Import Python Project Tutorial with UV

Jeremy Mikecz

2026-01-13

Importing an existing Python Project with VSC and UV

This document provides detailed instructions for setting up your Python environment for this course/workshop. You should have already read the information in the 00_setup_slides and completed the installation of Visual Studio Code and Python using uv by following Setup Notebook 01.

Table of Contents

Downloading a Project Repository from Github

Option 1: Download as ZIP

This method is recommended for newer programmers not yet familiar with git and Github and for any user who only needs a static copy of the repository.

1. open the chosen GitHub repository. In this tutorial, we will import the repository for Dartmouth Libraries' Python Essential Workshop Series.
2. Click the green "Code" button on the repository page
3. Select "Download ZIP"
4. Extract the ZIP file to a memorable location (e.g., `~/Documents/Python-Courses/`)
5. **Important:** To get updates, you'll need to re-download

Option 2: Using Git (Advanced)

This method is recommended for users familiar with git and Github, have git installed, have a Github account, and who wish to create a dynamic copy ("clone") of the repository. A clone allows you to update your local repo when its online counterpart as been updated (using the pull command in git). It also allows you to push any changes you made locally back up to the remote repository, but only if the repository administrator has given you these privileges.

If you have Git installed:

```
# before cloning, move the current working directory to the folder where you want to clone t
git clone [REPOSITORY-URL]
# for example:
# git clone git@github.com:Dartmouth-Libraries/py-essentials_ws.git
```

To update the local version of the repository after the online version has been updated:

```
git pull origin main
```

Setting Up the Python Environment

Step 1: Navigate to the Project Directory

```
cd path/to/[repository-name]
```

Step 2: Create Virtual Environment and Install Dependencies

2a. If the Imported Project is a uv project

You can tell if a repository is set up for **uv** if it has a `uv.lock` file in its root folder.

Run the following in the **terminal**:

```
# This creates a .venv directory and installs all packages
uv sync
```

This command:

- Creates a virtual environment in `.venv/`
- Installs Python if the correct version isn't available

- Installs all packages specified in `pyproject.toml`
- Creates/updates `uv.lock` for reproducibility

2b. If the Imported Project has a `requirements.txt` file

Prior to **uv**, Python project dependencies were usually stored in a `requirements.txt` file. To install all dependencies from this file and convert the project to a uv project, run the following in the **terminal**:

1. First, initiate **uv** for this project:

```
uv sync
```

2.

```
uv add -r requirements.txt
```

If this doesn't provide desired results, review the documentation on this process in the uv guide.

2c. If the Imported Project has no dependencies list

Step 3: Verify Installation

Note: This may take 5-10 minutes on first run.

```
# Run the setup verification script
uv run python scripts/00_setup.py
```

Configuring Visual Studio Code

Step 1: Open the Project

1. Launch VS Code
2. File → Open Folder
3. Select the course repository folder

Step 2: Select Python Interpreter

1. Open Command Palette (Ctrl+Shift+P on Windows/Linux, Cmd+Shift+P on macOS)
2. Type "Python: Select Interpreter"
3. Choose the interpreter from `.venv` (it should show up as `./venv/bin/python` or similar)

If the .venv interpreter doesn't appear:

- Click "Enter interpreter path..."
- Navigate to: `.venv/Scripts/python.exe` (Windows) or `.venv/bin/python` (Mac/Linux)

Step 3: Configure Jupyter

When you open a `.ipynb` file:

1. VS Code will prompt you to select a kernel
2. Choose the same `.venv` Python interpreter

Step 4: Test in VS Code

1. Open `scripts/00_setup.py`
2. You should see the Python version in the bottom-right status bar
3. Right-click in the editor → "Run Python File in Terminal"

Verifying Your Setup

Quick Verification

```
# Activate environment and run verification script
uv run python scripts/00_setup.py
```

Manual Verification

You can also test manually:

```
# Check Python version
uv run python --version

# Test package imports
uv run python -c "import pandas; import numpy; import matplotlib; print('All packages loaded

# List installed packages
uv pip list
```

Troubleshooting

Issue: `uv: command not found`

Solution: The installation directory isn't in your PATH.

Windows:
Add `%USERPROFILE%\.cargo\bin` to your PATH environment variable.

macOS/Linux:
Add this to your `~/.bashrc`, `~/.zshrc`, or equivalent:

```
export PATH="$HOME/.cargo/bin:$PATH"
```

Then run: `source ~/.bashrc` (or `~/.zshrc`)

Issue: "Python version X.Y required but X.Z found"

Solution: uv can install Python for you:

```
uv python install 3.11
uv sync
```

Issue: Package installation fails with compilation errors

Solution 1: Ensure you have build tools installed

Windows:

- Install Visual Studio Build Tools

macOS:

```
xcode-select --install
```

Linux (Ubuntu/Debian):

```
sudo apt-get install build-essential python3-dev
```

Solution 2: Try installing pre-built wheels:

```
uv pip install --only-binary :all: [package-name]
```

Issue: VS Code doesn't recognize the .venv interpreter

Solution:

1. Make sure `.venv` folder exists in your project root
2. Reload VS Code (Ctrl+Shift+P → "Developer: Reload Window")
3. Manually browse to the interpreter:

- `Windows: `.venv\Scripts\python.exe``
- `Mac/Linux: `.venv/bin/python``

Issue: Jupyter notebooks won't run

Solution:

1. Ensure Jupyter extension is installed
2. Install ipykernel in your environment:

```
uv pip install ipykernel
```

1. Select the `.venv` kernel when opening a notebook
2. Restart VS Code if needed

Issue: "Import could not be resolved" warnings in VS Code

Solution:

1. Ensure you've selected the correct Python interpreter
2. Check that `python.analysis.extraPaths` includes your project root
3. Reload VS Code window

Issue: Cannot write to `.venv` directory (permission errors)

Solution:

- Don't use `sudo` with uv commands
- Ensure you have write permissions to the project directory
- On Windows, run terminal as regular user (not administrator)

Issue: `uv sync` is very slow

Possible causes:

- First run always takes longer
- Slow internet connection
- Antivirus scanning

Solutions:

- Be patient on first run
- Disable antivirus temporarily for `.venv` folder
- Use `uv sync --offline` if you've run it before

Best Practices for This Course

File Organization

- Keep all work within this project folder
- Use relative paths (or `pathlib`)
- Save scripts with descriptive names: `verb_noun.py` (e.g., `analyze_sales.py`)

Naming Conventions

Good:

- `clean_data.py`
- `2024_01_15_sales_report.csv`
- `figure_01_scatterplot.png`

Bad:

- `script1.py`
- `final FINAL (2) copy.py`
- `sales data.csv` (spaces can be problematic)

Running Python Code

In Terminal:

```
# Always use uv run to ensure correct environment
uv run python scripts
```