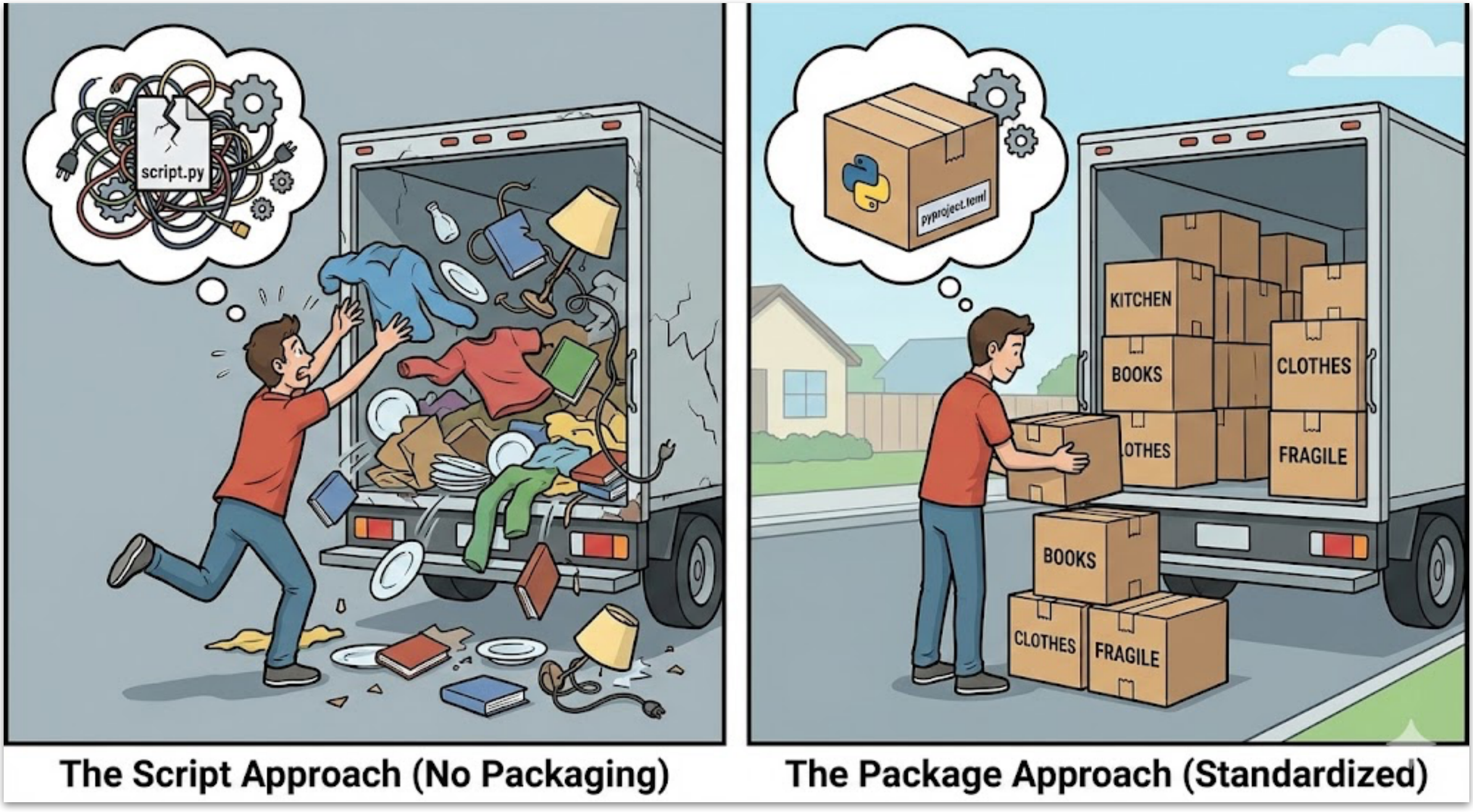
Lecture 7: Packaging and Automation

CEE 690

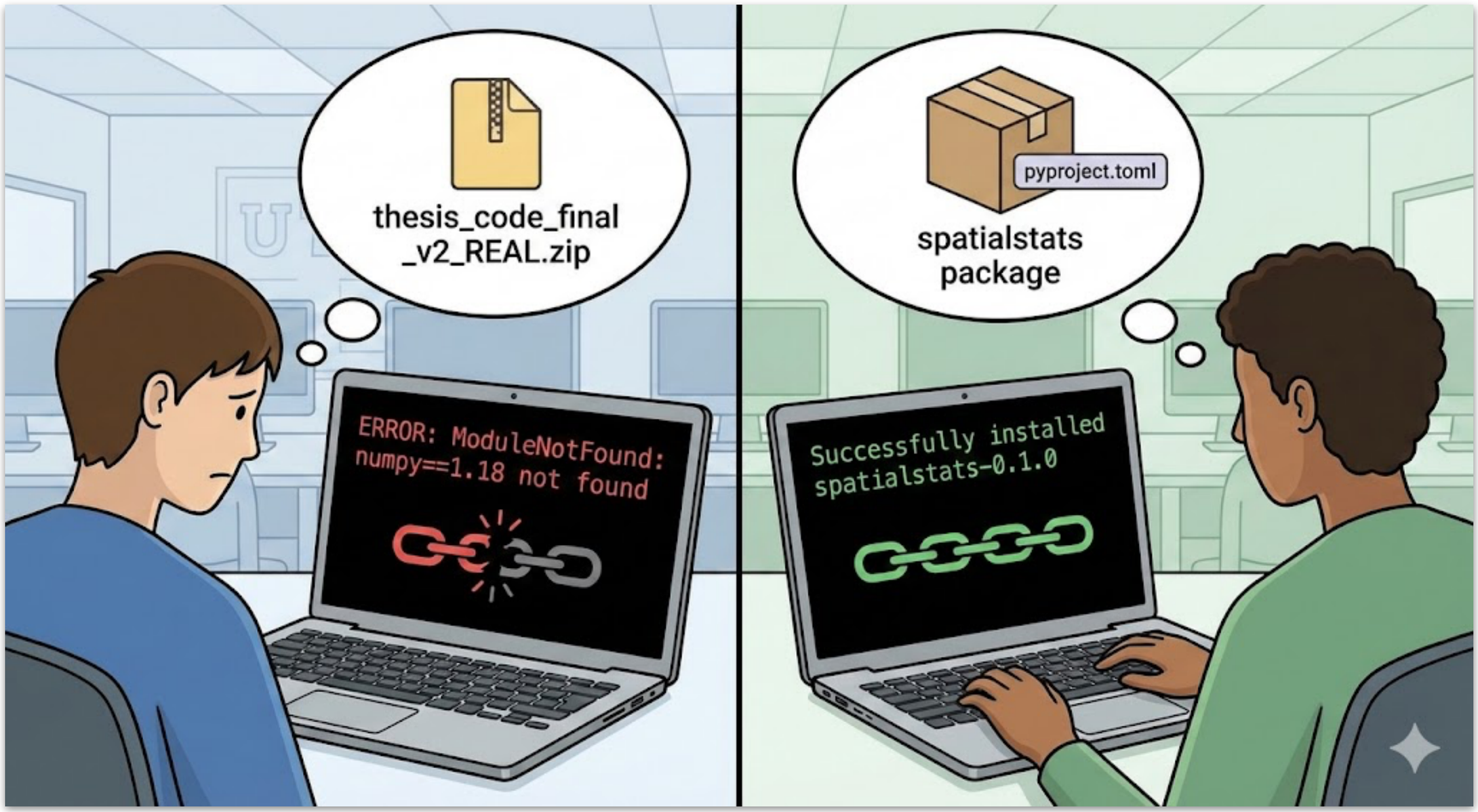
Packaging

What is packaging?



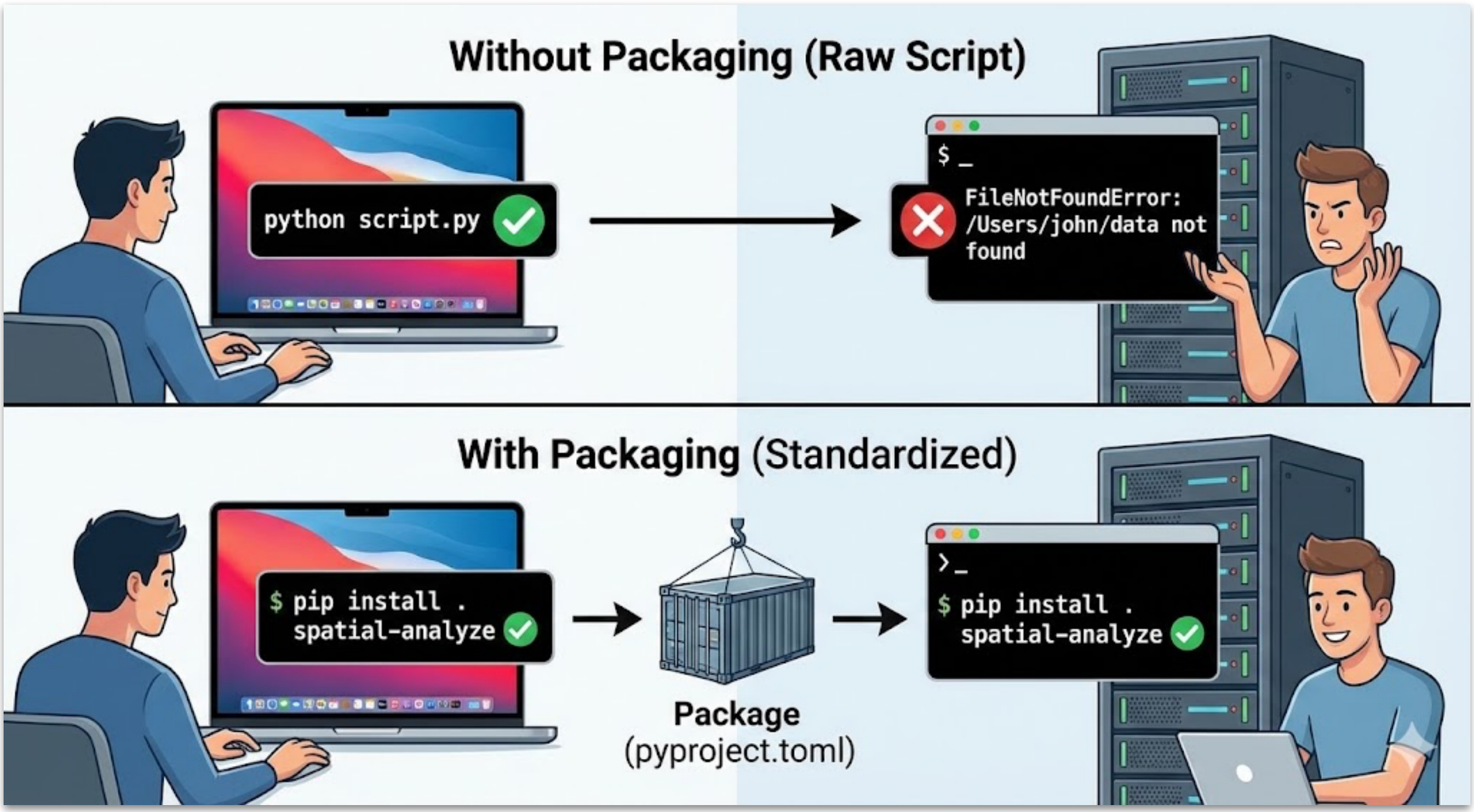
Source: Gemini

Reason 1: Reproducibility



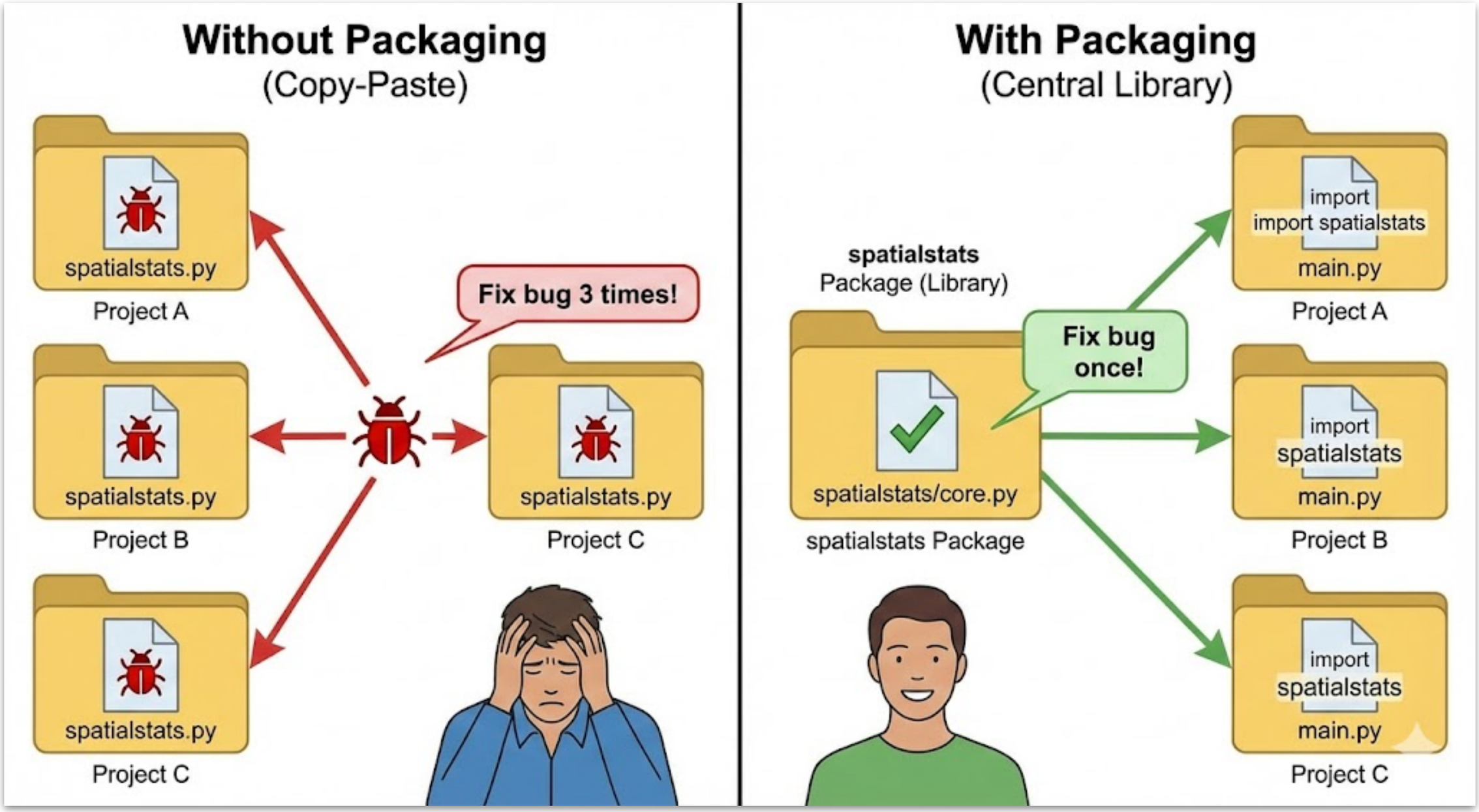
Source: Gemini

Reason 2: Portability



Source: Gemini

Reason 3: Usability



Source: Gemini

Getting ready for packaging

```
(cee690) nc153 $ tree -L 3
.
├── docs
│   ├── _build
│   │   ├── doctrees
│   │   └── html
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── Makefile
├── era_interim_annual_197901_201512_upscaled_subset.nc
├── example.json
├── README.md
└── spatialstats.py
```

```
(cee690) nc153 $ tree -L 3
.
├── data
│   ├── era_interim_annual_197901_201512_upscaled_subset.nc
│   └── example.json
├── docs
│   ├── _build
│   │   ├── doctrees
│   │   └── html
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── Makefile
├── Makefile
├── pyproject.toml
├── README.md
├── src
│   └── spatialstats
│       ├── core.py
│       └── __init__.py
└── tests
    └── test_spatialstats.py
```

Most of this is just organizing it clearly, however, there are few things that we need to discuss

Getting ready for packaging

```
(cee690) nc153 $ tree -L 3
.
├── docs
│   ├── _build
│   │   ├── doctrees
│   │   └── html
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── Makefile
├── era_interim_annual_197901_201512_upscaled_subset.nc
├── example.json
├── README.md
└── spatialstats.py
```

```
(cee690) nc153 $ tree -L 3
.
├── data
│   ├── era_interim_annual_197901_201512_upscaled_subset.nc
│   └── example.json
├── docs
│   ├── _build
│   │   ├── doctrees
│   │   └── html
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── Makefile
├── Makefile
├── pyproject.toml
├── README.md
├── src
│   └── spatialstats
│       ├── core.py
│       └── __init__.py
└── tests
    └── test_spatialstats.py
```

Most of this is just organizing it clearly, however, there are few things that we need to discuss

Getting ready for packaging

```
(cee690) nc153 $ tree -L 3
.
├── docs
│   ├── _build
│   │   ├── doctrees
│   │   └── html
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── Makefile
├── era_interim_annual_197901_201512_upscaled_subset.nc
├── example.json
├── README.md
└── spatialstats.py
```

```
(cee690) nc153 $ tree -L 3
.
├── data
│   ├── era_interim_annual_197901_201512_upscaled_subset.nc
│   └── example.json
├── docs
│   ├── _build
│   │   ├── doctrees
│   │   └── html
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── Makefile
├── Makefile
├── pyproject.toml
├── README.md
├── src
│   └── spatialstats
│       ├── core.py
│       └── __init__.py
└── tests
    └── test_spatialstats.py
```

Most of this is just organizing it clearly, however, there are few things that we need to discuss

Pyproject.toml

This is what tells pip “what to do.

```toml
[build-system]
requires = ["setuptools>=61.0"]
build-backend = "setuptools.build_meta"

[project]
name = "spatialstats"
version = "0.1.0"
description = "A package for spatial statistics on NetCDF data"
authors = [{name = "Nate", email = "nc153@duke.edu"}] # Updated example
dependencies = [
    "numpy",
    "netCDF4",
    "matplotlib"
]

# This maps the command "spatialstats" to the main() function in core.py
[project.scripts]
spatialstats = "spatialstats.core:main"

[tool.setuptools.packages.find]
where = ["src"]
```

Installing package

```
(cee690) nc153 $ pip install -e .
Obtaining file:///hpc/home/nc153/spatialstats
  Installing build dependencies ... done
  Checking if build backend supports build_editable ... done
  Getting requirements to build editable ... done
  Preparing editable metadata (pyproject.toml) ... done
Requirement already satisfied: numpy in /hpc/home/nc153/miniconda3/
0) (2.4.1)
Requirement already satisfied: netCDF4 in /hpc/home/nc153/miniconda
1.0) (1.7.4)
Requirement already satisfied: matplotlib in /hpc/home/nc153/minicc
=0.1.0) (3.10.8)
Requirement already satisfied: contourpy>=1.0.1 in /hpc/home/nc153/
lib->spatialstats==0.1.0) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /hpc/home/nc153/mini
>spatialstats==0.1.0) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /hpc/home/nc153
tlib->spatialstats==0.1.0) (4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /hpc/home/nc153
tlib->spatialstats==0.1.0) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /hpc/home/nc153/m
ib->spatialstats==0.1.0) (25.0)
Requirement already satisfied: pillow>=8 in /hpc/home/nc153/minicor
atialstats==0.1.0) (12.1.0)
Requirement already satisfied: pyparsing>=3 in /hpc/home/nc153/mini
```

No need to carry the script around anymore or make sure it is in a certain directory. It is now part of your conda environment.

```
(cee690) nc153 $ python
Python 3.14.2 | packaged by conda-forge | (main, Dec  6 2025, 11:21:58) [GCC 14.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import spatialstats
>>>
```

Pip install . Vs. Pip install -e .

Feature	pip install .	pip install -e .
Full Command	Regular Install	Editable (Developer) Install
Mechanism	Copies files to site-packages	Creates a link to your source folder
Updates	Requires reinstall	Instant
Speed	Slower (builds wheel)	Fast (just links)
Use Case	Production / Final Release	Active Coding / Debugging

Two valid options…

```
(cee690) nc153 $ tree -L 3
.
├── data
│   ├── era_interim_annual_197901_201512_upscaled_subset.nc
│   └── example.json
├── docs
│   ├── _build
│   │   ├── doctrees
│   │   └── html
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── Makefile
├── Makefile
├── pyproject.toml
├── README.md
├── src
│   └── spatialstats
│       ├── core.py
│       └── __init__.py
└── tests
    └── test_spatialstats.py
```

```
(cee690) nc153 $ tree -L 3
.
├── data
│   ├── era_interim_annual_197901_201512_upscaled_subset.nc
│   └── example.json
├── docs
│   ├── _build
│   │   ├── doctrees
│   │   └── html
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── Makefile
├── Makefile
├── pyproject.toml
├── README.md
├── src
│   └── spatialstats.py
└── tests
    └── test_spatialstats.py
```

Both options are ok. Why is the first option preferred?

Two valid options...

```
(cee690) nc153 $ tree -L 3
.
├── data
│   ├── era_interim_annual_197901_201512_upscaled_subset.nc
│   └── example.json
├── docs
│   ├── _build
│   │   ├── doctrees
│   │   └── html
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── Makefile
├── Makefile
├── pyproject.toml
├── README.md
├── src
│   └── spatialstats
│       ├── core.py
│       └── __init__.py
└── tests
    └── test_spatialstats.py
```

```
(cee690) nc153 $ tree -L 3
.
├── data
│   ├── era_interim_annual_197901_201512_upscaled_subset.nc
│   └── example.json
├── docs
│   ├── _build
│   │   ├── doctrees
│   │   └── html
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── Makefile
├── Makefile
├── pyproject.toml
├── README.md
├── src
│   └── spatialstats.py
└── tests
    └── test_spatialstats.py
```

Both options are ok. Why is the first option preferred?

Two valid options...

```
(cee690) nc153 $ tree -L 3
.
├── data
│   ├── era_interim_annual_197901_201512_upscaled_subset.nc
│   └── example.json
├── docs
│   ├── _build
│   │   ├── doctrees
│   │   └── html
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── Makefile
├── Makefile
├── pyproject.toml
├── README.md
├── src
│   └── spatialstats
│       ├── core.py
│       └── __init__.py
└── tests
    └── test_spatialstats.py
```

```
(cee690) nc153 $ tree -L 3
.
├── data
│   ├── era_interim_annual_197901_201512_upscaled_subset.nc
│   └── example.json
├── docs
│   ├── _build
│   │   ├── doctrees
│   │   └── html
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── Makefile
├── Makefile
├── pyproject.toml
├── README.md
├── src
│   └── spatialstats.py
└── tests
    └── test_spatialstats.py
```

Both options are ok. Why is the first option preferred?

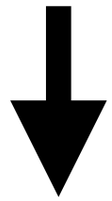
__init__.py

```
1 # src/spatialstats/__init__.py
2 from .core import SpatialAnalyzer,get_args
3
4
```

Why make our lives so complicated?

spatialstats executable

```
10 dependencies = [
11     "numpy",
12     "netCDF4",
13     "matplotlib"
14 ]
15
16 # This maps the command "spatialstats" to the main() function in core.py
17 [project.scripts]
18 spatialstats = "spatialstats.core:main"
```



```
(cee690) nc153 $ spatialstats --JSON_FILE spatialstats/data/example.json
Configuration loaded from spatialstats/data/example.json
Computing the statistics
Visualizing the data
Plot saved to output_plot.png
Saving statistics to output_temporal_stats.nc
Processing complete.
```

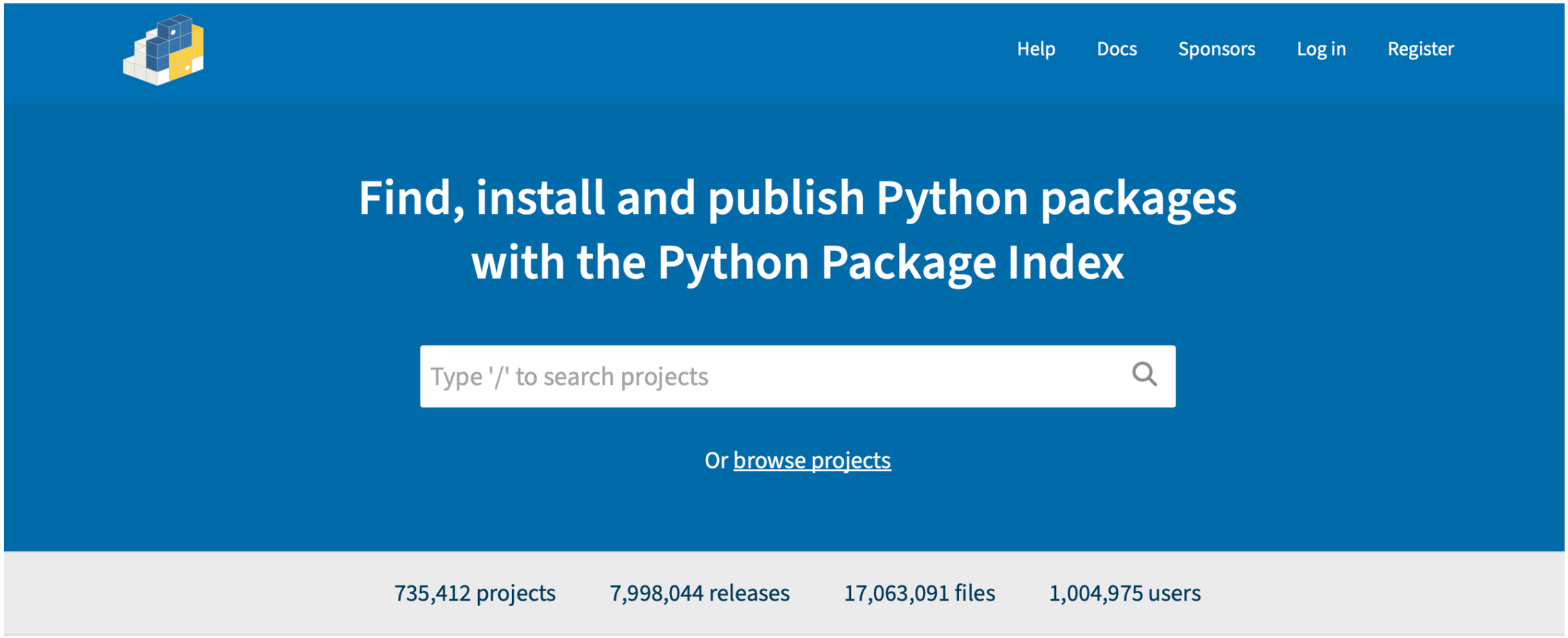
Makefile

```
.PHONY: install run clean

# Install the package in "editable" mode (-e)
# This means changes in src/ apply immediately without reinstalling
install:
	pip install -e .

# Run the tool using the new command we created in pyproject.toml
# Note: We point to the data in the new data/ folder
run:
	spatialstats --INPUT_FILE data/era_interim_annual_197901_201512_upscaled_subset.nc

clean:
	rm -rf build dist src/*.egg-info
	find . -type d -name "__pycache__" -exec rm -rf {} +
	find . -type f -name "*.pyc" -delete
```

```
(cee690) nc153 $ make install
pip install -e .
Obtaining file:///hpc/home/nc153/spatialstats
  Installing build dependencies ... done
  Checking if build backend supports build_edi
  Getting requirements to build editable ... d
  Preparing editable metadata (pyproject.toml)
Requirement already satisfied: numpy in /hpc/h
0) (2.4.1)
Requirement already satisfied: netCDF4 in /hpc
```

```
(cee690) nc153 $ make clean
rm -rf build dist src/*.egg-info
find . -type d -name "__pycache__" -exec rm -r
find . -type f -name "*.pyc" -delete
```

Ready for Pypi



It’s better to avoid adding anything to Pypi. Use GitHub instead

```
(cee690) nc153 $ pip install git+https://github.com/chaneyn/spatialstats.git
```

Automation

Can something/someone else run all checks for me?

Continuous integration

Before code is allowed to be merged on the main branch, it should pass these tests:

Continuous integration

Before code is allowed to be merged on the main branch, it should pass these tests:

1. **Linting** - Is the code clean?

Continuous integration

Before code is allowed to be merged on the main branch, it should pass these tests:

1. **Linting** - Is the code clean?

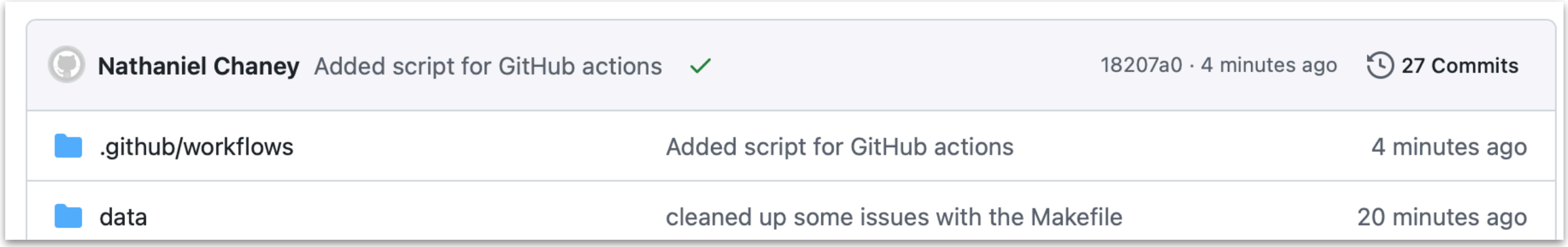
2. **Unit Tests** - Does the math work?

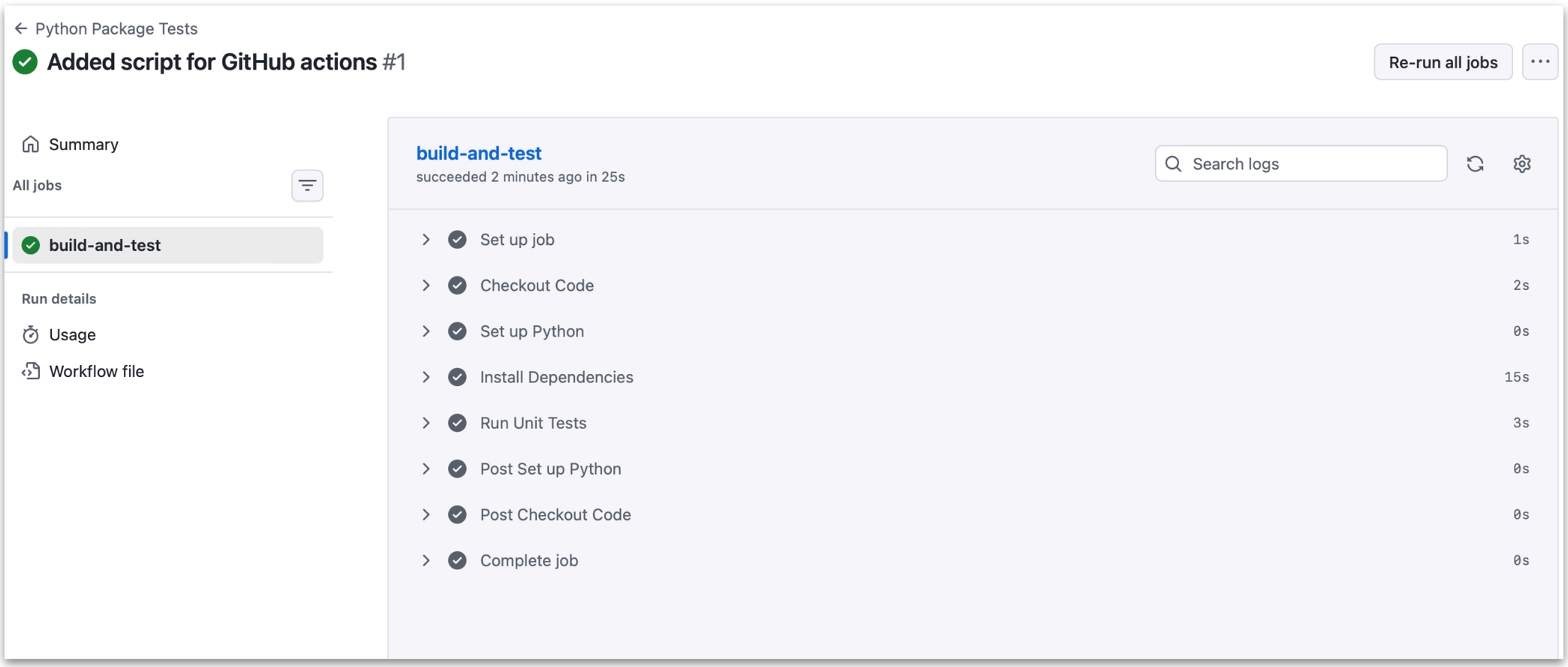
Continuous integration

Before code is allowed to be merged on the main branch, it should pass these tests:

1. **Linting** - Is the code clean?
2. **Unit Tests** - Does the math work?
3. **Installation** - Does pip install actually work?

Automation: GitHub Actions





Example for spatialstats

```
(cee690) nc153 $ vi .github/workflows/tests.yml
```

```yaml
name: Python Package Tests

on: [push, pull_request]

jobs:
  build-and-test:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Code
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.10'

    - name: Install Dependencies
      run: |
        python -m pip install --upgrade pip
        pip install .[test]  # Installs the package + t

    - name: Run Unit Tests
      run: |
        python -m unittest discover tests
```