Why Unit Test

◇ 1. Testing before deploy ≠ Unit testing

When you test “before deploy” (manually running your app, or running integration checks inside a container), you’re mostly checking that the **whole system works together**.

That’s **integration testing**, not **unit testing**.

- **Integration tests** = “Does everything work end-to-end?”
- **Unit tests** = “Does this tiny piece of logic behave correctly in all cases?”

If you skip unit tests:

- You might miss subtle bugs in edge cases (e.g., your `get_weather` function crashing on a string `"20C"`).
- Fixing bugs becomes harder because you don’t know *where* the failure comes from.

◇ 2. Containers solve environment problems, not logic problems

- Docker/Kubernetes ensure *“it runs the same way everywhere”*.
- But they can’t tell you if your function divides by zero, returns the wrong output, or mishandles a null value.
- Example: your app may run fine inside the container but still give the wrong result to users because of a bug in your business logic.

◇ 3. Tests are documentation and guardrails

- Unit tests **document how your code is supposed to behave**.
- When someone else (or future-you) changes the code, the tests will scream if behavior changes unexpectedly.
- Without tests, you’re blind when refactoring.

🔹 4. Bugs are cheaper to fix earlier

- A bug found at **unit level** = 2 minutes to fix.
- A bug found at **integration level** = hours (you need to trace the whole system).
- A bug found by a **user in production** = $$$ + reputation damage.

🔹 5. CI/CD + automation loves unit tests

In modern pipelines:

- Unit tests run automatically on every commit (GitHub Actions, GitLab CI, Jenkins).
- If one fails, the code won't even be merged/deployed.
- This saves you from shipping broken logic that "looked fine locally".

⚖️ So, why both?

Think of it like **layers of defense**:

- **Unit tests** → catch logic bugs early.
- **Integration/system tests** → ensure components play well together.
- **Containerization** → ensure it runs everywhere consistently.

They complement each other. Skipping unit tests is like saying:

"I won't check if the bricks are solid, because the whole house looks fine from the outside."

✅ **Recommendation for you (Python dev + DS/ML focus):**

- Don't obsess about testing every single tiny function.

- Focus on **core logic units** (data processing functions, model training pipelines, API endpoints).
- Learn **pytest** — it's the standard, clean, and powerful.

👉 Want me to sketch out a **practical roadmap** of "what tests are worth writing" for a project like your Django + ML apps, so you don't waste time over-testing?