Reproducible Scalable Workflows with Nix, Papermill and Renku

PyCon India 2020

Rohit Goswami, MInstP AMIE AMIChemE
rog32@hi.is

UNIVERSITY OF ICELAND
FACULTY OF PHYSICAL SCIENCES

October 3, 2020

Outline

Backstory

Standard Approach

- Language Agnostic

Workflow

- Write functions/objects
  - Refactor in modules
- Test
  - Unit
  - Integration
- Documentation
- Use after importing

Not interactive enough for data-files

Modern Data Analysis

- Try before you buy
- Doesn't play nice with tests

Python Interactivity

- IPython (`ipython`)
- Jupyter (Lab/Notebook)
- Colab (Google)

```
df = pd.read_excel("https://archive.ics.uci.edu/ml/machine-learning-databases/00342/Data_Cortex_Nuclear.xls")
df.head()
```

```
--INSERT--
```

	MouseID	DYRK1A_N	ITSN1_N	BDNF_N	NR1_N	NR2A_N	pAKT_N	pBRAF_N	pCAMKII_N	pCREB_N	pELK_N	pERK_N	pJNK_N
0	309_1	0.503644	0.747193	0.430175	2.816329	5.990152	0.218830	0.177565	2.373744	0.232224	1.750936	0.687906	0.306382
1	309_2	0.514617	0.689064	0.411770	2.789514	5.685038	0.211636	0.172817	2.292150	0.226972	1.596377	0.695006	0.299051
2	309_3	0.509183	0.730247	0.418309	2.687201	5.622059	0.209011	0.175722	2.283337	0.230247	1.561316	0.677348	0.291276
3	309_4	0.442107	0.617076	0.358626	2.466947	4.979503	0.222886	0.176463	2.152301	0.207004	1.595086	0.583277	0.296729
4	309_5	0.434940	0.617430	0.358802	2.365785	4.718679	0.213106	0.173627	2.134014	0.192158	1.504230	0.550960	0.286961

5 rows × 82 columns

```
plt.figure(figsize=(8,4))
sns.heatmap(df.corr(),cmap='Greens',annot=False)
```

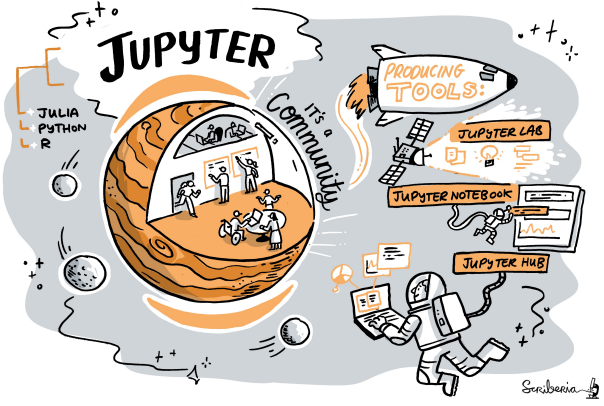
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5e19121fd0>
```



```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(neigh,X_test,y_test)
plt.show()
```

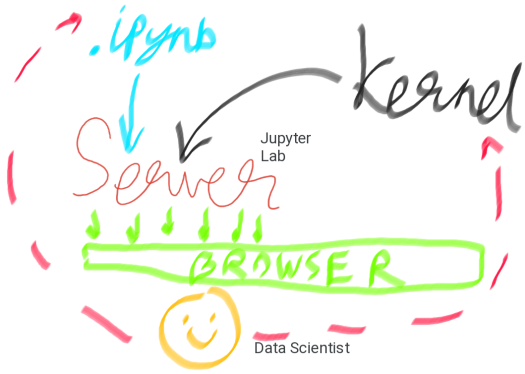
Jupyter

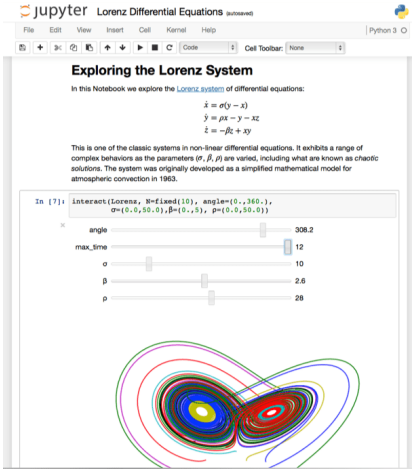
The Turing Way Community et al.
The Turing Way: A Handbook for Reproducible Data Science.
Version v0.0.4. Zenodo, Mar. 25, 2019



Jupyter Notebooks

- `ipynb` files can be exported independently
  - Or via server extensions

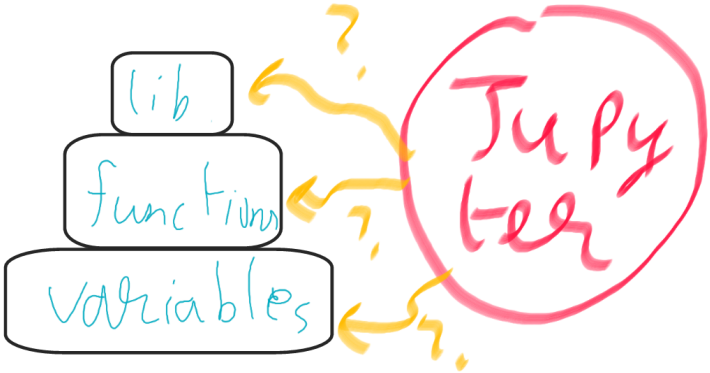




From the Jupyter documentation

Reconciliation

- A crisis of faith
- Made worse by Colab



Packaging

Python Modules

- A `.py` file is a module
- It is standalone if it only imports from the standard library

Pure Python Packages

- A directory with `__init__.py` in it is a package
- Use `pip`

Distributions

Standard

- Built by `setuptools` with `setup.py`
- Simple source only `.tar.gz`

Binary

- `wheel`
  - For all your interoperable needs
  - Includes static libraries

- Distributions have zero or more packages

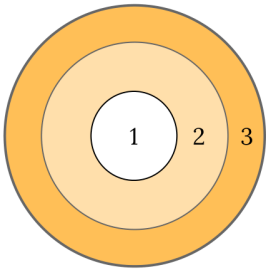
The Python Gradient

Consider the packaging gradient[1]

Packaging for Python **libraries**

1. **`.py`** - standalone modules
2. **sdist** - Pure-Python packages
3. **wheel** - Python packages

(With room to spare for static vs. dynamic linking)



- Libraries and Dev tools are all we get (from PyPI)

[1] by Mahmoud Hashemi (PyBay'17): `https://www.youtube.com/watch?v=iLVNWfPWAC8`

Pip Requirements

- Python
- System libraries
- Build tools
  - Wheels don't work for arbitrary distributions

Dependency Resolution

- `requirements.txt` (pip)
- Poetry (pretty)
  - `pyproject.toml`
  - `poetry.lock`
- Pipenv (older)
  - `Pipfile` + lockfile
- Pipx (`pip` but for applications)
- Pyenv and friends

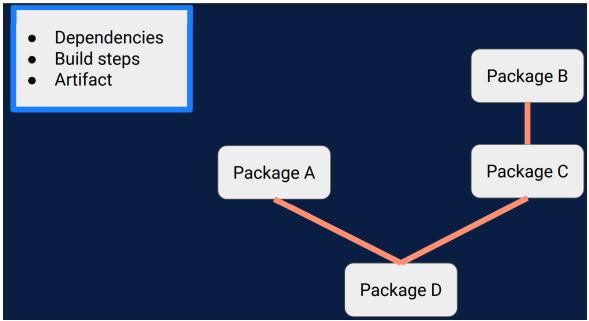
System Dependencies

- Appimages
- Containers
  - `docker`, `flatpak`, `snapcraft`
- Impure filesystems
  - Anaconda

Section III

Nix

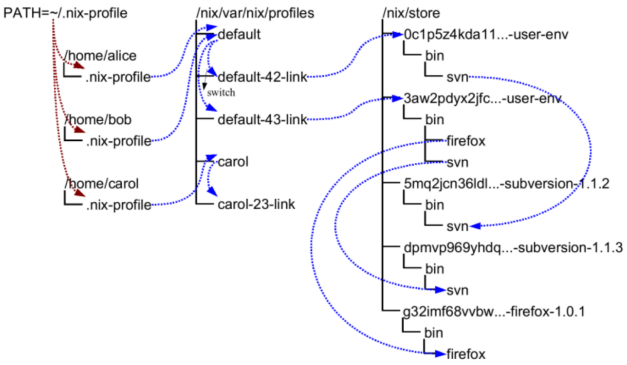
What?



- from `https://brianmckenna.org/files/presentations/rootconf19-nix.pdf`

Nix

Eelco Dolstra, Merijn de Jonge, and Eelco Visser. “Nix: A Safe and Policy-Free System for Software Deployment”. In: (2004), p. 15, Eelco Dolstra, Andres Löh, and Nicolas Pierron. “NixOS: A Purely Functional Linux Distribution”. In: *Journal of Functional Programming* 20.5-6 (Nov. 2010), pp. 577–615



Why?

Protects against self harm

Exposes things taken for granted

Enforces consistency

Reliable Purely functional, no broken dependencies

Reproducible Each package is in isolation

How? store + hash + name + version

Installation (Multi-User)

```
sh <(curl https://nixos.org/nix/install) --daemon
```

- Needs `sudo` but should not be run as root
- Will make build users with IDs between 30001 and 30032 along with a group ID 30000

Nix Python - Trial I

```
nix-shell -p 'python38.withPackages(ps: with ps; [ numpy toolz ])'
```

- Check which `python` is loaded
- Check which modules are present

Nix with Scripts

```
#! /usr/bin/env nix-shell
#! nix-shell -i python3 -p "python3.withPackages(ps: [ps.numpy])"

import numpy

print(numpy.__version__)
```

An Aside into Purity

```
nix-shell --pure --run
↪  'bash'
```

- Why?
- What do we solve with this?

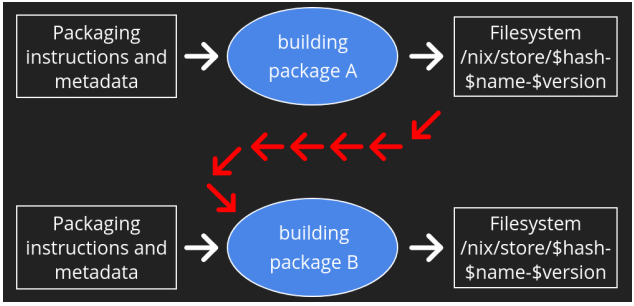


Figure: Stateless builds from `https://slides.com/garbas/mozilla-all-hands-london-2016#/7/0/3`

Shell in a File

```
with import <nixpkgs> {};

let
  pythonEnv = python35.withPackages (ps:
   ↪  [
    ps.numpy
    ps.toolz
  ]);
in mkShell {
  buildInputs = [
    pythonEnv
    which
  ];}
```

- What tools are we adding?
- What environment are we using?

Nix Python Expressions I

```
f90wrap = self.buildPythonPackage rec {
  pname = "f90wrap";
  version = "0.2.3";
  src = pkgs.fetchFromGitHub {
    owner = "jameskermode";
    repo = "f90wrap";
    rev = "master";
    sha256 =
     ↪  "0d06na14xzg8vv6sjdbmg2n88a8h8df5ajam72445mhzk08yin23";
  };
  buildInputs = with pkgs; [ gfortran
   ↪  stdenv ];
```

- The self portion is from overriding the python environment
- We will dispense with this later

Nix Python Expressions II

```
  propagatedBuildInputs = with self; [
    setuptools
    setuptools-git
    wheel
    numpy
  ];
  preConfigure = ''
    export
  ↪ F90=${pkgs.gfortran}/bin/gfortran
  '';
  doCheck = false;
  doIstallCheck = false;
};
```

- More details here: `https://rgoswami.me/posts/ccon-tut-nix/`
- Note that the `propagatedBuildInputs` are for the python packages

Friendly Nix

```
nix-env -i nox
nox niv
```

Niv	For pinning packages
Nox	Interactive package management
Lorri	For automatically reloading environments
Mach-Nix	For working with Python
Nix-Prefetch-Url	For obtaining SHA hashes

Setup

Replacing Conda I

```
let
  sources = import
   ↪  ./prjSource/nix/sources.nix;
  pkgs = import sources.nixpkgs { };
  mach-nix = import (builtins.fetchGit {
    url =
     ↪  "https://github.com/DavHau/mach-nix/";
    ref = "2.2.2";
  });
```

- Note our definition of `mach-nix`
- Best practices involve `niv` pinned sources

Replacing Conda II

```
  customPython = mach-nix.mkPython {
    requirements = builtins.readFile
     ↪  ./requirements.txt;
    providers = {
      _default = "nixpkgs,wheel,sdist";
      pytest = "nixpkgs";
    };
    pkgs = pkgs;
  };
in pkgs.mkShell { buildInputs = with
 ↪  pkgs; [ customPython ]; }
```

- More details here: `https://rgoswami.me/posts/mach-nix-niv-python/`

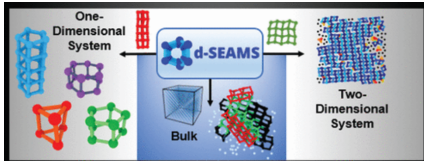
Replacing Conda III

```
overrides_pre = [
  (pythonSelf: pythonSuper: {
    pytest =
      ↪ pythonSuper.pytest.overrideAttrs
      ↪ (oldAttrs: {
       doCheck = false;
    });
    f90wrap =
      ↪ pythonSelf.buildPythonPackage
      ↪ rec {...};
  })
];
```

- An important aspect of `mkPython`
- More details here: `https://rgoswami.me/posts/mach-nix-niv-python/`

More Nix

- Try Nix Pills
- Roll your own environment
- Make a docker image
- Try a more complex system (d-SEAMS [4])



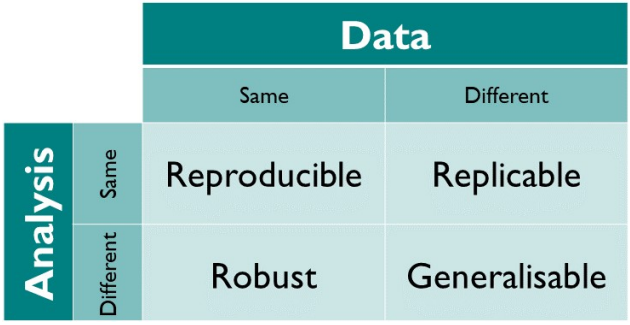
Reproducibility

What Reproducibility?

The Turing Way Community et al.
The Turing Way: A Handbook for Reproducible Data Science.
Version v0.0.4. Zenodo, Mar. 25, 2019

Data Science Woes

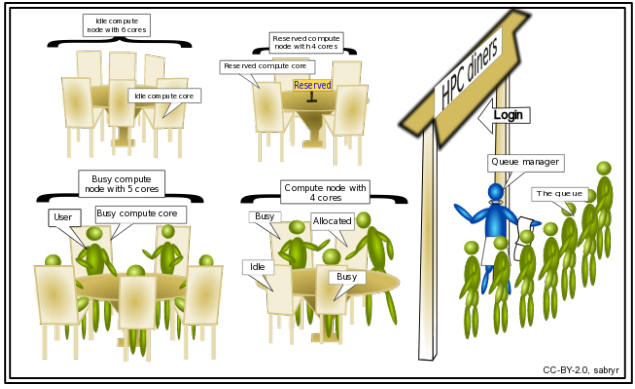
- Version Control
  - Git, SVN, Mercurial (hg)
- Collaboration
  - Overleaf, Google Drive, OneDrive
- Reproduce environments
  - Docker, Conda, Nix
- Re-run analysis
  - Luigi, any CWL runner



Provenance and Clusters

Cluster Woes

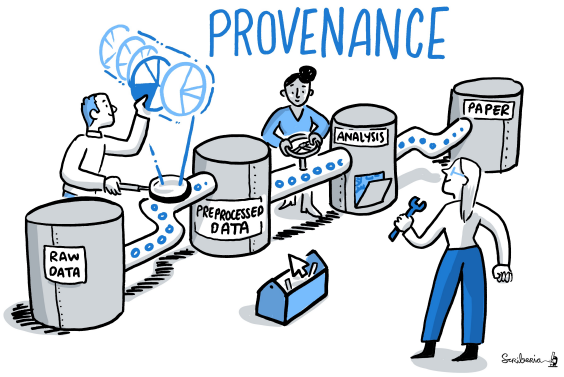
- No `docker`
  - If lucky, will have `singularity`
- No userspace support
- Probably runs CentOS or something
- Has a networked file system
- Uses a resource queue
  - Slurm, PBS
- Might have support for `lmod`



Provenance

The Turing Way Community et al.
The Turing Way: A Handbook for Reproducible Data Science.
Version v0.0.4. Zenodo, Mar. 25, 2019

COMMON WORKFLOW LANGUAGE



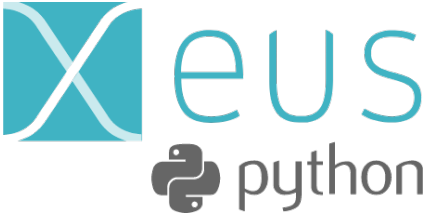
Setup Jupyter

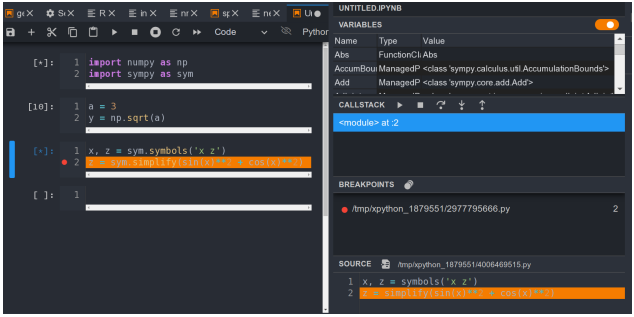
- Prefer `conda`
  - Export a `yml` with setup
- Use `nvm`
- Track provenance manually
  - For plugins and setup
- Consider `direnv`

```
jupyter lab --generate-config
vim ~/.jupyter/jupyter_notebook_config.py
# Change c.NotebookApp.notebook_dir to a
  ↪ full path
```

Xeus Python

- Best Jupyter debugger
- Does not support all magics





Reusing Notebooks


```
- jupy
+ text
```

Papermill

- Notebooks are functions
- Can be parameterized on the fly
  - No need to refactor
  - Cells become the analysis
- Mostly supports integration tests

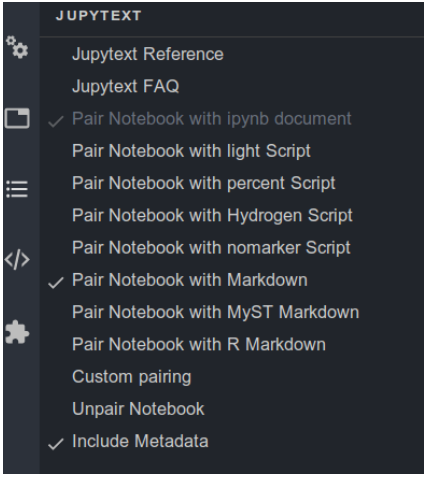
Jupytext

- Notebooks are literate snippets
- Must refactor cells into functions
- Supports testing more transparently
  - Unit tests

- Actually these work best together, especially as `papermill` can be called in `python` directly

Jupytext I

- Works best with version control
  - Never commit an `.ipynb`!
- Encourages functions
  - Easier to unit-test later
- Is literate
  - Closer to `ipython` histories
  - Fits well with `orgmode` (via `pandoc`)



Jupytext II

File: **getMBP.ipynb**

```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 19,
   "metadata": {},
   "outputs": [],
   "source": [
    "from ase.io import read as asr\n",
    "from ase.io import write as asw\n",
    "from pprint import pprint as ppp\n",
    "import itertools, collections\n",
    "import os, shutil, tempfile\n",
    "import sys, subprocess\n",
    "from pathlib import Path"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# MBX Helper\n",
    "This document describes the usage of the `mbx` softwa
re along with auxillary tasks. The following tasks are car
ried out:\n",
    "- [ ] Splitting `xyz` trajectories\n",
    "- [ ] Converting `xyz` files to `nrg` files\n",
    "- [ ] Obtaining energies with the `single_point` bina
ry"
   ]
  },
  {
```

File: **getMBP.md**

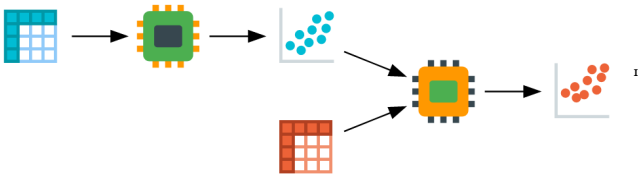
```
---
jupyter:
  jupytext:
    formats: ipynb,md
    text_representation:
      extension: .md
      format_name: markdown
      format_version: '1.2'
      jupytext_version: 1.6.0
  kernelspec:
    display_name: Python 3
    language: python
    name: python3
---

```python
from ase.io import read as asr
from ase.io import write as asw
from pprint import pprint as ppp
import itertools, collections
import os, shutil, tempfile
import sys, subprocess
from pathlib import Path
```

MBX Helper
This document describes the usage of the `mbx` software al
ong with auxillary tasks. The following tasks are carried
out:
- [ ] Splitting `xyz` trajectories
- [ ] Converting `xyz` files to `nrg` files
- [ ] Obtaining energies with the `single_point` binary
```

Renku

- Has a Web-UI
- Uses standard Git LFS under the hood
- Generates CWL files for each command
  - These become a provenance or lineage history
  - Image from renku docs



*Renku (連句 "linked verses"), is a Japanese form of popular collaborative linked verse poetry, written by more than one author working together.**

—Wikipedia

```
renku run python
↪ run_analysis.py -i
↪ inputs -o outputs
```

Towards the Future

Parting Practicalities

- Keep Jupyter impure
- Do not rely on Colab
- Always keep a plain-text version
- Replace functions with parameterized Jupyter notebooks
- Reduce magics in parameterized notebooks
  - Maximize Xeus where possible
- Ask your sys-admins for `nix` or try a user-install
  - Unsupported but try: `https://rgoswami.me/posts/local-nix-no-root/`
- `conda` should be used for global installations
  - Like Jupyter
- Use `nix` derivations for actual environments
- Use `renku` to track provenance per project
  - Also tracks databases

Conclusions

- Interactivity is here to stay
  - Especially in data science
- Jupyter notebooks are here to stay
  - Scalable development is still possible
- Nix ensures reproducible system dependencies

- Meeting old-school TDD developers halfway is best

Tools

Xeus Python	PDB on steroids for Notebooks
Jupytext	Version Control and literate programming
Papermill	Notebooks-are-functions
Renku	Write CWL without tears

References I

- The Turing Way Community et al. *The Turing Way: A Handbook for Reproducible Data Science*. Version v0.0.4. Zenodo, Mar. 25, 2019.
- Eelco Dolstra, Merijn de Jonge, and Eelco Visser. "Nix: A Safe and Policy-Free System for Software Deployment". In: (2004), p. 15.
- Eelco Dolstra, Andres Löh, and Nicolas Pierron. "NixOS: A Purely Functional Linux Distribution". In: *Journal of Functional Programming* 20.5-6 (Nov. 2010), pp. 577–615.
- Rohit Goswami, Amrita Goswami, and Jayant K. Singh. "D-SEAMS: Deferred Structural Elucidation Analysis for Molecular Simulations". In: *Journal of Chemical Information and Modeling* 60.4 (Apr. 27, 2020), pp. 2169–2177.

End

Thank you