Python and SQL: intro / SQL platforms

Ewa Weychert

Class 11: Virtual Environment

Why do we need virtual environments?

- Python projects depend on external libraries (Django, pandas, matplotlib)
- Different projects need:
  - different versions of the same library
  - different Python versions
- Installing everything globally causes:
  - version conflicts
  - broken system Python

Solution: isolate each project in its own environment

What is a virtual environment (venv)?

A **virtual environment** is:

- a folder containing:
  - a private Python interpreter
  - its own pip
  - its own installed packages
- completely independent from:
  - system Python
  - other projects

Think of it as a **sandbox** for one project.

macOS Homebrew: important warning

On macOS (Homebrew Python):

- global `pip install` is often **blocked**
- error comes from **PEP 668**

Typical error:

`externally-managed-environment`

Correct fix: use a virtual environment

Step 1: Go to your project folder

Open Terminal and navigate to your project directory:

```
cd ~/Desktop/class_11
```

- Always create the venv **inside the project folder**
- One venv per project

Step 2: Create the virtual environment

Run:

```
python3 -m venv .venv
```

What this does:

- creates a folder called `.venv`
- copies Python into it
- prepares isolated pip

Dot prefix (.venv) hides the folder — this is normal

Step 3: Activate the virtual environment

On macOS / Linux (zsh, bash):

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

After activation:

- terminal prompt changes
- you see `(.venv)` at the beginning

If you do not see (.venv), it is NOT active

Step 4: Verify activation

Check which Python is used:

```
which python
```

Correct output should contain:

.venv/bin/python

Check Python version:

```
python --version
```

Step 5: Install packages safely

Now you can install packages without errors:

```
python -m pip install django pandas matplotlib
```

Why this works:

- pip installs only into `.venv`
- system Python is untouched

Step 6: Use venv with Django

Always run Django commands like this:

```
python manage.py runserver
```

- **venv must be active**
- otherwise Django may not be found

Deactivating the virtual environment

When finished:

```
deactivate
```

- returns you to system Python
- safe to close terminal afterwards

Common mistakes (and fixes)

- `pip install django` (global)
  - activate venv first
- creating venv outside project
  - create inside project folder
- forgetting activation
  - look for `(.venv)` in terminal

Summary

1. Go to project folder
2. Create venv: `python3 -m venv .venv`
3. Activate: `source .venv/bin/activate`
4. Install packages safely
5. Run Django inside venv

One project = one virtual environment

Questions?