📦 Packaging AI Projects

Generative AI Bootcamp – Week 1, Day 4, Session 1

November 20, 2025

Learning Objectives

- Understand Python project structure and metadata
- Explore `pyproject.toml` and modern packaging standards
- Learn dependency management and semantic versioning
- Prepare AI projects for reuse and CI/CD integration

Why Packaging Matters

- Ensures reproducibility and dependency consistency
- Enables modular reuse across services
- Simplifies deployment (pip installable)
- Supports continuous integration pipelines

Semantic Versioning

- **MAJOR.MINOR.PATCH** (e.g., 1.3.5)
- Increment based on:
  - MAJOR → breaking changes
  - MINOR → new features (backward compatible)
  - PATCH → bug fixes

Standard Project Layout

```
my_ai_project/
├── pyproject.toml
├── README.md
├── src/
│   └── my_ai_project/
│       ├── __init__.py
│       ├── pipeline.py
│       ├── models/
│       └── utils/
├── tests/
│   └── test_pipeline.py
└── requirements.txt
```

`pyproject.toml` Basics

```toml
[project]
name = "my-ai-project"
version = "0.1.0"
description = "Reusable AI pipeline components"
authors = [{name="AI Bootcamp", email="ai@bootcamp.org"}]
dependencies = ["fastapi", "pydantic", "requests"]
requires-python = ">=3.10"

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

`pyproject.toml` Dependency Versioning

- comma-separated bounds, operators `==`, `>=`, `<`, `!=`
- `~=1.2.3` is equivalent to `>=1.2.3, <2`, or `>=1.2.3, ==1.*`
- conditional dependencies:
  - `backports.ssl_match_hostname ~= 3.5; python_version < "3.5"`
  - `colorama ~= 0.4; sys_platform == "win32"`

`pyproject.toml` Build Systems

Frontend (CLI) and backend. Examples:

- `setuptools`, `setuptools.build_meta`
- `poetry`, `poetry-core` (different `pyproject.toml` conventions)
- Flit, Hatch, PDM…

Dependency Management Tools

Tool	Description
`pip`	Base installer
`pip-tools`	Dependency pinning
`poetry`	Full-featured dependency + packaging
`uv`	Fast modern alternative
`conda`	Handles non-Python dependencies, useful in data science and scientific computing

Packaging Best Practices

- Use `src/` layout for clear imports
- Include `__init__.py` in all modules
- Keep dependencies minimal and pinned
- Add `__version__` in your main package
- Document install instructions in README.md

Building

- When developing, install in "development (or *editable*) mode" with `pip install -e .`
- Once done, test a regular install with `pip install .`
- Then, build your package with `python -m build`
- …and install from the wheel file with `pip install dist/my_ai_project-0.1.0-py3-none-any.whl`

Takeaways

- Proper packaging ensures scalability and maintainability
- `pyproject.toml` replaces legacy `setup.py`
- Essential step for CI/CD, MLOps, and deployment workflows