

# Python Project Setup & Best Practices

## Complete Project Workflow

1. Create Project Directory
2. Initialize Git
3. Create Virtual Environment
4. Install Dependencies
5. Setup Project Structure
6. Configure .gitignore
7. Write README
8. Connect to GitHub
9. Start Development

## Quick Start Template

```
# 1. Create and enter project
mkdir my-project && cd my-project

# 2. Initialize UV + Git
uv init
git init

# 3. Add dependencies
uv add pandas numpy requests
uv add --dev pytest black ruff

# 4. Create structure
mkdir -p src tests data docs

# 5. Setup GitHub
```

```
git add .
git commit -m "Initial commit"
git remote add origin https://github.com/username/my-project.git
git push -u origin main
```

## Standard Project Structure

### Simple Project

```
my-project/
├── .git/
├── .venv/
├── .gitignore
├── README.md
├── pyproject.toml
├── uv.lock
├── src/
│   ├── __init__.py
│   └── main.py
└── tests/
    └── test_main.py
```

### Data Science Project

```
ml-project/
├── .git/
├── .venv/
├── .gitignore
├── README.md
├── pyproject.toml
├── uv.lock
├── notebooks/
│   ├── 01_exploration.ipynb
│   └── 02_modeling.ipynb
```

```
|— src/
|   |— __init__.py
|   |— data_loader.py
|   |— preprocessing.py
|   |— model.py
|— data/
|   |— raw/
|   |— processed/
|   |— sample.csv
|— models/
|   |— .gitkeep
|— tests/
|   |— test_preprocessing.py
|— configs/
|   |— config.yaml
```

## Web Application

```
web-app/
|— .git/
|— .venv/
|— .gitignore
|— README.md
|— pyproject.toml
|— uv.lock
|— app/
|   |— __init__.py
|   |— main.py
|   |— routes/
|   |— models/
|   |— templates/
|   |— static/
|— tests/
```

```
|— migrations/  
|— .env.example
```

## Essential Files

### **.gitignore (Comprehensive)**

```
# Virtual Environments  
.venv/  
venv/  
ENV/  
env/  
.python-version  
  
# Python  
__pycache__/  
*.py[cod]  
*$py.class  
*.so  
.Python  
*.egg-info/  
dist/  
build/  
*.egg  
  
# Jupyter  
.ipynb_checkpoints/  
*.ipynb_checkpoints  
  
# Data (adjust as needed)  
data/raw/*  
data/processed/*  
!data/sample.csv  
*.csv  
*.xlsx
```

```
*.json
*.pkl
*.h5
*.hdf5

# Models
models/*.pt
models/*.pth
models/*.h5
weights/
*.ckpt

# Secrets
.env
.env.local
secrets/
*.key
*.pem
config/credentials/

# IDE
.vscode/
.idea/
*.swp
*.swo
*~

# OS
.DS_Store
.DS_Store?
._*
Thumbs.db
Desktop.ini

# Logs
*.log
```

logs/

# Testing

.pytest\_cache/

.coverage

htmlcov/

.tox/

# Misc

\*.bak

\*.tmp

temp/

## README.md Template

# Project Name

Brief description (one paragraph)

## Features

- Feature 1
- Feature 2
- Feature 3

## Installation

### Prerequisites

- Python 3.10+
- UV package manager

### Setup

```bash

# Clone repository

git clone https://github.com/username/project.git

cd project

```
# Install dependencies
uv sync

# Run
uv run python src/main.py
```

## Usage

```
from src import main

# Example usage
result = main.process_data()
```

## Project Structure

```
src/      - Source code
tests/    - Unit tests
data/     - Data files (not in repo)
docs/     - Documentation
```

## Development

```
# Run tests
uv run pytest

# Format code
uv run black .

# Lint
uv run ruff check .
```

# Contributing

Pull requests welcome. Please:

1. Fork the repo
2. Create feature branch
3. Commit changes
4. Push to branch
5. Open PR

# License

MIT

```
### pyproject.toml (UV)
``toml
[project]
name = "my-project"
version = "0.1.0"
description = "Brief description"
readme = "README.md"
requires-python = ">=3.10"
dependencies = [
    "pandas>=2.0.0",
    "numpy>=1.24.0",
]

[project.optional-dependencies]
dev = [
    "pytest>=7.0.0",
    "black>=23.0.0",
    "ruff>=0.1.0",
]
```

```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"
```

## **.env.example**

```
# Copy to .env and fill in values
# NEVER commit .env

API_KEY=your_key_here
DATABASE_URL=postgresql://localhost/mydb
SECRET_KEY=generate_random_string
DEBUG=True
```

# **Development Best Practices**

## **Code Organization**

```
# Good: Modular structure
src/
├── data/
│   ├── loader.py    # Data loading
│   └── cleaner.py   # Data cleaning
├── models/
│   └── predictor.py # ML models
└── utils/
    └── helpers.py   # Utilities

# Bad: Everything in one file
src/
└── main.py          # 2000 lines of code
```

## **Naming Conventions**

```
# Files & Modules
my_module.py      # lowercase_with_underscores

# Classes
class DataProcessor # PascalCase

# Functions & Variables
def load_data()    # lowercase_with_underscores
user_count = 10

# Constants
MAX_RETRIES = 3    # UPPERCASE_WITH_UNDERSCORES

# Private
_internal_func()   # Leading underscore
__private_var      # Double underscore
```

## Import Organization

```
# Standard library
import os
import sys
from pathlib import Path

# Third-party
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Local
from src.data import loader
from src.models import predictor
```

## Documentation

```
def process_data(df: pd.DataFrame, threshold: float = 0.5) → pd.DataFrame:
    """
    Process dataframe by filtering and transforming.

    Args:
        df: Input dataframe
        threshold: Filtering threshold (default: 0.5)

    Returns:
        Processed dataframe

    Raises:
        ValueError: If df is empty
    """
    if df.empty:
        raise ValueError("DataFrame cannot be empty")

    return df[df['score'] > threshold]
```

## Testing Setup

### Basic Test Structure

```
# tests/test_processor.py
import pytest
from src.processor import process_data

def test_process_data():
    # Arrange
    data = create_test_data()

    # Act
    result = process_data(data)
```

```
# Assert
assert len(result) > 0
assert result['score'].min() > 0.5

def test_process_empty_raises():
    with pytest.raises(ValueError):
        process_data(pd.DataFrame())
```

## Run Tests

```
# Run all tests
uv run pytest

# Run with coverage
uv run pytest --cov=src

# Run specific test
uv run pytest tests/test_processor.py

# Verbose output
uv run pytest -v
```

## Code Quality Tools

### Black (Formatter)

```
# Format all files
uv run black .

# Check without modifying
uv run black --check .

# Format specific file
uv run black src/main.py
```

## Ruff (Linter)

```
# Lint all files
uv run ruff check .

# Auto-fix issues
uv run ruff check --fix .

# Check specific file
uv run ruff check src/main.py
```

## Pre-commit Setup

```
# .pre-commit-config.yaml
repos:
  - repo: https://github.com/psf/black
    rev: 23.12.0
    hooks:
      - id: black

  - repo: https://github.com/astral-sh/ruff-pre-commit
    rev: v0.1.9
    hooks:
      - id: ruff
        args: [--fix]
```

## Configuration Files

### pytest.ini

```
[pytest]
testpaths = tests
python_files = test_*.py
python_classes = Test*
```

```
python_functions = test_*
addopts = -v --tb=short
```

## **ruff.toml**

```
[lint]
select = ["E", "F", "I"]
ignore = ["E501"] # Line too long

[lint.per-file-ignores]
"__init__.py" = ["F401"] # Unused imports
```

# **Environment Management**

## **Multiple Environments**

```
# Development
uv sync

# Production (no dev deps)
uv sync --no-dev

# Specific Python version
uv venv --python 3.10
```

## **Requirements Files (Legacy Support)**

```
# Export from UV
uv pip freeze > requirements.txt

# Split dev/prod
uv pip freeze > requirements-dev.txt
# Manually create requirements.txt with prod only
```

## CI/CD with GitHub Actions

```
# .github/workflows/test.yml
name: Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Install UV
        run: curl -LsSf https://astral.sh/uv/install.sh | sh

      - name: Install dependencies
        run: uv sync

      - name: Run tests
        run: uv run pytest

      - name: Lint
        run: uv run ruff check .
```

## Common Project Types

### CLI Application

```
# src/cli.py
import argparse

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('input', help='Input file')
```

```

parser.add_argument('--output', help='Output file')
args = parser.parse_args()

# Process
print(f"Processing {args.input}")

if __name__ == '__main__':
    main()

```

## Data Pipeline

```

# src/pipeline.py
class DataPipeline:
    def load(self, path):
        return pd.read_csv(path)

    def clean(self, df):
        return df.dropna()

    def transform(self, df):
        return df.apply(lambda x: x * 2)

    def run(self, input_path, output_path):
        df = self.load(input_path)
        df = self.clean(df)
        df = self.transform(df)
        df.to_csv(output_path)

```

## API Server

```

# src/app.py
from fastapi import FastAPI

app = FastAPI()

```

```

@app.get("/")
def read_root():
    return {"message": "Hello World"}

@app.get("/predict")
def predict(data: dict):
    result = model.predict(data)
    return {"prediction": result}

```

## Deployment Checklist

Before deploying:

- ☐ All tests pass
- ☐ No secrets in code
- ☐ .env.example provided
- ☐ README updated
- ☐ Dependencies locked (uv.lock)
- ☐ .gitignore configured
- ☐ License added
- ☐ Version tagged
- ☐ Documentation complete

## Quick Reference

| Task           | Command                                               |
|----------------|-------------------------------------------------------|
| New project    | <code>uv init my-project</code>                       |
| Add dependency | <code>uv add package</code>                           |
| Run script     | <code>uv run python script.py</code>                  |
| Run tests      | <code>uv run pytest</code>                            |
| Format code    | <code>uv run black .</code>                           |
| Lint code      | <code>uv run ruff check .</code>                      |
| Git commit     | <code>git add . &amp;&amp; git commit -m "msg"</code> |

| Task         | Command               |
|--------------|-----------------------|
| Push changes | <code>git push</code> |

## Tips

- **Start with structure** - Setup files before coding
- **Test early** - Write tests as you code
- **Document as you go** - Update README with changes
- **Small commits** - Commit working features incrementally
- **Use branches** - Feature branches for new work
- **Review .gitignore** - Never commit secrets or large files
- **Pin versions** - Lock files ensure reproducibility
- **Automate checks** - Use pre-commit hooks or CI/CD