Here's a **structured Python testing roadmap** tailored to your profile and projects (Django apps, ML models, pipelines, automation scripts). I'll break it down step by step so you know **what to implement first and how to progress**:

Python Testing Roadmap for Your Projects

1️⃣ Unit Testing (Foundational)

Goal: Test the smallest pieces of code (functions, methods) in isolation.

- **When to do:** Always, for core logic and functions.
- **Tools:** `unittest` (built-in), `pytest`.
- **Focus areas for you:**
  - Functions in ML preprocessing scripts.
  - Core methods in your Django models or utilities.
  - Small Python automation functions (like reading/writing Excel, parsing CSV/JSON).
- **Example:**

```
# test_utils.py
import pytest
from data_utils import clean_text

def test_clean_text():
    assert clean_text(" Hello ") == "Hello"
    assert clean_text("") == ""
```

2️⃣ Integration Testing

Goal: Test how multiple components work together (DB + models, API + logic, pipeline steps).

- **When to do:** After unit tests are solid.
- **Tools:** `pytest`, `pytest-django` for Django apps.
- **Focus areas:**
  - Django forms and models together.
  - End-to-end ML pipelines (data ingestion → preprocessing → prediction).
  - APIs that call internal functions.
- **Example in Django:**

```
from django.test import TestCase
from restaurant.models import Menu

class MenuIntegrationTest(TestCase):
    def test_menu_creation(self):
        menu = Menu.objects.create(name="Salad", price=10,
description="Fresh salad")
        self.assertEqual(Menu.objects.count(), 1)
```

3 Functional / End-to-End Testing

Goal: Simulate user behavior or system-level scenarios.

- **When to do:** Once integration tests are stable.
- **Tools:**
  - Web: `Selenium`, `pytest-selenium`
  - APIs: `requests`, `pytest`
- **Focus areas:**
  - Django app pages: form submission, login, menu navigation.
  - Automated bots/scripts: simulate real workflow.
  - ML web apps: user uploads CSV → prediction results.
- **Example (Selenium):**

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("http://127.0.0.1:8000/menu/")
assert "Menu" in driver.title
```

```
driver.quit()
```

4️⃣ Automated Testing in CI/CD

Goal: Run all tests automatically on every commit or pull request.

- **When to do:** Once your test suite is ready.
- **Tools/Platforms:**
  - GitHub Actions, GitLab CI, Jenkins, CircleCI.
  - Combine unit + integration + functional tests.
- **Focus areas for you:**
  - Run `pytest` for all Python scripts and Django apps.
  - Run API and ML model tests automatically.
  - Generate coverage reports.
- **Example GitHub Actions workflow:**

```
name: Python CI

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: 3.10
      - name: Install dependencies
        run: pip install -r requirements.txt
      - name: Run tests
        run: pytest --cov=.
```

5️⃣ Coverage and Best Practices

- **Measure:** `pytest --cov` or `coverage.py`.
- **Best Practices:**
    - Start **unit tests** for all core functions first.
    - Gradually add **integration tests** for models, pipelines, APIs.
    - Include **functional/UI tests** only for high-value flows.
    - Automate everything with **CI/CD**.
    - Keep tests **isolated, deterministic, and fast**.
    - Test edge cases and failures, not just happy paths.

☑️ Suggested Order for You

1. Unit tests for Python scripts, ML preprocessing, utilities.
2. Unit + integration tests for Django models, forms, and views.
3. Functional tests for web apps (Django + ML apps).
4. Add automated tests to CI/CD pipelines for continuous verification.
5. Optionally, expand to performance testing or load testing once your apps are production-ready.