# 📅 DAY 07 — Virtual Environments & `pip`

**Goal : Mange Python libraries safely**

---

## 1️⃣ The Problem (Why We Need Environments)

Imagine this :

- Project A needs `numpy==1.21`
- Project B needs `numpy==2.0`

If you install both globally❌ → conflict, errors, broken projects.

---

## 2️⃣ What Is a Virtual Environment? (Simple)

A **virtual environment** is:

- An **isolated Python workspace**
- Each project gets its **own libraries**.
- Nothing affects your system Python

👉 One project = one environment

---

## 3️⃣ Creating a Virtual Environment

from your project root:

```
python -m venv venv
```

This creates:

```
venv/
```

---

## 4️⃣ Activating the Environment

### On Windows

```
venv\Scripts\activate
```

### On macOS / Linux

```
source venv/bin/activate
```

You'll see:

```
(venv)
```

in your terminal → ✅ environment is active.

---

## 5️⃣ Installing Packages with `pip`

Once activated:

```
pip install numpy pandas
```

✔️ Installed

✔️ Safe

✔️ Reproducible

Check installed packages:

```
pip list
```

---

## 6️⃣ Freezing Dependencies (VERY IMPORTANT)

This creates a list of exact versions.

```
pip freeze > requirements.txt
```

Examples:

```
numpy==1.26.4
pandas==2.1.0
```

Anyone can recreate your environment using:

```
pip install -r requirements.txt
```

📌 **This is mandatory for ML projects**

---

## 7️⃣ Why ML Engineers Care So Much

ML projects depend on:

- numpy
- pandas
- scikit-learn
- tensorflow / pytorch

Wrong versions = ❌ crashes, ❌ incompatibility.

Virtual environments prevent this.

---

## 8️⃣ What NOT to Do ❌

❌ Install ML libraries globally

❌ Forget to activate venv

❌ Commit `venv/` folder to Github

❌ No `requirements.txt`

---

## 9️⃣ `.gitignore`

Create `.gitignore` file:

```
venv/
__pycache__/
*.pyc
```

This keeps your repo clean.

---