Reproducibility & Environments — Follow-Along Guide

Week 9 · CS 203 · Software Tools and Techniques for AI

Prof. Nipun Batra · IIT Gandhinagar

Spring 2026

How to Use This Guide

- Open `reproducibility_followalong.sh` in your editor (left half of screen)
- Open a terminal (right half of screen)
- Copy-paste each command, one at a time
- Compare your output with the expected output shown here
- **Type it yourself** — that's how you learn

Legend: `$` = command to type (don't type the `$`). `>>` = expected output. Blue boxes = look at the projector slide.

Projector: Slides 2–4 — "Works on My Machine" + Why It Matters Look at the projector. Your friend gets `ImportError`. Three hours later, still debugging.

Act 1: The Problem — "It Works on My Machine" ~5 min

Create a simple ML project and try to share it:

```
$ mkdir -p ~/repro-demo && cd ~/repro-demo
$ mkdir movie-predictor && cd movie-predictor
```

Write a training script:

```
$ cat > train.py << 'PYEOF'
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

np.random.seed(42)
X = np.random.rand(200, 5)
y = (X[:, 0] + X[:, 2] > 1).astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
```

```
print(f"Accuracy: {model.score(X_test, y_test):.3f}")
PYEOF
$ python train.py
>> Accuracy: 0.850   (yours may differ!)
```

Now imagine emailing this to a friend. They get `ModuleNotFoundError: No module named 'sklearn'`. Even if they figure out the package is called `scikit-learn`, they might get a different version. **Your code is versioned (Git). Your environment is not.**

Projector: Slides 6–9 — Virtual Environments (rooms in a house) Look at the projector. Each project gets its own room with its own packages.

Act 2: Virtual Environments (venv) ~10 min

Create an isolated Python environment:

```
$ python -m venv .venv
$ ls -la .venv/
>> bin/  include/  lib/  pyvenv.cfg
```

Activate it:

```
$ source .venv/bin/activate
$ which python
>> /path/to/movie-predictor/.venv/bin/python
```

Tip

Your prompt shows `(.venv)` when the environment is active. Always check before installing packages!

See what's installed (almost nothing):

```
$ pip list
>> pip        xx.x
>> setuptools xx.x
```

Install what we need and save the list:

```
$ pip install scikit-learn numpy
$ python train.py
>> Accuracy: 0.850
$ pip freeze > requirements.txt
$ cat requirements.txt
>> joblib==1.4.2
>> numpy==1.26.4
>> scikit-learn==1.5.0
>> scipy==1.13.1
>> threadpoolctl==3.5.0
```

Now anyone can recreate your exact environment:

```
$ python -m venv .venv && source .venv/bin/activate
$ pip install -r requirements.txt
```

Projector: Slides 10–11 — Good vs Bad requirements.txt Look at the projector. Pinned versions = time capsule. Unpinned = ticking bomb.

Act 3: Version Pinning ~8 min

Bad requirements.txt (no versions):

```
numpy
scikit-learn
```

Good requirements.txt (pinned — what `pip freeze` gives you):

```
numpy==1.26.4
scikit-learn==1.5.0
```

Prove the environment is isolated:

```
$ deactivate
$ python -c "import sklearn" 2>/dev/null || echo "Not found!"
>> Not found!
$ source .venv/bin/activate
$ python -c "import sklearn; print(sklearn.__version__)"
>> 1.5.0
```

Prove we can recreate from scratch:

```
$ deactivate && rm -rf .venv
$ python -m venv .venv && source .venv/bin/activate
$ pip install -r requirements.txt
$ python train.py
>> Accuracy: 0.850   (same result!)
```

Warning

Never commit `.venv/` to Git. Commit `requirements.txt` instead. The venv folder is 50+ MB and platform-specific.

Projector: Slides 13–16 — Random Seeds Look at the projector. "Which result do you report?" Three runs, three answers.

Act 4: Reproducible Randomness (Seeds) ~8 min

Run the script three times — different results each time:

```
$ python train.py && python train.py && python train.py
>> Accuracy: 0.850
>> Accuracy: 0.825
>> Accuracy: 0.875
```

Fix it by adding seeds everywhere:

```
$ cat > train.py << 'PYEOF'
import random
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

SEED = 42
random.seed(SEED)
np.random.seed(SEED)

X = np.random.rand(200, 5)
y = (X[:, 0] + X[:, 2] > 1).astype(int)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=SEED
)
model = RandomForestClassifier(n_estimators=100, random_state=SEED)
model.fit(X_train, y_train)
print(f"Accuracy: {model.score(X_test, y_test):.3f}")
PYEOF
$ python train.py && python train.py && python train.py
>> Accuracy: 0.925
>> Accuracy: 0.925
>> Accuracy: 0.925
```

Tip

Seeds go in **two places**: (1) Global — `random.seed()`, `np.random.seed()`, `torch.manual_seed()`. (2) Per-function — `random_state=42` in sklearn calls.

Projector: Slide 18 — Virtual Environments Aren't Enough Look at the projector. venv handles Python packages. What about the OS, system libraries, Python version itself?

Act 5: Limits of venv — OS & System Dependencies ~3 min

venv + requirements.txt handles Python packages. But not:

- System libraries (libssl, libblas, libffi)
- Python version itself (3.8 vs 3.11)
- OS differences (Mac Accelerate vs Linux OpenBLAS)
- CUDA/GPU drivers

```
$ python --version
>> Python 3.x.x   (whatever YOUR system has --- your friend's may differ)
```

venv isolates Python packages. Docker isolates everything.

Projector: Slides 19–24 — Docker Concepts, Dockerfile, Commands Walk through the Docker concept slides: Image, Container, Dockerfile, Registry.

Act 6: Docker — Same Result Everywhere ~15 min

Create a Dockerfile:

```
$ cat > Dockerfile << 'EOF'
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY train.py .
CMD ["python", "train.py"]
EOF
$ cat > .dockerignore << 'EOF'
.venv/
__pycache__/
.git/
EOF
```

Build and run:

```
$ docker build -t movie-predictor .
$ docker run movie-predictor
>> Accuracy: 0.925
$ docker run movie-predictor
>> Accuracy: 0.925   (same --- on ANY machine with Docker)
```

Explore inside the container:

```
$ docker run -it movie-predictor /bin/bash
>> root@abc123:/app# python --version
```

```
>> Python 3.10.x
>> root@abc123:/app# exit
```

Tip

Copy `requirements.txt` before your code in the Dockerfile. Docker caches layers — if requirements don't change, it skips the slow `pip install` step.

Projector: Slide 25 — Docker Compose Look at the projector. Multiple services defined in one YAML file.

Act 7: Docker Compose — App + Service ~8 min

Create a prediction server and compose file:

```
$ cat > docker-compose.yml << 'EOF'
services:
  trainer:
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - ./output:/app/output
  predictor:
    build:
      context: .
      dockerfile: Dockerfile.server
    ports:
      - "8000:8000"
EOF
```

Start everything:

```
$ docker compose up --build -d
$ docker compose ps
$ curl -s http://localhost:8000 | python -m json.tool
>> {"prediction": 1, "features": [...]}
$ docker compose down
```

Tip

`docker-compose.yml` is version-controlled. Anyone can run `docker compose up` and get the exact same multi-service setup.

Projector: Slides 27–30 — Project Structure + Config Look at the projector. Directory tree, config.yaml, .gitignore examples.

Act 8: Project Structure & Best Practices ~5 min

Organize the project properly:

```
$ mkdir -p src data/raw data/processed models notebooks
$ mv train.py src/train.py
```

Create a config file (no hardcoded values!):

```
$ cat > config.yaml << 'EOF'
model:
  type: random_forest
  n_estimators: 100
  seed: 42
data:
  n_samples: 200
  test_size: 0.2
paths:
  model: models/model.pkl
EOF
```

Create `.env` for secrets, `.gitignore`, `Makefile`, and `README.md`:

```
$ echo "API_KEY=sk-abc123secret" > .env
$ cat > .gitignore << 'EOF'
.venv/
__pycache__/
data/raw/
models/*.pkl
.env
.DS_Store
EOF
$ cat > Makefile << 'EOF'
setup:
    python -m venv .venv
    . .venv/bin/activate && pip install -r requirements.txt
train:
    python src/train.py
docker:
    docker build -t movie-predictor . && docker run movie-predictor
EOF
```

Warning

Never commit `.env` files. They contain secrets (API keys, passwords). Add `.env` to `.gitignore` and use a `.env.example` template instead.

Projector: Slides 31–32 — Reproducibility Checklist Walk through the checklist on the projector while running it live.

Act 9: Reproducibility Checklist — Verify & Share ~5 min

Run the checklist for our project:

```
$ echo "=== REPRODUCIBILITY CHECKLIST ==="
```

Check	Status
Virtual environment (`.venv/`)	Yes
`requirements.txt` with pinned versions (`==`)	Yes
Random seeds (`random_state`, `np.random.seed`)	Yes
`README.md` with setup instructions	Yes
`config.yaml` (no hardcoded values)	Yes
`.gitignore` (exclude data, models, secrets)	Yes
Dockerfile (optional, for full isolation)	Yes

The ultimate test: Can someone else clone your repo and get the exact same results?

1. Push to GitHub
2. Friend clones
3. `make setup && source .venv/bin/activate && make train`
4. Same accuracy? **You're reproducible!**

Quick Reference

I want to…	Command
Create virtual environment	`python -m venv .venv`
Activate (Mac/Linux)	`source .venv/bin/activate`
Activate (Windows)	`.venv\Scripts\activate`
Install packages	`pip install <package>`
Save dependencies	`pip freeze > requirements.txt`
Install from file	`pip install -r requirements.txt`
Deactivate	`deactivate`
Build Docker image	`docker build -t name .`
Run container	`docker run name`
Interactive shell	`docker run -it name /bin/bash`
Mount volume	`docker run -v $(pwd)/data:/app/data name`
Start Compose	`docker compose up --build -d`
Stop Compose	`docker compose down`

The Progression:

Git (version code) → venv (version environment) → Docker (version everything) → W&B (version experiments)