtual environments - Summary

-	`conda`	`venv`	`virtualenv` / `virtualenvwrapper`
Pros	Easy to use if you use conda to manage Python	Easy to use with Python 3	Provides more memorable commands and features
Cons	`conda` package management can be confusing	Lacks some features of its parent package	Requires installation and setup

Jupyter Notebooks

Assuming Jupyter is installed on your computer. What if you want to run a Jupyter Notebook within your virtual environment?

No need to install Jupyter in every environment!

You only need to install the IPython kernelspec for that environment. See here for more information.

Jupyter Notebooks - `conda`

If we want to run a Jupyter notebook in the `myenv` `conda` environment,

```
conda activate myenv  # make sure we are in myenv
conda install ipykernel
python -m ipykernel install --user --name myenv --display-name "Python 3 (myenv)"
```

Launch Jupyter and you should see a new kernel option named "Python 3 (myenv)".

Jupyter Notebooks - `pip`

It is similar when using `pip`,

```
source ~/.virtualenvs/myenv/bin/activate  # if using venv or virtualenv OR
workon myenv                              # if using virtualenvwrapper

pip install ipykernel
python -m ipykernel install --user --name myenv --display-name "Python 3 (myenv)"
```

Launch Jupyter and you should see a new kernel option named "Python 3 (myenv)".

> Help! I updated Python and now my code is broken!

How do I manage different Python versions?

- If using `conda` each environment can have its own Python interpreter, e.g.

```
conda create --name myenv python=3.7
```

  will create an environment specifically for Python 3.7

- `virtualenv` allows you to specify the path to a Python interpreter
- Otherwise, consider using `pyenv` (download instructions here)

What is `pyenv`?

- Can be installed with Homebrew on MacOS or compiled from source
- Manages several versions of Python on your machine
- Provides easy ways to install different Python versions
- Can be used with `virtualenv` and `virtualenvwrapper` via `pyenv-virtualenv` and `pyenv-virtualenvwrapper`
- If interested, see these blogs for getting started here and here

Workflow

Starting a new project with Python?

1. Create a dedicated environment for the project
2. Activate that environment
3. Install packages required for the project
4. Write and run code within that environment
5. Keep a list of requirements so others can run your code
6. Deactivate the environment when you're done

Demonstration

I will switch to the Terminal to demonstrate setting up and using virtual environments...

Thank you! Any questions?