Reproducible work environments

For Data Scientists using Nix

Avik Basu

About Me

- Based in Sunnyvale, CA
- Staff Data Scientist at Intuit
- Build Models for Revenue Predictions
- Engineering + Data Science
- Love RPG Games
- Driving is therapy iff
  - Car is fun + stick shift
  - Roads are curvy
  - No minivan in front of me

What does Deterministic behavior mean?

Replicate the exact same outcome of an experiment across different environments

Why is deterministic behavior important?

1. **Ensures Consistency**
    - Deterministic output
    - Dev machine —> Production system
2. **Allows Collaboration**
    - ~~“Well…, but! It works on my machine 😏”~~
    - Speeds up dev velocity
3. **Provides Transparency**
    - Verifiable
    - Non technical folks can jump in too
4. **Maintains Integrity**
    - Especially true for Data Science projects

Components of Deterministic Behavior

From a Data Science standpoint

A. Code

- Project version
- Scripts, notebooks and other config files

B. Data

- Datasets
- Data sources

C. Models

- Versions
- Random seeds
- Model Stochasticity [1]

D. Environment

- Package versions (Python + Non Python)
- OS versions

[1] https://pytorch.org/docs/stable/generated/torch.use_deterministic_algorithms.html#torch.use_deterministic_algorithms

Components of Deterministic Behavior

Complexity ordering

A. **Code** [Easy]

B. **Data** [Mostly Easy]

C. **Models** [Medium]

D. **Environment** [Hard]

Why is a Deterministic Environment so hard to create?

Why is a deterministic env hard to create?

Python Specific

- Dependency versions
- Python versions
- Non-Python dependencies
- Type of Operating System
- OS versions
- Different platform architecture

Solutions?

1. Python Package managers

Pros

- Poetry, PDM, UV, Pipenv
- Provides dependency locking
    - Direct
    - Transitive
- Can provide Python version locking
- Deterministic Python environments
- **Declarative**

1. Python Package managers

Cons

- Non-Python dependencies can create some troubles
    - C/C++/Rust/Fortran
    - Many scientific computing libraries can fall in this
- Captures only the Python environment; not the full dev environment
    - e.g. users need to have their own Python tools setup in order to run the project

2. Docker containers

Pros

- Can capture the whole dev environment
- Dev containers can be helpful for development
- Great documentation and support
- Go-to standard in production environments

2. Docker containers

Cons

- Some containers can be really resource intensive
- **Imperative** configuration
    - Describe steps rather than a desired state
- Security vulnerabilities
- Might be an overkill for development purposes

3. Nix

3. Nix

What is it?

- Purely functional package manager
    - Built by functions that don’t have side effects
    - Never change after they are built
- Atomic upgrades and rollbacks
    - Never overwrite packages
    - Previous versions never conflict with newer ones
- **Declarative**
- The core idea revolves around reliability and reproducibility

The Nix Ecosystem

Core components

Nix

- ~ pip

Nix Language

- Functional
- Dynamically typed

NixOS

- Fully declarative Linux distribution

Nixpgks

- Largest and most up-to-date software distribution
- ~ PyPI

Nix shell

- Creates shell environments
- ~ virtualenv

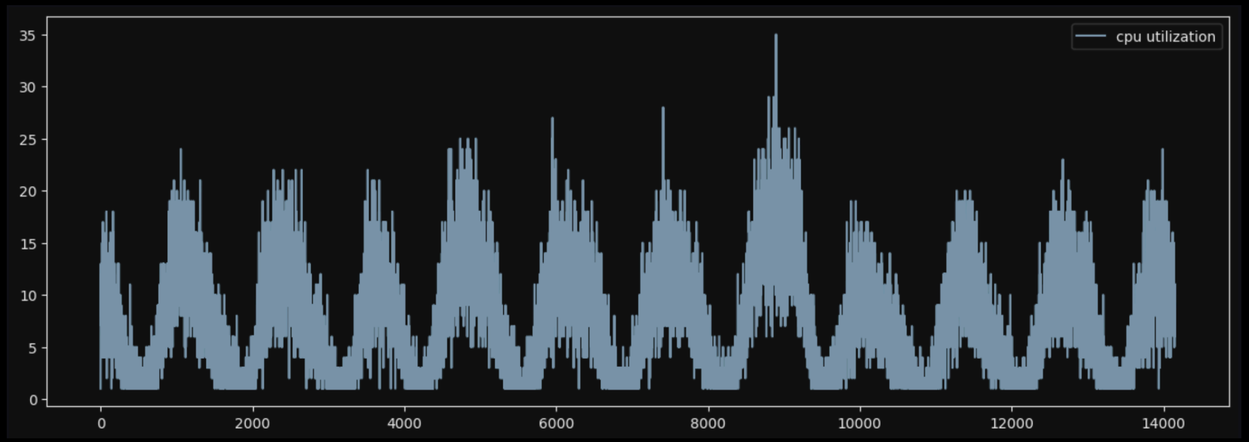
Sample Project

- Uses “uv" for package management
- Conservative versioning
- Just plots the data

```
.
├── .gitignore
├── .python-version
├── README.md
├── data
│   └── tsdata.csv
├── default.nix
├── notebooks
├── pyproject.toml
├── src
│   ├── __init__.py
│   ├── _constants.py
│   └── plot.py
└── uv.lock
```

```
[project]
name = "mydsproject"
version = "0.1.0"
description = "Nothing is here really!"
readme = "README.md"
requires-python = ">=3.12,<3.13"
dependencies = [
    "matplotlib>=3.9,<3.10",
    "numpy>=2.1,<2.2",
    "pandas>=2.2,<2.3",
]
```

```
>> git:(main) ✗ python -m src.plot
```



What if I want to share this project ?

To someone who….

- Is not familiar with Python
- Does not have Python installed (e.g. in a default windows machine)
- Someone non technical (e.g. product managers)
- Who is one of many attendees in a hands-on project workshop

Add default.nix file

In the main directory

```
>> git:(main) ✗ nix-shell

these 58 paths will be fetched (29.38 MiB download, 187.99 MiB
unpacked):
  /nix/store/ykbzldqyxch123y6h1q5v7mk9lp5zkkv-
python3.12-matplotlib-3.9.1
  /nix/store/dksms31747w6szcxc9pynbw5jqblb54m-
python3.12-pandas-2.2.2
…

…Plotting data..
<SHOW THE PLT PLOT>

>> [nix-shell:~/…/mydsproject]$
```

```nix
let
  pkgs = import <nixpkgs> {};
  python = pkgs.python312;
  pythonPackages = python.pkgs;
in with pkgs; mkShellNoCC {
  packages = [
    pythonPackages.numpy
    pythonPackages.pandas
    pythonPackages.matplotlib
  ];
  buildInputs = [
    curl
  ];

  shellHook = ''
    python -m src.plot
  '';
}
```

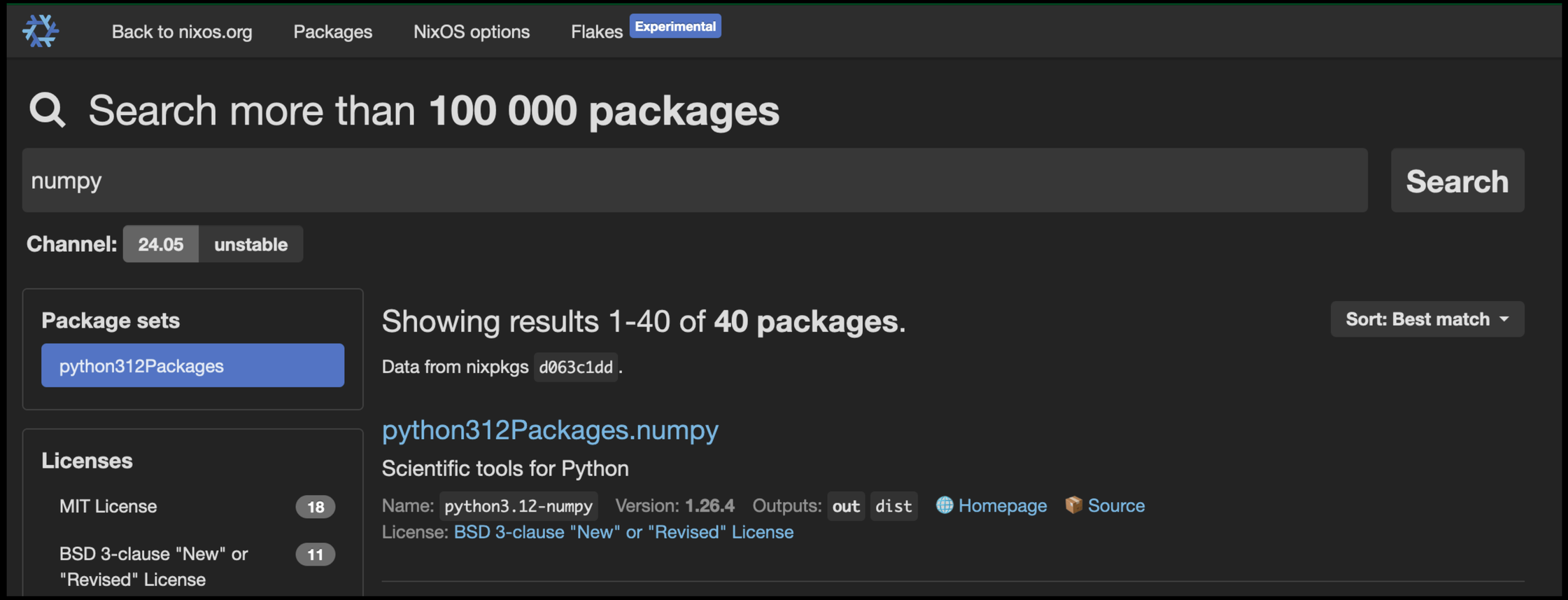
Deterministic env

But not exactly what we wanted..

```
>> [nix-shell: mydsproject]$ which python

/nix/store/ybnf7k6i9p2r-python3-3.12.6/bin/python
```

Package	Version
contourpy	1.2.1
cycler	0.12.1
defusedxml	0.8.0rc2
fonttools	4.53.1
kiwisolver	1.4.5
matplotlib	3.9.1
numpy	1.26.4
olefile	0.47
packaging	24.1
pandas	2.2.2
pillow	10.4.0
pip	24.0
pyparsing	3.1.2
python-dateutil	2.9.0.post0
pytz	2024.2
six	1.16.0
tzdata	2024.1


Back to nixos.org
Packages
NixOS options
Flakes
Experimental
Search more than 100 000 packages
numpy
Search
Channel:
24.05
unstable
Package sets
python312Packages
Licenses
MIT License
18
BSD 3-clause "New" or "Revised" License
11
Showing results 1-40 of 40 packages.
Sort: Best match
Data from nixpkgs d063c1dd.
python312Packages.numpy
Scientific tools for Python
Name: python3.12-numpy
Version: 1.26.4
Outputs: out dist
Homepage
Source
License: BSD 3-clause "New" or "Revised" License

Install from PyPI

Fix dependencies

- Does not use the python packages from nixpkgs
- Uses a virtual env
- Requirements.txt is exported using uv
- Just one of many ways of achieving this!

```
let
  pkgs = import (
    fetchTarball "https://github.com/NixOS/nixpkgs/tarball/nixos-24.05-small"
  ) {};
  python = pkgs.python312;
  pythonPackages = python.pkgs;
in with pkgs; mkShellNoCC {
  packages = [
    pythonPackages.venvShellHook
  ];
  buildInputs = [
    curl
  ];
  shellHook = ''
    VENV=.venv
    python3.12 -m venv $VENV
    source ./$VENV/bin/activate
    export PYTHONPATH=`pwd`/$VENV/${python.sitePackages}/:$PYTHONPATH
    pip install -r requirements.txt
    python -m src.plot
  '';
  postShellHook = ''
    ln -sf ${python.sitePackages}/* ./.venv/lib/python3.12/site-packages
  '';
}💡
```

Validate

```
>> [nix-shell: mydsproject]$ which python

~/Users/avikbasu/Projects/mydsproject/.venv/bin/python
```

```
Package           Version
----------------- -----------
contourpy         1.3.0
cycler            0.12.1
fonttools         4.54.1
kiwisolver        1.4.7
matplotlib        3.9.2
numpy             2.1.3
packaging         24.1
pandas            2.2.3
pillow            11.0.0
pip               24.0
pyparsing         3.2.0
python-dateutil 2.9.0.post0
pytz              2024.2
six               1.16.0
tzdata            2024.2
```

Q: How can someone else run my project in a deterministic manner?

A: Install Nix, and run nix-shell

Drawbacks

No free lunch! 🥲

- Hard language to learn
- Fairly complex concepts to grasp
- Not beginner friendly
- Not very widely adopted in the Python community
- There is a minor performance overhead

Other tools in the ecosystem

Can make life easier

Nix Flakes

- Enforce a uniform structure for Nix projects
- Pin dependency versions in a lock file
- Still experimental

devenv

- Declarative, Reproducible and Composable dev envs
- JSON like language
- Written in Nix

Thank You!