GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Data Science – Basics

Lecture 08 – Consistency

Fabian Sinz & Dominik Becker

03. June 2024

Institute for Computer Science – Campus Institute for Data Science (CIDAS)

Reproducibility?

Even if you code does not change it's behavior could.

What could affect that?

Reproducibility?

Even if you code does not change it's behavior could.

What could affect that?

Reproducibility

- Different operating system
- Different versions of software libraries
- Different hardware

Reproducibility?

Even if you code does not change it's behavior could.

What could affect that?

Reproducibility

- Different operating system
- Different versions of software libraries
- Different hardware

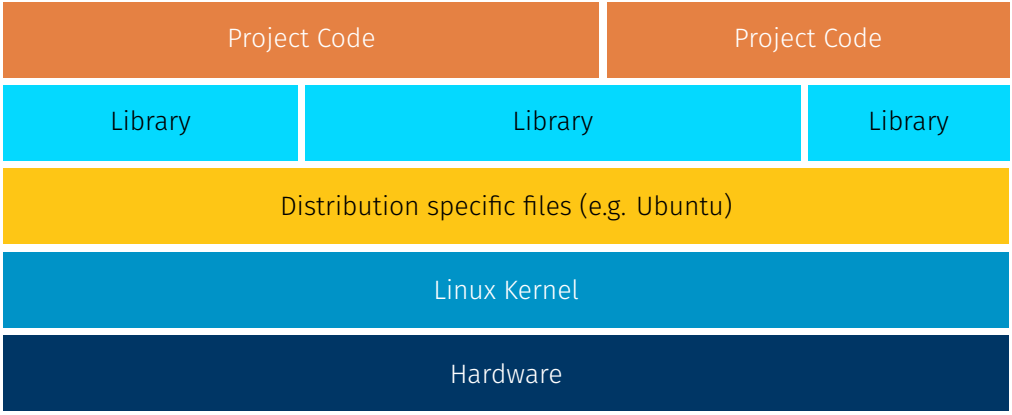
Today we are talking about three tools to deal with the software problems

1. Virtual Environments
2. Containerization (Docker)
3. Unit Tests

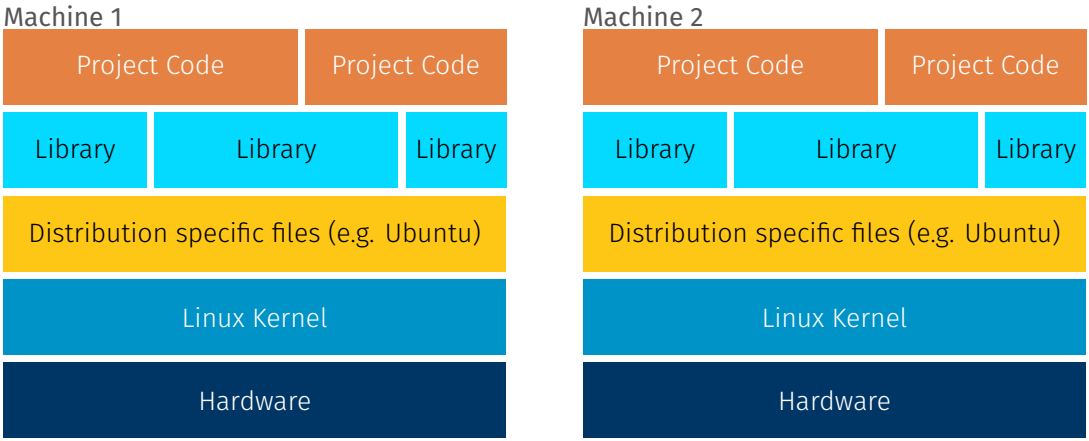
This will help you to ensure that your code/projects also work

- quite some time after you have finished them
- on a different machine

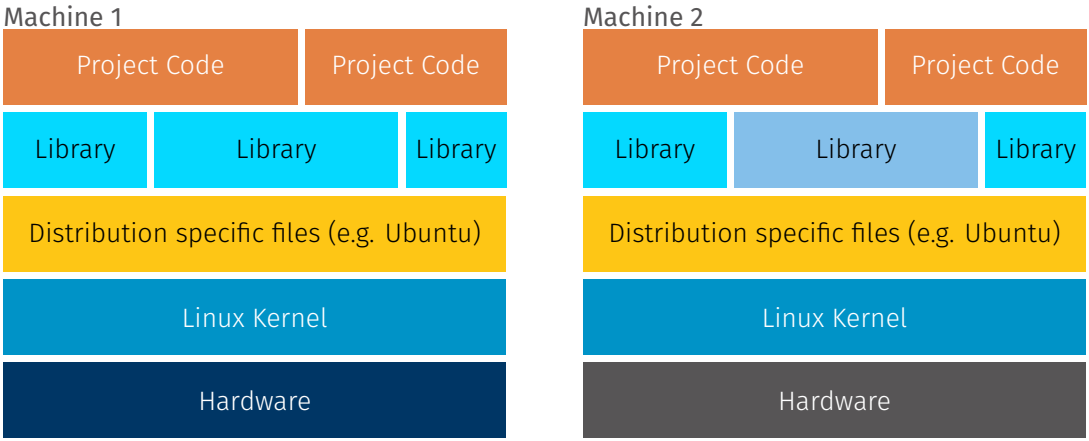
Code environment



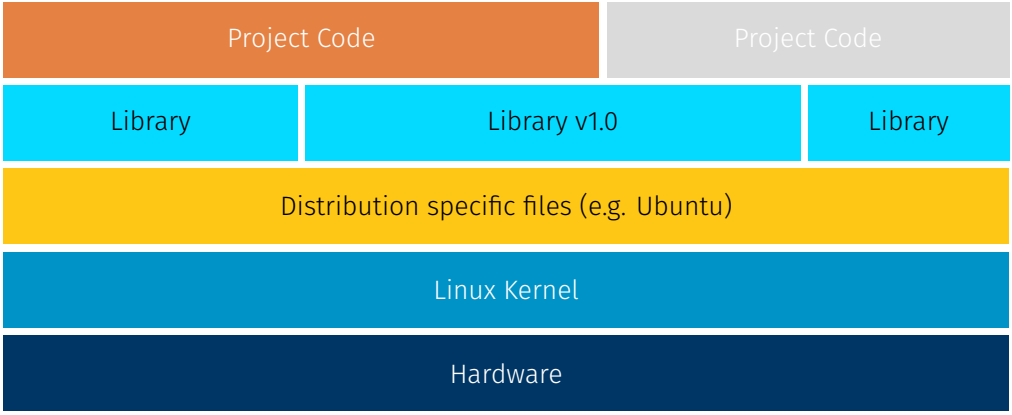
Ideal replication



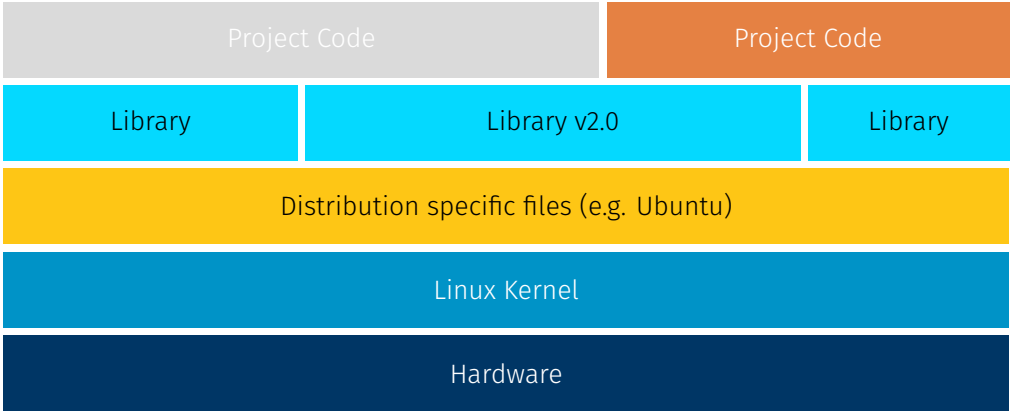
What tends to happen



Libraries can conflict

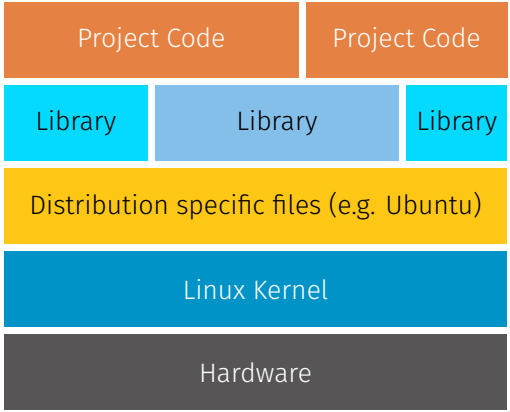


Libraries can conflict



Additional complications

- Some libraries are pain to install with a lot of other dependencies
- Getting the library installed and configured properly is not a trivial task
- This is particularly bad if there are already many other libraries present
- This is important but a very wasteful use of any of our time!



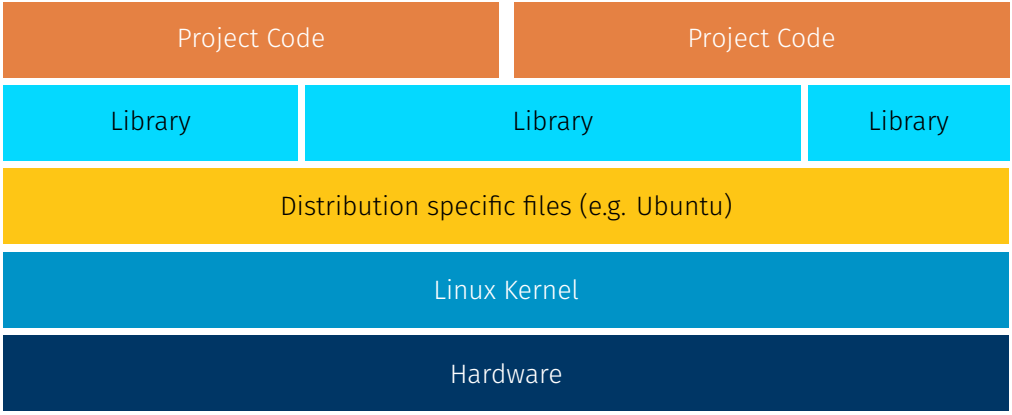
Virtual Environments

Python Virtual Environment (venv)

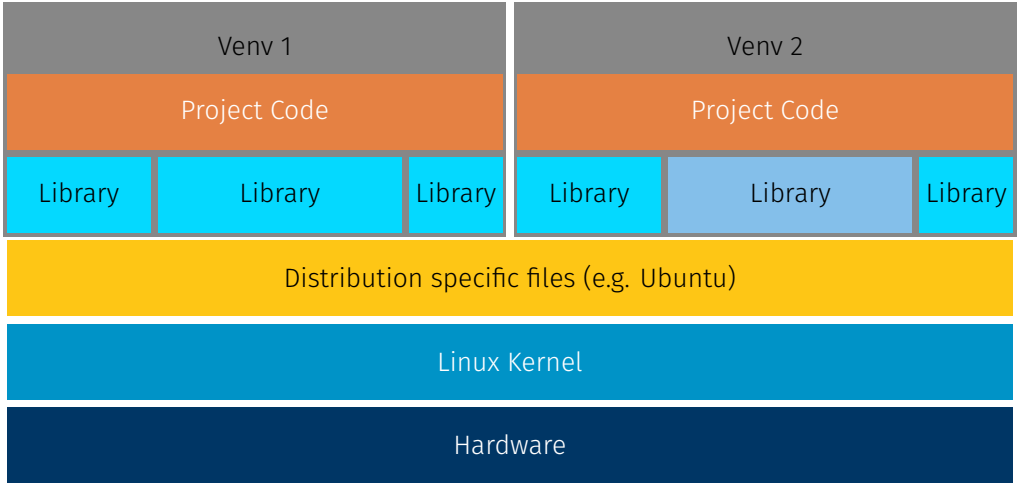
Python Virtual Environments

- isolate the dependencies of your python project
- install local copies of your python libraries into the environment
- allow you to customize your local library setup for each project
- make your environment reproducible
- `venv` is part of the standard library and the recommended way for virtual environments since Python 3.5

Code environment with venv



Code environment with venv



Work cycle with venv

At the beginning you need to create the virtual environment **once** with `python3 -m venv venv` in your project folder.

Each time you work on your project you need to

1. **activate** the enviroment with `source venv/bin/activate`
2. **work and code** in your virtual enviroment. This include package **installation** with `python -m pip install <package-name>` or simply `pip install <package-name>`.
3. **deactivate** it when you stop working on it via `deactivate`

Note: The commands might differ a bit on Windows. Please see the linked tutorial.

[https://realpython.com/python-virtual-environments-a-primer/]

Why use virtual environments

- Python is not great at **dependency management**. By default `pip` will install all dependencies into the global `site-packages` of your system.
- This can create **conflicts** and could mess up the dependencies of you project when your system is updated.
- Your project (especially when it gets older) might need **different versions of libraries** than your system. This can create conflicts.
- You do not need administrator rights to install packages into a virtual environment.
- It's easy to get the **exact dependencies** of you project in an virtual environment.

```
(venv) bash-3.2$ pip freeze
  cycler==0.11.0
  fonttools==4.37.1
  [...]
  seaborn==0.11.2
  six==1.16.0
```

[https://realpython.com/python-virtual-environments-a-primer/]

What is a virtual environment

venv creates a local file structure with all dependencies. When you look at it (with `tree -L 1 venv`), you will see something like

```
(venv) bash-3.2$ tree -L 1 venv/
venv/
├── bin
├── include
├── lib
├── pyvenv.cfg
└── share
```

- bin/ contains all executables.
- includes/ contains the header files with packages with C dependencies
- lib/ contains the site-packages
- pyvenv.cfg is the configuration file for the environment

[https://realpython.com/python-virtual-environments-a-primer/]

How does a virtual environment work

- `venv` recreates the folder structure of a standard installation and copies/symlinks the binaries for `python`.
- For the standard libraries, it uses you global system's python installation. Python finds that in the pyvenv.cfg

```
(venv) bash-3.2$ cat venv/pyvenv.cfg
  home = /usr/local/opt/python@3.10/bin
  include-system-site-packages = false
  version = 3.10.6
```

- It modifies your PYTHONPATH to change where Python is looking for libraries.
- On activation it also changes your shell's PATH variable and command prompt.

[https://realpython.com/python-virtual-environments-a-primer/]

Live demo venv

live demo

Python's `venv` (not to be confused with `virtualenv`) is a tool to isolate the dependencies of your project from the main system by

1. creating a local folder structure and installing the site-packages there
2. changing the paths of Python and your shell to point to that local installation.

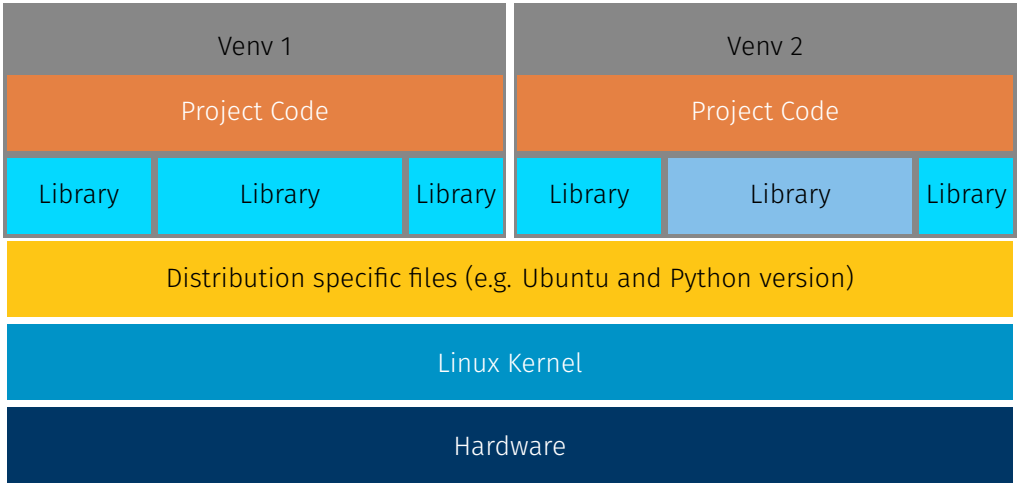
Important things to keep in mind:

- You need to create the environment **once**. After that you only need to **activate** or **deactivate** your environment.
- Your virtual environment **depends on the python installation of your system**. If that changes, the virtual environment needs to be updated or breaks.

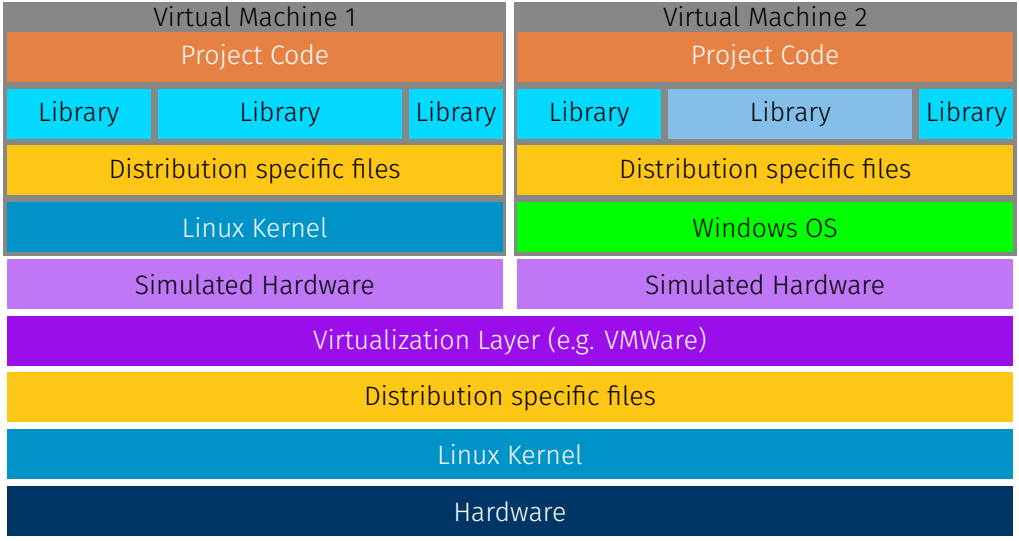
Containerization

Code environment with venv

venv depends on your operating system and Python version.



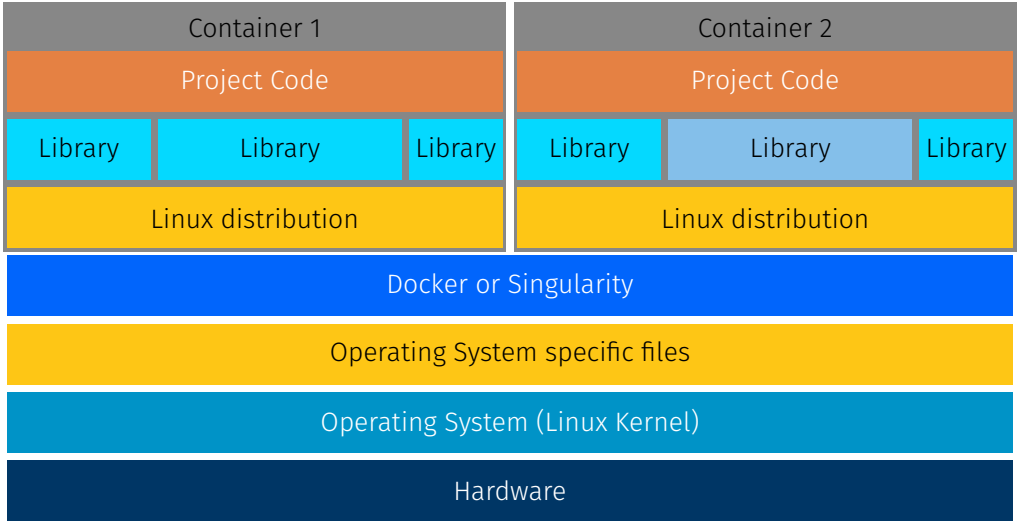
Potential solution: Virtual Machines



Possible issues with VMs

- A lot of overhead just to reproduce the code for one project
- Virtualization layers could lead to performance drop
- Resource intensive
- Potentially difficult to install and administer

Possible solution: Containers



- Docker uses **OS-level virtualization**
- It consists of a **daemon** (`dockerd`) and a **client** (`docker`) that manage containers and allow user interaction.
- A docker **image** consists of several (write protected) layers that can build on top of each other.
- Your project packaged up with all the files it needs and a light-weight Linux distribution. You can add new user-defined layers.
- Each image can be instantiated into multiple **containers**.
- All containers have access to the host system **kernel** (more efficient than virtualization).
- Images can be **shared** via a registry (Dockerhub).
- **But:** Docker usually needs root access (security hazard).

Running your first container

1. Make sure docker is installed and the daemon is running.
2. Open a shell and type

```
docker run hello-world # on Linux you might need sudo
```

which will produce an output like

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest:
↪  sha256:7d246653d0511db2a6b2e0436cfd0e52ac8c066000264b3ce63331ac66dca625
Status: Downloaded newer image for hello-world:latest
```

What just happened?

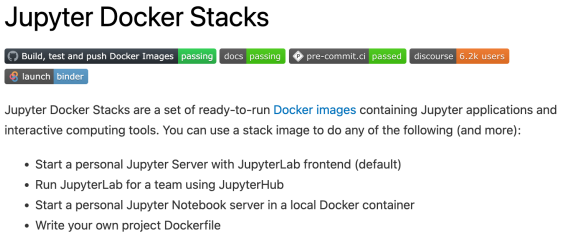
The `hello-world` container will also tell you what happened

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.
```

Useful knowledge about containers

- A typical container will start, execute one command and then exit (like the `hello-world` container).
- A container can also run a service (like jupyter lab).
- For many applications there are already preconfigured containers



- However, running these containers needs additional argument so the containers can interact with your host systems (e.g. files and ports).

[https://jupyter-docker-stacks.readthedocs.io/en/latest/index.html]

Running jupyter lab in a container

The command to run a lab server in a container is

```
docker run -it --rm -p 10000:8888 -v "${PWD}":/home/jovyan/work
↪   jupyter/datascience-notebook:0fd03d9356de
```

Afterwards, the jupyter lab server is accessible under

`http://127.0.0.1:10000/lab?token=<TOKEN>`

where `<TOKEN>` will be specific to your environment.

[https://jupyter-docker-stacks.readthedocs.io/en/latest/index.html]

Running jupyter lab in a container: step by step

GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

```
docker run -it --rm -p 10000:8888 -v "${PWD}":/home/jovyan/work
↪   jupyter/datascience-notebook:0fd03d9356de
```

1. `-it` will make it interactive and attache a terminal. That's needed so you can stop the server with CTRL-C.
2. `-rm` will remove the container when it exits. Otherwise it would stay there and take up space (and could be restarted).
3. `-p 10000:8888` maps the port `8888` inside the container to `10000` outside.
4. `-v "$PWD":/home/jovyan/work` maps your current directory to `/home/jovyan/work` inside the container
5. `jupyter/datascience-notebook:0fd03d9356de` is the name and the tag of a particular image.

That's a lot of switches

- Docker can have a lot of parameters. Calling them from the commandline can be annoying.
- `docker-compose` is a tool so you can store the configuration in a file `docker-compose.yml` and just start the container(s) with `docker-compose up`.

```
version: "3"
services:
  datascience-notebook:
      image:  jupyter/datascience-notebook:0fd03d9356de
      volumes:
        - ./:/home/jovyan/work
      ports:
        - 10000:8888
      container_name: fabians-datascience-container
```

How to interact with docker and containers

- `docker ps` will list all running containers.
- `docker stop <CONTAINERNAME>` will stop a running container.
- `docker kill <CONTAINERNAME>` will kill a running container.
- `docker exec <CONTAINERNAME> <COMMAND>` will run a command in a running container.
- `docker images` shows all downloaded images.
- `docker help` will list more options.

What if I want to install more libraries

- You can add new layers to an existing image by creating a `Dockerfile` in you directory.

```
FROM jupyter/datascience-notebook:0fd03d9356de

RUN pip3 install datajoint
```

- You can build the image by running

```
docker build -t my-datascience-container .
```

  in the directory of the `Dockerfile`.

- You can start the new image via

```
docker run -it --rm -p 10000:8888 -v "${PWD}":/home/jovyan/work
↪  my-datascience-container
```

Live demo docker

live demo

- Docker is a very useful tool to create reproducible environments for your projects.
- We only scatched the surface. Do look into docker and maybe do a tutorial.
- Images can easily be share with others.
- You can build on top of existing images to install additional libraries.
- There are many images with pre-installed and configured libraries.
- They always run Linux inside.
- BUT: Docker usually has root access. So be careful and protect your containers.

Unit Tests

What is code testing?

- In every project you should always run a loop of
  1. form an expectation
  2. run analysis/code
  3. compare results to expectation
- Everytime you change your code or environment, something in your code could break.
- You could run little checks manually everytime, but that's cumbersome and costs a lot of time.
- Unit tests are ways to automate this.

Code structure for testing

- To allow for easier testing it is better to separate the machinery (functions, classes, and modules) from the execution environment (plotting, analysis, jupyter notebook).
- `my_functions.py` contains the code that does the actual analysis.
- `analysis_and_plotting.ipynb` runs the functions and plots the results.
- This also makes your analysis code easier to read.

```
pytest-example/
├── analysis_and_plotting.ipynb
└── my_functions.py
```

Assume you have the following function in `my_functions.py`

```python
import numpy as np

def my_average(data: np.ndarray) -> float:
    return data.sum() / len(data)
```

In order to write a test you put another function in a new file `test_my_functions.py`

```python
from my_functions import my_average, np

def test_my_average():
    test_data = np.random.randn(10)
    assert (test_data.mean() - my_average(test_data)) < 1e-8
```

Then you execute the test by running `pytest` in a shell in the directory.

- PyTest will run all files of the form `test_*.py` or `*_test.py` in the current directory and its subdirectories (click for all discovery rules).
- To run more than one test, just add additional functions to the file.
- You can also group multiple test into a class.

You can also reuse the same function to run multiple tests

```
from my_functions import my_average, np
import pytest

...

test_data = np.random.randn(100)

@pytest.mark.parametrize("n, expected",
            [(n, test_data[:n].mean()) for n in range(10, len(test_data), 10)])
def test_my_average_bulk(n, expected):
    assert (expected - my_average(test_data[:n])) < 1e-8
```

Live demo `unit tests`

live demo

1. **Unit tests** test particular parts of code.
2. In python one of the most common libraries is `pytest`.
3. Testing is encouraged to ensure that code produces the right results even when it's changed or run on a different machine.
4. Testing can also be integrated with GitHub (via other services) and be combined with Docker (run tests in different setups).

Thanks for listening.
Questions?