

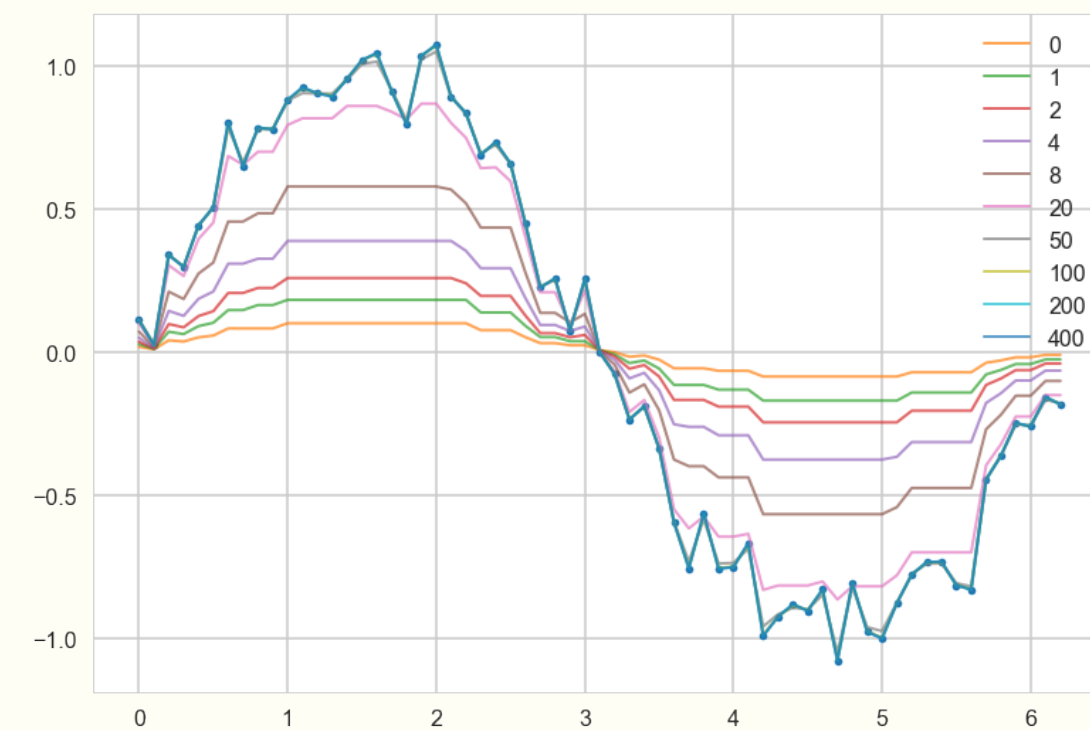
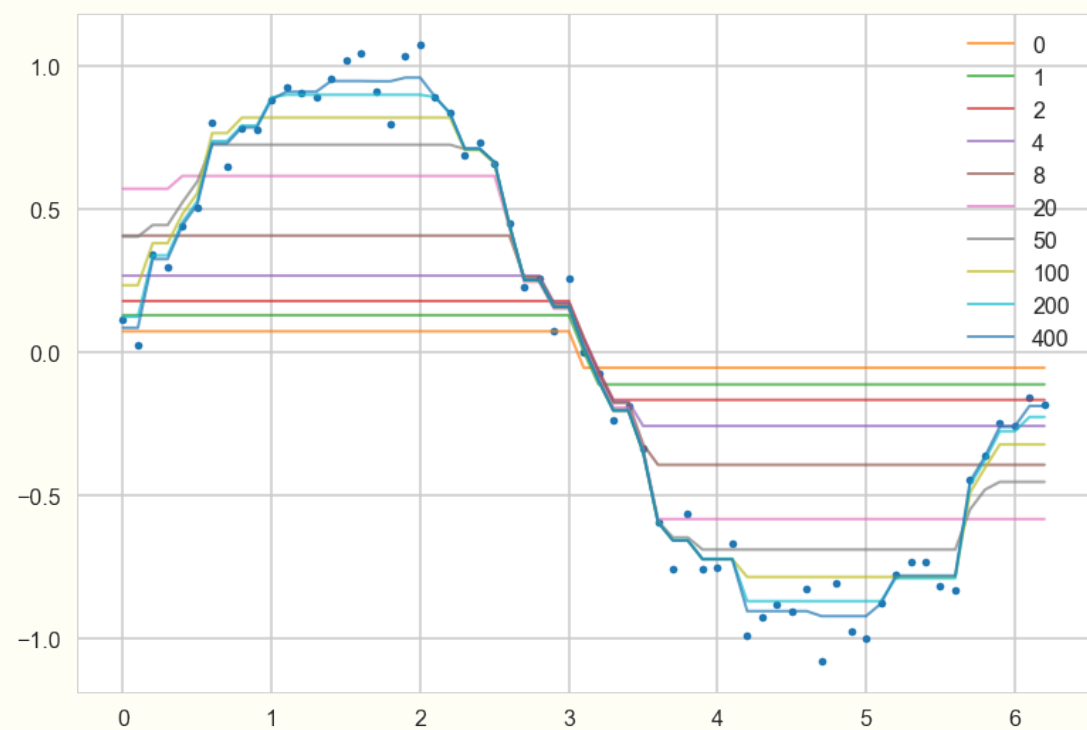
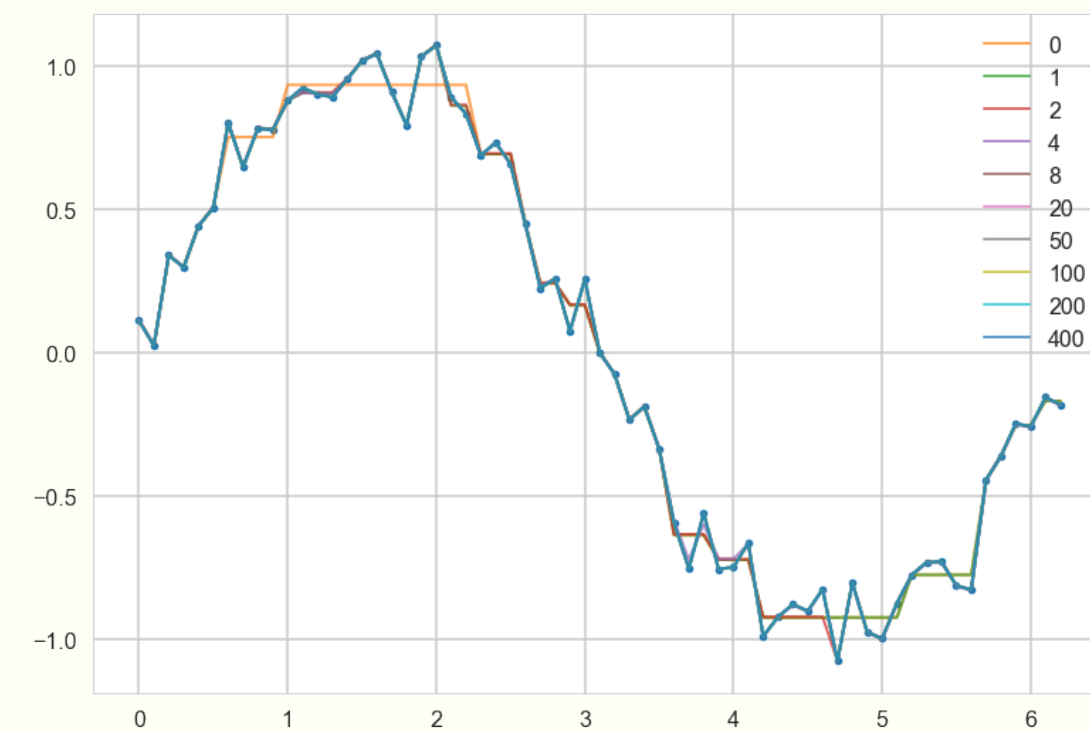
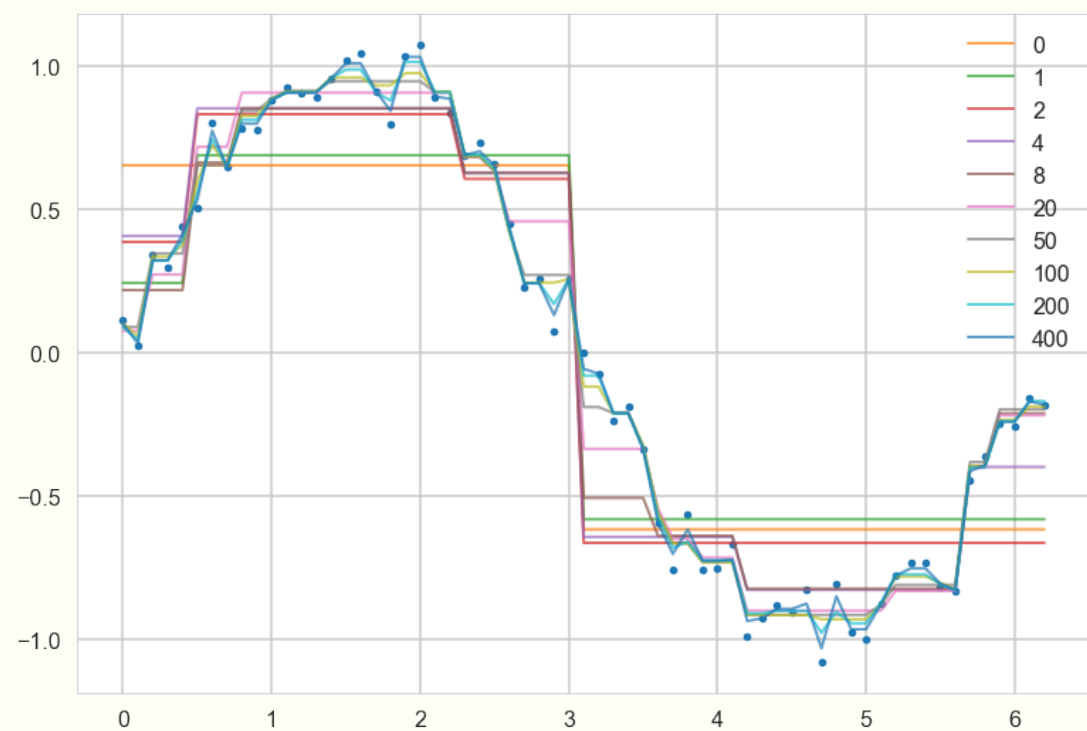
# Python Environments

# Hyper-parameter optimization

Eg, **Gradient boosting**: The basic idea is to fit the residuals of tree based regression models again and again. Peter Prettenhofer, who wrote sklearn's GBRT implementation writes in his pydata14 talk:

*I usually follow this recipe to tune the hyperparameters:*

- Pick `n_estimators` as large as (computationally) possible (e.g. 3000)
- Tune `max_depth`/`min_samples_leaf`, `learning_rate`, and `max_features` via grid search
- A lower `learning_rate` requires a higher number of `n_estimators`. Thus increase `n_estimators` even more and tune `learning_rate` again holding the other params fixed.



# Why is this bad? Or, why pipelines?

```
from sklearn.model_selection import GridSearchCV

vectorizer = TfidfVectorizer()
vectorizer.fit(text_train)

X_train = vectorizer.transform(text_train)
X_test = vectorizer.transform(text_test)

clf = LogisticRegression()
grid = GridSearchCV(clf, param_grid={'C': [.1, 1, 10, 100]}, cv=5)
grid.fit(X_train, y_train)
```

**The purpose of the pipeline is to assemble several steps that can be cross-validated together while setting different parameters.**

# Grid search on pipelines

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import fetch_20newsgroups

categories = [
    'alt.atheism',
    'talk.religion.misc',
]
data = fetch_20newsgroups(subset='train', categories=categories)
pipeline = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', SGDClassifier())
])
grid = {
    'vect__ngram_range': [(1, 1)],
    'tfidf__norm': ['l1', 'l2'],
    'clf__alpha': [1e-3, 1e-4, 1e-5]}

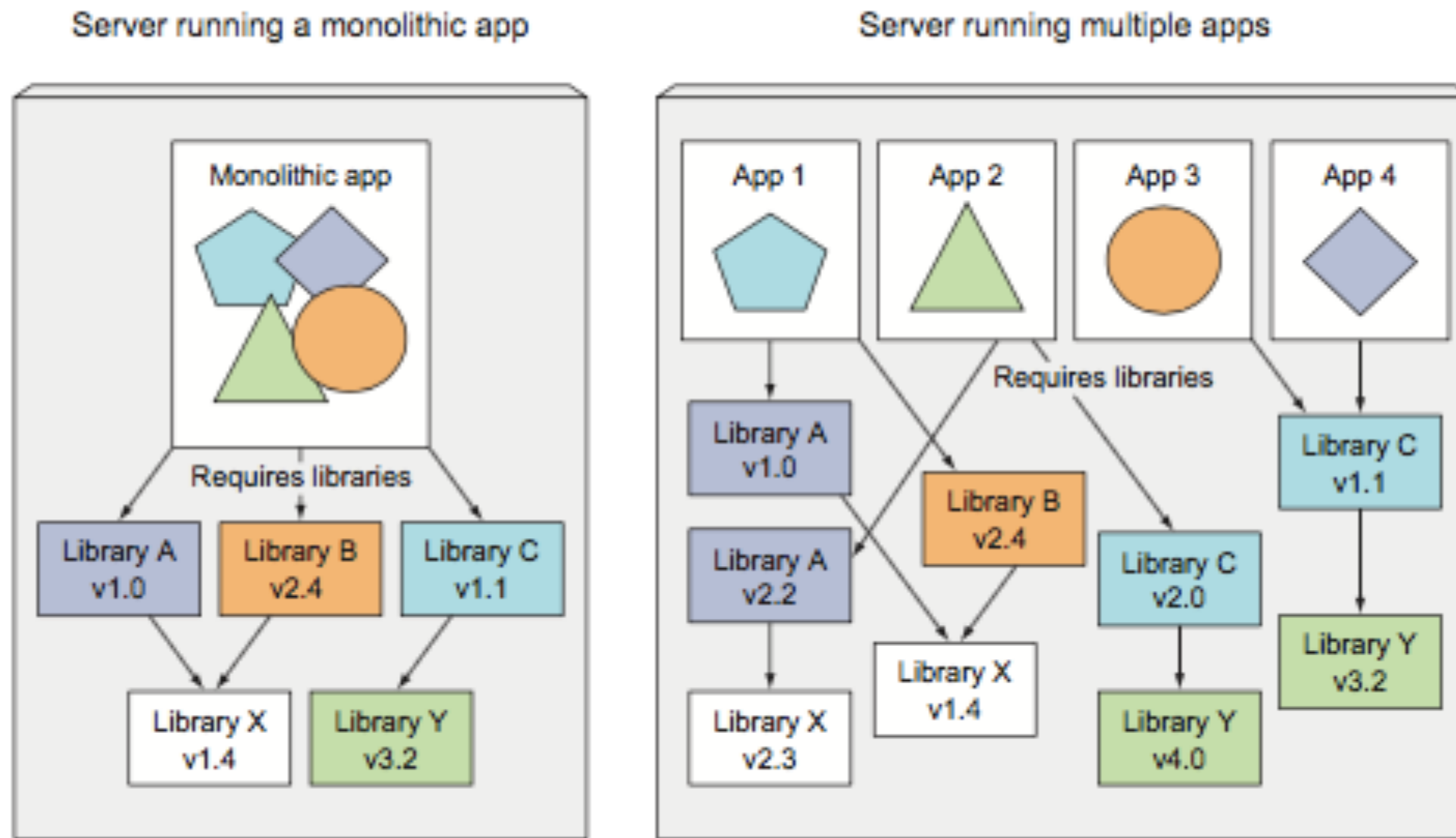
if __name__ == '__main__':
    grid_search = GridSearchCV(pipeline, grid, cv=5, n_jobs=-1)
    grid_search.fit(data.data, data.target)
    print("Best score: %0.3f" % grid_search.best_score_)
    print("Best parameters set:", grid_search.best_estimator_.get_params())
```

# ok, so you want to do this

## PERFORMANTLY, AND REPRODUCIBLY

- same random seed
- same programming environment on multiple machines (ideally same version of OS/python-conda stack/BLAS libraries, etc)
- then run the same code with a different parameter combination on each machine
- deal with the possible loss of some machines in this computation (they die, you got an amazon spot instance..)
- combine all the data output from these runs to make hyperparameter choices

# Programming environment: The multiple libraries problem



The solution

# Virtual Environments



But first:

# 1. Setting up your Python Infrastructure

# Installing Anaconda/Miniconda/Miniforge

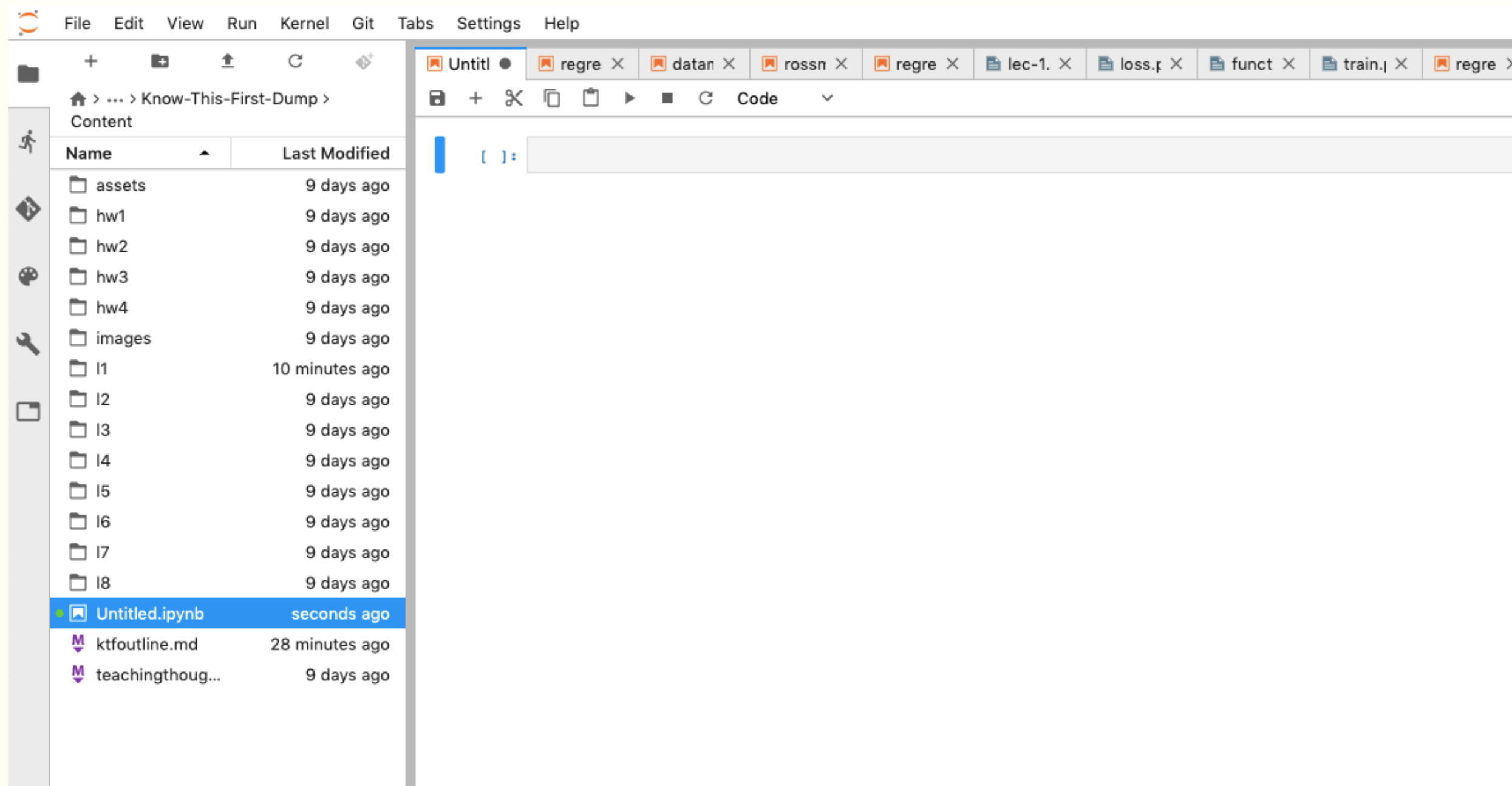
- If you are completely new, you can use anaconda, and you get a nice graphical user interface
- if you have some command line experience, you might want to use [Miniconda](#)
- Install miniconda on non m1 platforms if you like, and you get access to Anaconda Inc's default channel
- But you can (must if commercial and you dont want to pay) also leverage the community driven conda-forge channel using [miniforge](#)
- Use miniforge on a m1 mac

# How this workshop will run

- We are not going to bother with the initial anaconda/miniconda/miniforge installation: this will take too much time. If you don't have such a system, install it and let us know on the community forum. We can help you.
- Instead, we have set up a conda-forge based Jupyterlab installation for you at: <https://mybinder.org/v2/gh/univai-ghf/python-environments/HEAD>. This uses the mybinder.org service. Click on the link to have a python environment set up for you in the cloud. You will get a web browser that looks like this:



This screen is called the **Launcher**. Click on "Python" to get a new document window, called a **Jupyter Notebook** to this process.



You can now type in text boxes in the Jupyter Notebook, called **cells** in this new window. The left side is a file manager and is likely showing your home folder. This notebook is called `Untitled.ipynb`.

Type `1+1` in the text box and hit "Shift-Enter" or mouse-press the "Play icon" on the toolbar at the top.

FileEditViewRunKernelGitTabsSettingsHelp

+

+

+

↺

↻

✎

Untitled ● regre × datan × rossn × regre × lec-1. × loss.꜑ × funct × train.꜑ × regre ×

Content

Content

Name

▲

Last Modified

assets

9 days ago

hw1

9 days ago

hw2

9 days ago

hw3

9 days ago

hw4

9 days ago

images

9 days ago

I1

10 minutes ago

I2

9 days ago

I3

9 days ago

I4

9 days ago

I5

9 days ago

I6

9 days ago

I7

9 days ago

I8

9 days ago

Untitled.ipynb

seconds ago

ktfoutline.md

28 minutes ago

teachingthoug...

9 days ago

[ ]:

1 + 1

FileEditViewRunKernelGitTabsSettingsHelp

+

+

+

↺

↻

✎

Untitled ● regre × datan × rossn × regre × lec-1. × loss.꜑ × funct × train.꜑ × regre ×

Content

Content

Name

▲

Last Modified

assets

9 days ago

hw1

9 days ago

hw2

9 days ago

hw3

9 days ago

hw4

9 days ago

images

9 days ago

I1

11 minutes ago

I2

9 days ago

I3

9 days ago

I4

9 days ago

I5

9 days ago

I6

9 days ago

I7

9 days ago

I8

9 days ago

Untitled.ipynb

seconds ago

ktfoutline.md

29 minutes ago

teachingthoug...

9 days ago

[1]:

1 + 1

[1]:

2

[ ]:

# Jupyter cells have modes (and other GUI controls)

By default the cells are in Code mode. These can be changed to Markdown mode in the toolbar to enter text. The next picture shows some buttons and what they do.



JupyterLab interface showing a file browser on the left and a code editor on the right. The interface includes a menu bar (File, Edit, View, Run, Kernel, Git, Tabs, Settings, Help) and a toolbar with icons for file operations and cell management.

**File Browser (Left Panel):**

Name	Last Modified
BasicMLWithRegr...	a year ago
Corestuff	5 months ago
course.univ.ai	a month ago
courses	a month ago
docker-stacks	9 months ago
GoogleDrive	5 months ago
henbane	a day ago
hugobook	a year ago
jupyter-book	a year ago
KnowThisFirst	13 hours ago
kubdaskhw	23 days ago
mistletoe	2 days ago
univai-ai1-fall2019	24 days ago
univai-summersch...	8 months ago
univaihub	8 months ago
passwords.txt	9 months ago

**Code Editor (Right Panel):**

Python 3

Cell 1: `%load_ext autoreload`  
`%autoreload 2`

Cell 2: `%matplotlib inline`  
`import numpy as np`  
`import matplotlib.pyplot as plt`  
`import pandas as pd`

Cell 3: `df = pd.read_csv("regression3.csv")`  
`df.head()`

	x1	x2	y
0	-0.491130	-1.591899	-99.922032
1	-1.206935	0.120860	-39.136708
2	1.097253	-0.957712	31.252468
3	1.486125	0.475206	137.261690
4	-0.686401	-0.769527	-68.969280

Cell 4: `from kudzu.train import setup_data, train`

Cell 5: `dit = setup_data(df[['x1', 'x2']].values, df.y.values)`

Cell 8: `l,w,g,b, gb = train(dit, 200)`

Epoch 0  
pred shape (500,)  
>>> (500,)

**ValueError** Traceback (most recent call last)  
`<ipython-input-8-ccc5e6ce5bec> in <module>()  
----> 1 l,w,g,b, gb = train(dit, 200)`

**Handwritten Annotations:**

- new launcher files** (points to the file browser icon)
- running kernels** (points to the kernel icon)
- run cell** (points to the run button)
- Type of cell. click to change to markdown** (points to the cell type dropdown)
- add new cell** (points to the plus button)
- stop running cell** (points to the stop button)
- cut current cell** (points to the cut button)
- save doc.** (points to the save button)

# 2. Setting up Virtual Environments

# What is a virtual environment?

- It's own set of packages that do not conflict with other sets of packages
- thus we can have competing packages with different dependency versions isolated from each other
- space is not wasted: packages can be installed into multiple environments if the dependencies are met

# Creating a virtual environment

- `conda create --name environment-name [python=3.6]`
- `conda activate environment-name`
- `conda deactivate environment-name`
- `conda install <packagename>`

Eg:

```
conda create -n newe
```

```
conda activate newe
```

```
conda install numpy pandas ipykernel
```

```
conda deactivate newe
```

# Making sure environments are available on jupyterlab

- in the base installation, make sure you conda install `nb_conda_kernels`.
- this will depend on how you installed the base. But run the above command is needed
- in our binder system, this is already installed.
- now in every new environment make sure you install `ipykernel`.

+

Filter files by name

/

Name	Last Modified
newe2	2 hours ago
Y: environme...	2 hours ago
newfile.txt	2 hours ago
README.md	2 hours ago

jovyan@jupyter-univai-2dg

Launcher

newfile.txt

Notebook

Python 3 (ipykernel)

Python [conda env:newe]

Python [conda env:notebook] \*

Console

Python 3 (ipykernel)

Python [conda env:newe]

Python [conda env:notebook] \*

Other

Terminal

Text File

Markdown File

Python File

Show Contextual Help

# Capturing a virtual environment

`conda env export` will capture the exact dependencies. You can now redirect into a file, and use elsewhere on the same OS to recreate this environment.

The file is usually called `environment.yml`, so we do:

```
conda env export > environment.yml
```

If this file is in a particular folder, just type `conda env create` to create an environment with these packages.



# Using environment.yml files

- you can write your own environment.yml file to be much less tied to specific package versions, unless needed.

For example, for an environment newe2, we create a folder newe2, and in that folder, we create a environment.yml with the contents:

```
name: newe2
channels:
- conda-forge
dependencies:
- ipykernel
- matplotlib
- pandas
- numpy
- scipy
- seaborn
- scikit-learn
- tensorflow
- keras
```



# A conda env per project

- create a conda environment for each new project
- put an `environment.yml` in each project folder (like `newe2`) with a `name` line reflecting the folder name
- `conda|mamba env create` in project folder (like `newe2`)
- if not per project, at least have one for each new class, or class of projects
- environment for class of projects may grow organically, but capture its requirements from time-to-time.
- for example, on my dual-gpu machine, I have 3 separate environments for `pytorch`, `tensorflow`, and `jax`, as they even had slightly different CUDA requirements.

see [here](#)

# Mamba

- mamba is a version of conda that works with conda but is faster.
- use mamba to create and/or install
- use conda to activate/deactivate
- See [https://mamba.readthedocs.io/en/latest/user\\_guide/mamba.html](https://mamba.readthedocs.io/en/latest/user_guide/mamba.html)
- on our binder environment I can issue `mamba env create`. Its faster. But then I must use `conda activate newe2` to startup the new environment
- by default conda will use both default and conda-forge channels, mamba will use conda-forge. I found keras currently only installable with mamba.

```
# file name: environment.yml

# Give your project an informative name
name: project-name

# Specify the conda channels that you wish to grab packages from, in order of priority.
channels:
- defaults
- conda-forge

# Specify the packages that you would like to install inside your environment.
#Version numbers are allowed, and conda will automatically use its dependency
#solver to ensure that all packages work with one another.
dependencies:
- python=3.7
- conda
- scipy
- numpy
- pandas
- scikit-learn

# There are some packages which are not conda-installable. You can put the pip dependencies here instead.
- pip:
  - tqdm # for example only, tqdm is actually available by conda.
```

( from <http://ericmjl.com/blog/2018/12/25/conda-hacks-for-data-science-efficiency/> )

# More information

- <https://carpentries-incubator.github.io/introduction-to-conda-for-data-scientists/>
- <https://goodresearch.dev/setup.html> , which is part of the excellent book
- <https://goodresearch.dev/index.html>

# 3. Structure

# The Importance of Structure

- one might as well use the one env per project or set-of-projects structure to organize work
- it is really important to organize your data science work well
- a good tool for this is cookiecutter, which sets up a template folder structure for you. Install by `pip install cookiecutter` in your base.
- you install a cookiecutter by doing `cookiecutter source`.

## Two nice cookiecutters

- <https://github.com/patrickmineault/true-neutral-cookiecutter>
- Install via: `cookiecutter gh:patrickmineault/true-neutral-cookiecutter`
- <https://drivendata.github.io/cookiecutter-data-science/>
- Install via: `cookiecutter gh:drivendata/cookiecutter-data-science`

# True Neutral Cookiecutter

```
graph LR; root[ ] --- data; root --- doc; root --- results; root --- scripts; root --- src; root --- tests; root --- gitignore[.gitignore]; root --- envyaml[environment.yml]; root --- readmedoc[README.md]; root --- setuppy[setup.py]; src --- src_initpy[src/__init__.py];
```

- data
- doc
- results
- scripts
- src
  - `__init__.py`
- tests
- .gitignore
- environment.yml
- README.md
- setup.py



# An example

- We do: `cookiecutter gh:patrickmineault/true-neutral-cookiecutter`
- name the project `perceptron`
- create the conda environment:
- `conda create --name perceptron; conda activate perceptron;`  
`mamba install ipykernel numpy tensorflow keras` or do `mamba env create` with an appropriate environment file
- then do `cd perceptron; pip install -e .` which creates the `src` directory loadable into python

# Best practices for use

- now we can do development with both the notebook and files. In a notebook cell put the following to have the notebook automatically reload the file when it changes.

```
%load_ext autoreload  
%autoreload 2
```

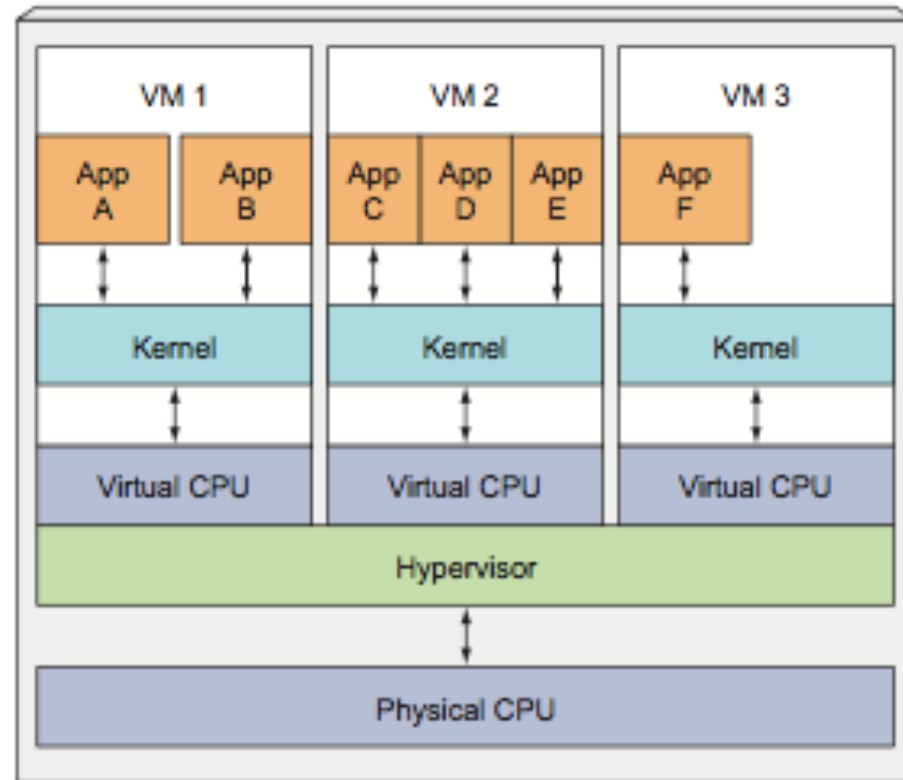
- refactor anything repeated multiple times to python files with functions in them. Notebooks should be very readable
- output all intermediate files into data or results while writing your pipelines: files from train test splits, parameter values and results, etc
- future you will thank current you.

4. Extra

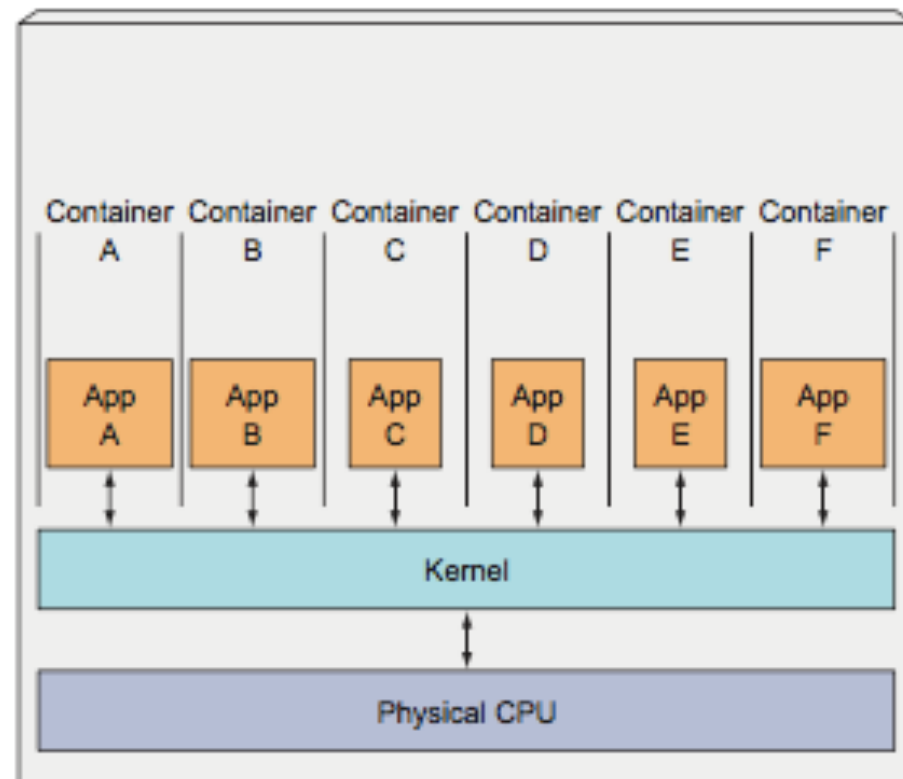
# Going even further: Docker

- More than python libs
- C-library, BLAS, linux kernel, etc
- we could use virtual machines (VMs) like vmware/virtualbox/lvm
- but these are heavy duty, OS level "hypervisor"s
- more general, but resource hungry

Apps running in multiple VMs



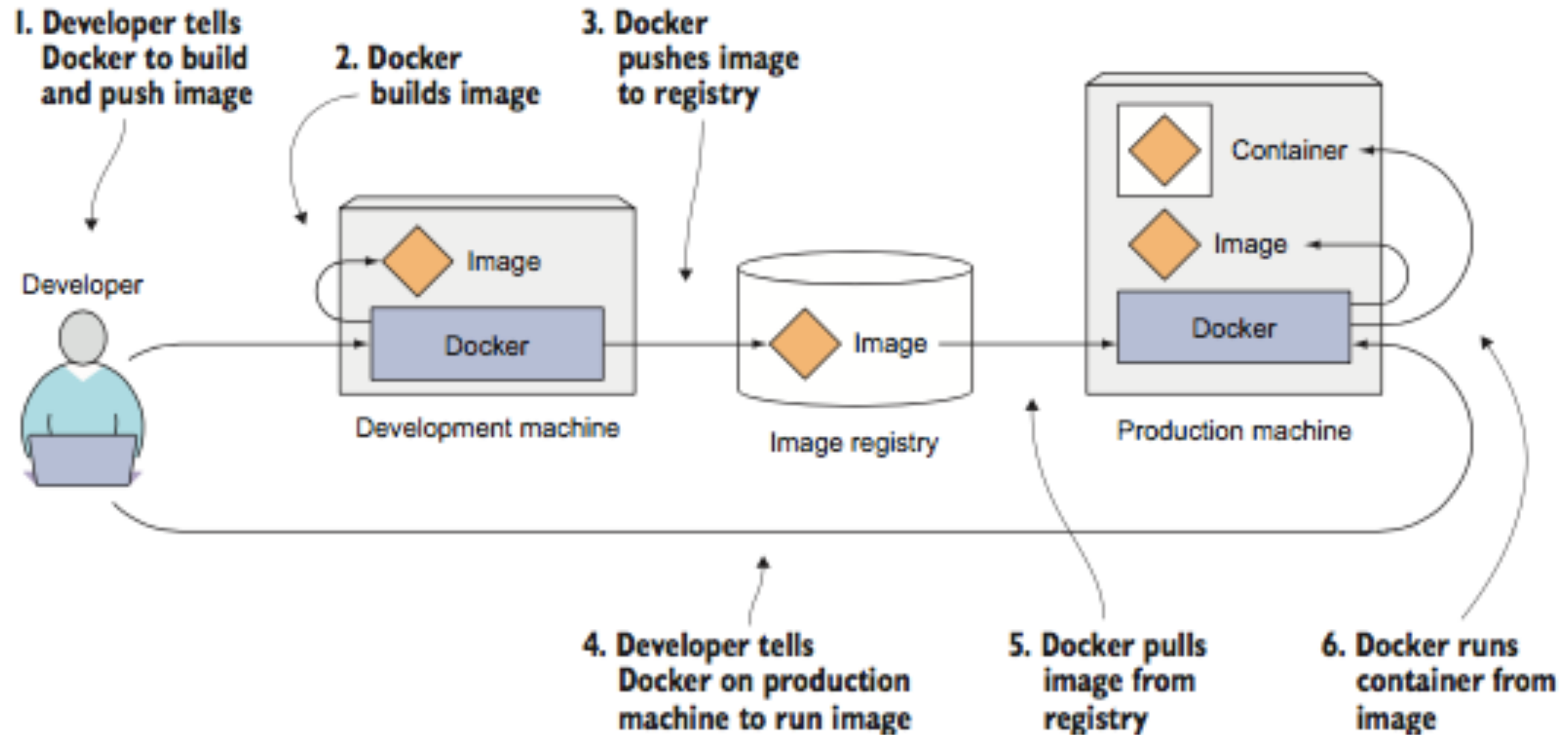
Apps running in isolated containers



# Containers vs Virtual Machines

- containers provide process isolation, process throttling
- but work at library and kernel level, and can access hardware more easily
- hardware access important for gpu access
- containers can run on VMS, this is how docker runs on mac, and on many cloud providers. They can also run on a host linux OS on bare metal

# Docker Architecture



# Docker images

- docker is linux only, but other OS's now have support
- allow for environment setting across languages and runtimes
- can be chained together to create outcomes
- base image is a linux (full) image, others are just layers on top
- side benefit: integration testing and testing for CI

# An Example

base notebook -> minimal notebook -> scipy notebook ->  
tensorflow notebook

```
# Copyright (c) Jupyter Development Team.
# Distributed under the terms of the Modified BSD License.
ARG BASE_CONTAINER=jupyter/scipy-notebook
FROM $BASE_CONTAINER

LABEL maintainer="Jupyter Project <jupyter@googlegroups.com>"

# Install Tensorflow
RUN conda install --quiet --yes \
    'tensorflow=1.12*' \
    'keras=2.2*' && \
    conda clean -tipsy && \
    fix-permissions $CONDA_DIR && \
    fix-permissions /home/$NB_USER
```

```
ARG BASE_CONTAINER=jupyter/minimal-notebook
FROM $BASE_CONTAINER
...
# ffmpeg for matplotlib anim
RUN apt-get update && \
    apt-get install -y --no-install-recommends ffmpeg && \
    rm -rf /var/lib/apt/lists/*
RUN conda install --quiet --yes \
    'conda-forge::blas*=openblas' \
    'ipywidgets=7.4*' \
    'pandas=0.23*' \
    'numexpr=2.6*' \
    'matplotlib=2.2*' \
    'scipy=1.1*' \
    'seaborn=0.9*' \
    'scikit-learn=0.20*' \
    'scikit-image=0.14*' \
    'sympy=1.1*' \
    'cython=0.28*' \
    'patsy=0.5*' \
    'statsmodels=0.9*' \
    'cloudpickle=0.5*' \
    'dill=0.2*' \
    'numba=0.38*' \
    'bokeh=0.13*' \
    'sqlalchemy=1.2*' \
    'hdf5=1.10*' \
    'h5py=2.7*' \
    'vincent=0.4.*' \
    'beautifulsoup4=4.6.*' \
    'protobuf=3.*' \
    'xlrd' && \
    conda remove --quiet --yes --force qt pyqt && \
    ...

# Install facets which does not have a pip or conda package at the moment
RUN cd /tmp && \
    git clone https://github.com/PAIR-code/facets.git && \
    cd facets && \
    jupyter nbextension install facets-dist/ --sys-prefix && \
    cd && ...
```



```

ARG
BASE_CONTAINER=ubuntu:bionic-20180526@sha256:c8c275751219dadad8fa56b3ac41ca6c
b22219ff117ca98fe82b42f24e1ba64e
FROM $BASE_CONTAINER
ARG NB_USER="jovyan"
...
USER root
RUN apt-get update && apt-get -yq dist-upgrade \
  && apt-get install -yq --no-install-recommends \
    wget \
    ...
RUN echo "en_US.UTF-8 UTF-8" > /etc/locale.gen && \
  locale-gen
ENV CONDA_DIR=/opt/conda \
  NB_USER=$NB_USER \
  ...
ADD fix-permissions /usr/local/bin/fix-permissions
RUN groupadd wheel -g 11 && \
  useradd -m -s /bin/bash -N -u $NB_UID $NB_USER && \
  ...
USER $NB_UID
...
ENV MINICONDA_VERSION 4.5.11
RUN cd /tmp && \
  wget --quiet https://repo.continuum.io/miniconda/Miniconda3-$(
    MINICONDA_VERSION)-Linux-x86_64.sh && \
  echo "e1045ee415162f944b6aebfe560b8fee *Miniconda3-$(MINICONDA_VERSION)-
    Linux-x86_64.sh" | md5sum -c - && \
  /bin/bash Miniconda3-$(MINICONDA_VERSION)-Linux-x86_64.sh -f -b -p
  $CONDA_DIR && \
  ...
RUN conda install --quiet --yes 'tini=0.18.0' && \
  ...
RUN conda install --quiet --yes \
  'notebook=5.7.2' \
  'jupyterhub=0.9.4' \
  'jupyterlab=0.35.4' && ...
USER root
EXPOSE 8888
ENTRYPOINT ["tini", "-g", "--"]
CMD ["start-notebook.sh"]
COPY start.sh /usr/local/bin/
...
USER $NB_UID

```

```

ARG BASE_CONTAINER=jupyter/base-notebook
FROM $BASE_CONTAINER

```

```

LABEL maintainer="Jupyter Project <jupyter@googlegroups.com>"

```

```

USER root

```

```

# Install all OS dependencies for fully functional notebook server
RUN apt-get update && apt-get install -yq --no-install-recommends \
  build-essential \
  emacs \
  git \
  inkscape \
  jed \
  libsm6 \
  libxext-dev \
  libxrender1 \
  lmodern \
  netcat \
  pandoc \
  python-dev \
  texlive-fonts-extra \
  texlive-fonts-recommended \
  texlive-generic-recommended \
  texlive-latex-base \
  texlive-latex-extra \
  texlive-xetex \
  unzip \
  nano \
  && rm -rf /var/lib/apt/lists/*

```

```

# Switch back to jovyan to avoid accidental container runs as root
USER $NB_UID

```

# repo2docker and binder

- building docker images is not dead simple
- the Jupyter folks created [repo2docker](#) for this.
- provide a github repo, and repo2docker makes a docker image and uploads it to the docker image repository for you
- [binder](#) builds on this to provide a service where you provide a github repo, and it gives you a working jupyterhub where you can "publish" your project/demo/etc
- built repo's can be used with multiple software such as jupyterhub/dask/kubernetes, etc