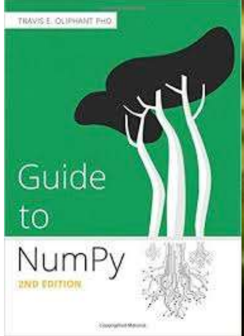
Python for Data Science using Anaconda

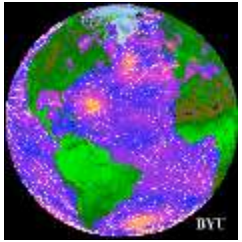
TDWI Accelerate

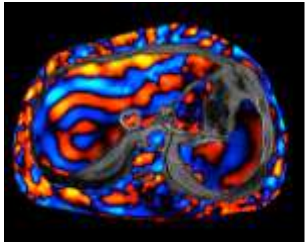
Travis E. Oliphant
President, Chief Data Scientist, Co-founder

A bit about me

- PhD 2001 from Mayo Clinic in Biomedical Engineering (Ultrasound and MRI)
- MS/BS degrees in Elec. Comp. Engineering
- Creator and Developer of **SciPy** (1999-2009)
- Professor at BYU (2001-2007)
- Author of **NumPy** (2005-2012)
- Started **Numba** (2012)
- Founding Chair of **NumFOCUS** / **PyData**
- Previous Python Software Foundation Director
- Co-founder of Continuum Analytics
- CEO => President, Chief Data Scientist







NumPy

SciPy

PyData

NUMFOCUS
OPEN CODE = BETTER SCIENCE

python

2012 - Created Two Orgs for Sustainable Open Source

Company

ANACONDA
Powered by Continuum Analytics

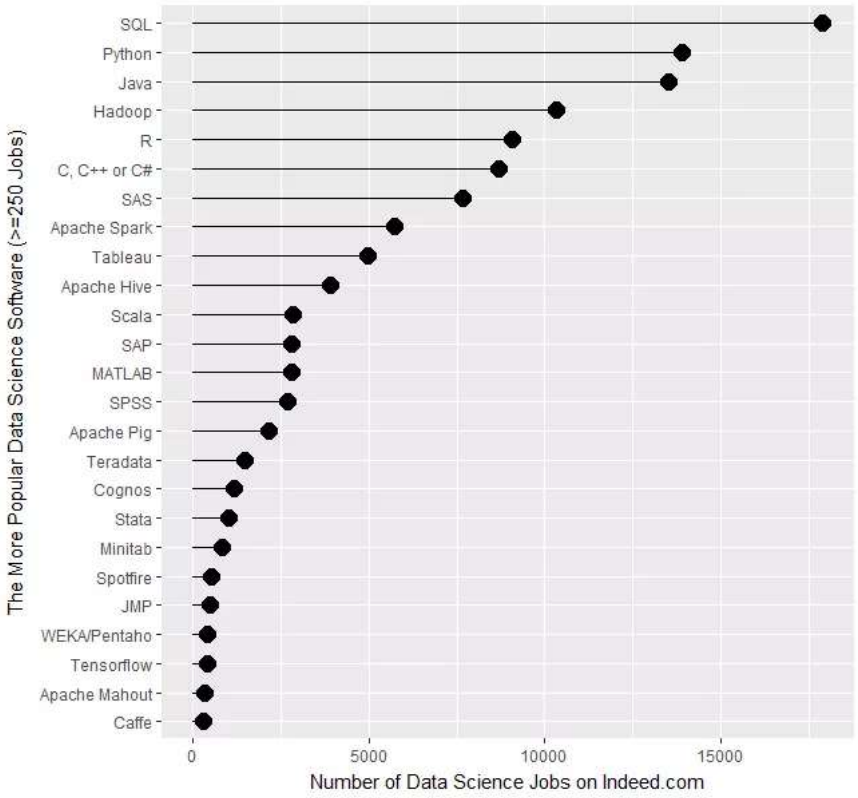
Enterprise **software, support** and **services** to empower people who change the world to get rapid insight from **all** of their data — built on **open-source** that we contribute to and sustain.

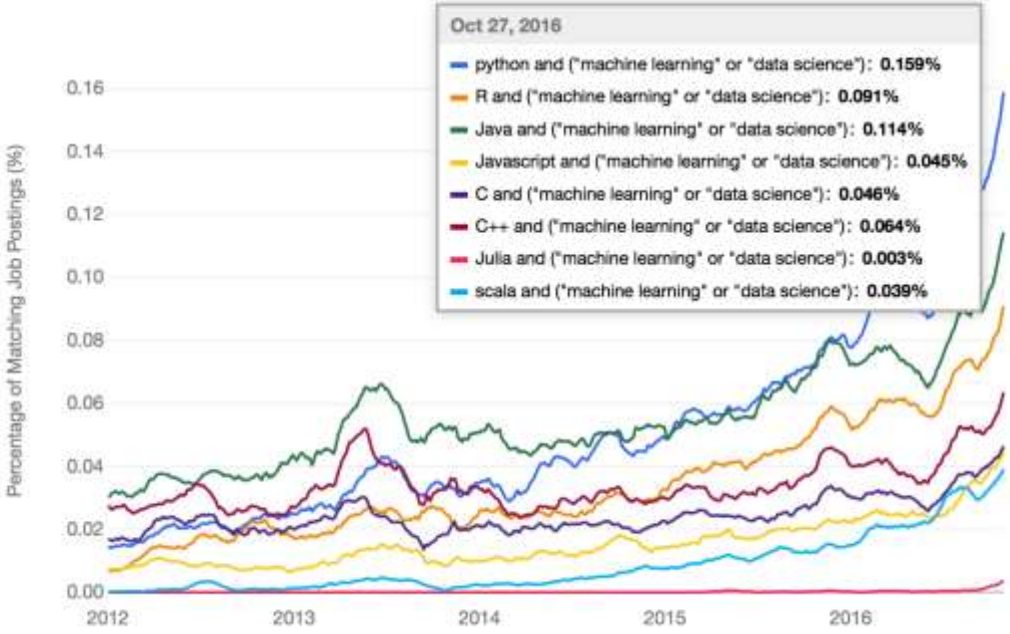
Community

PyData

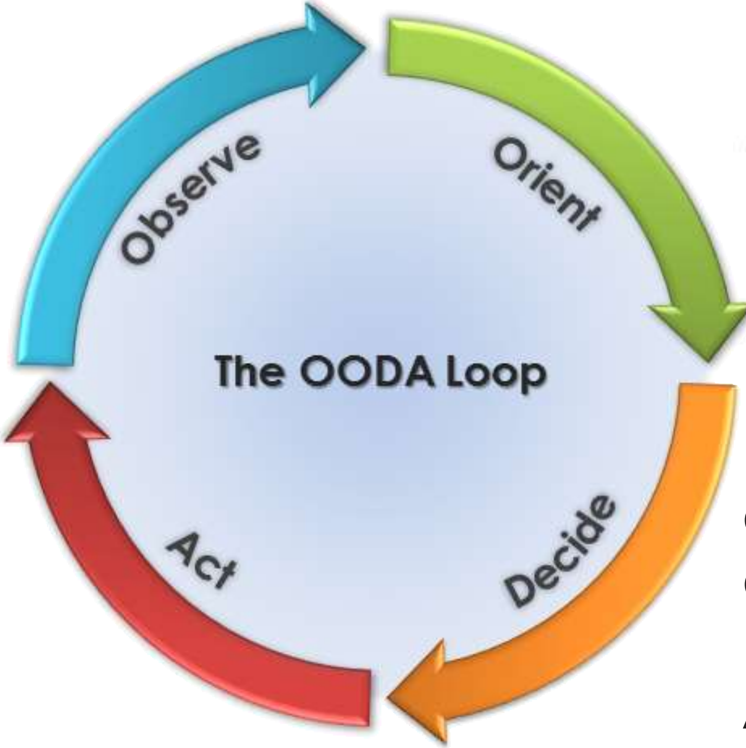
NUMFOCUS
OPEN CODE = BETTER SCIENCE

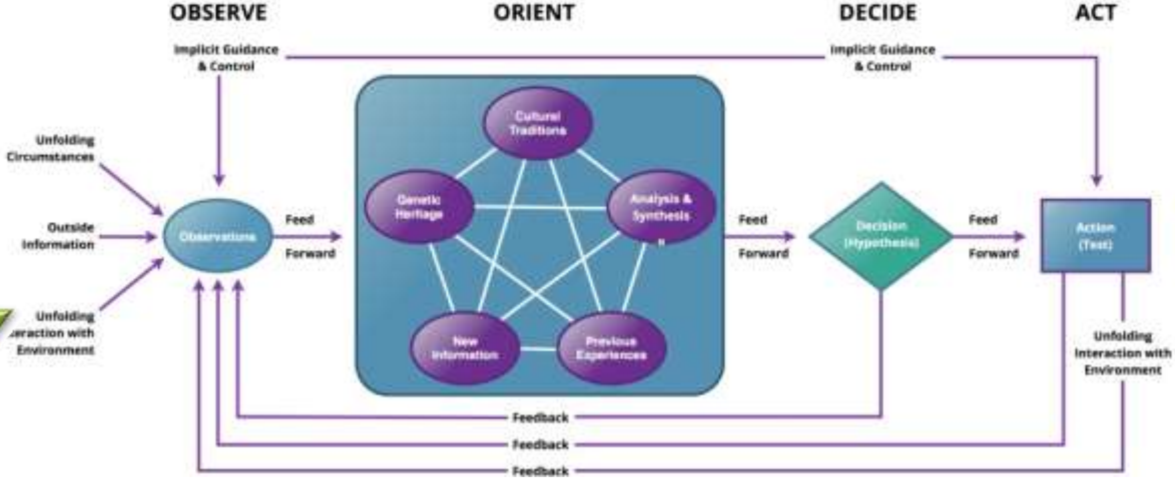
Data Science is growing with Python leading the way





Dealing with a lot of information: OODA loop





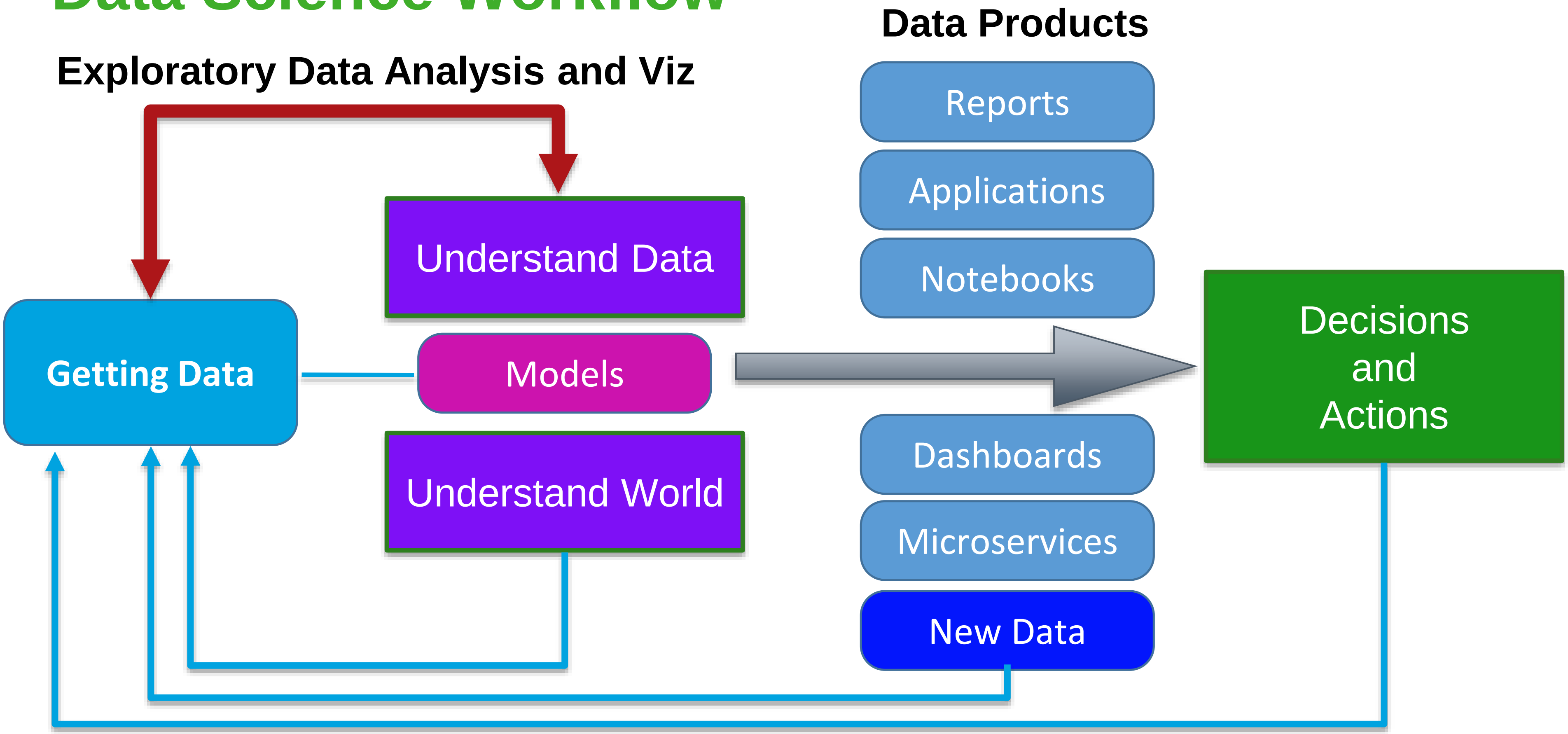
Observe — Gather data

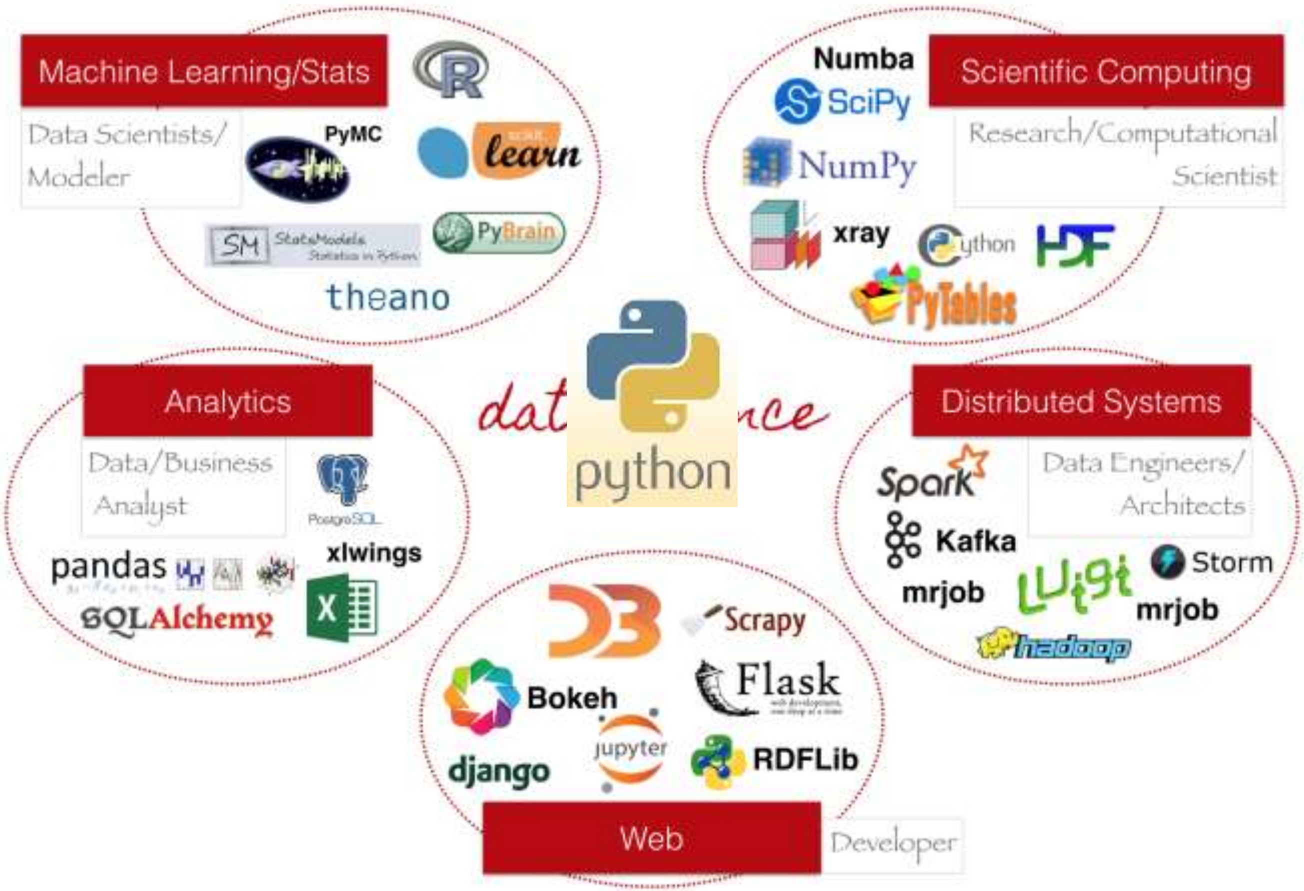
Orient — How will you interpret the data

Decide — Make a decision

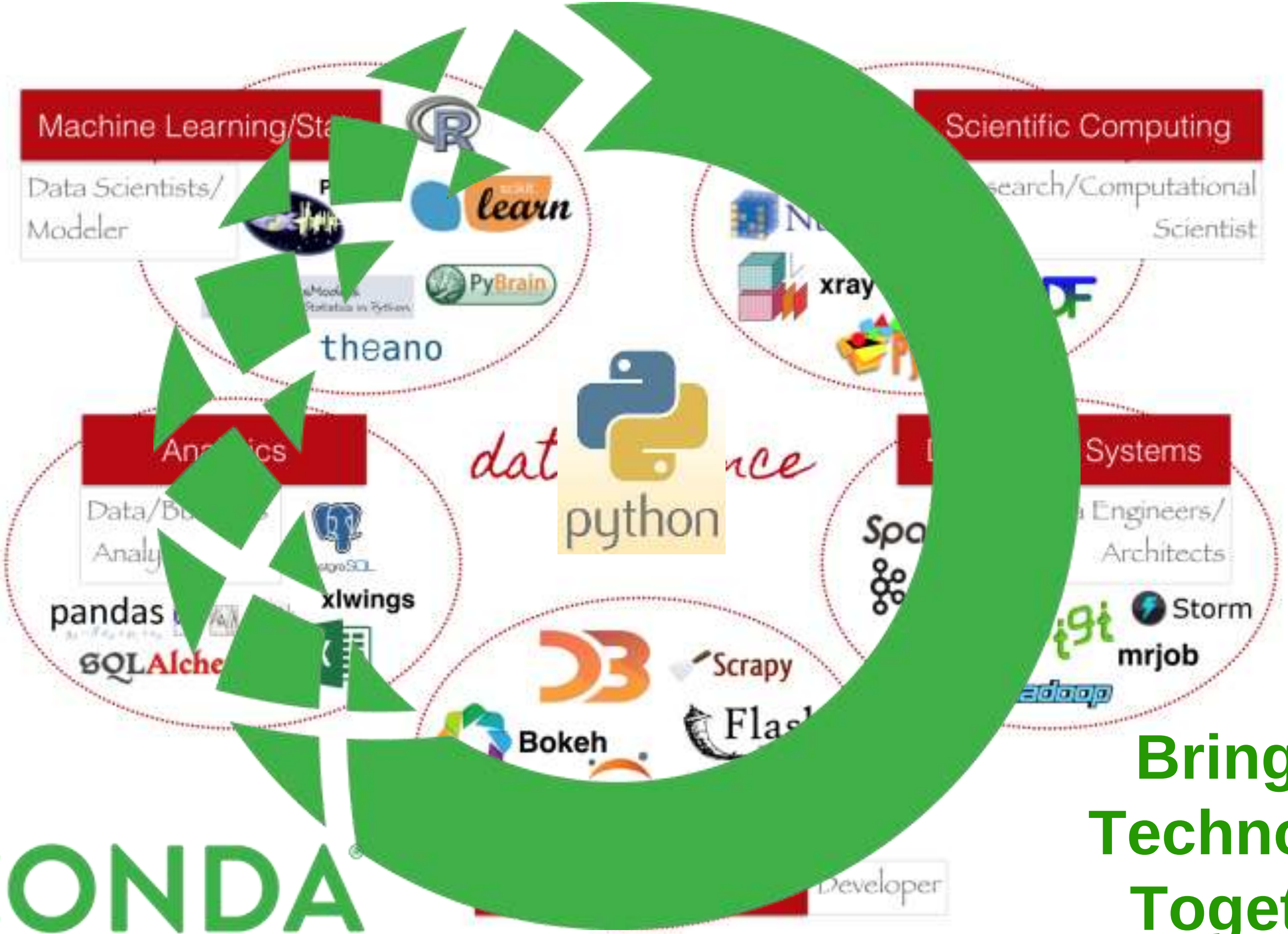
Act — Execute decision and gather feedback

Data Science Workflow





Machine Learning/Stats
Data Scientists/ Modeler
PyMC
learn
StatsModels Statistics in Python
PyBrain
theano
Scientific Computing
Research/Computational Scientist
Numba
SciPy
NumPy
xray
Cython
HDF
PyTables
Analytics
Data/Business Analyst
PostgreSQL
xlwings
pandas
SQLAlchemy
python
Distributed Systems
Data Engineers/ Architects
Spark
Kafka
mrjob
luigi
Storm
mrjob
hadoop
Scrapy
Bokeh
Flask
django
jupyter
RDFLib
Web
Developer



Bringing Technology Together

Accelerate adoption of Data Science

http://continuum.io/downloads

ANACONDA®

PYTHON & R OPEN SOURCE ANALYTICS

NumPy	SciPy	Pandas	Scikit-learn	Jupyter/IPython	
Numba	Matplotlib	Spyder	TensorFlow	Cython	Theano
Scikit-image	NLTK	Dask	Caffe	dplyr	shiny
ggplot2	tidyr	caret	PySpark	**& 1000+ packages**	

CONDA®

ANACONDA

http://continuum.io/downloads

Trusted by Industry Leaders

Financial Services
Risk Mgmt, Quant modeling, Data exploration and processing, algorithmic trading, compliance reporting

Government
Fraud detection, data crawling, web & cyber data analytics, statistical modeling

Healthcare & Life Sciences
Genomics data processing, cancer research, natural language processing for health data science

High Tech
Customer behavior, recommendations, ad bidding, retargeting, social media analytics

Retail & CPG
Engineering simulation, supply chain modeling, scientific analysis

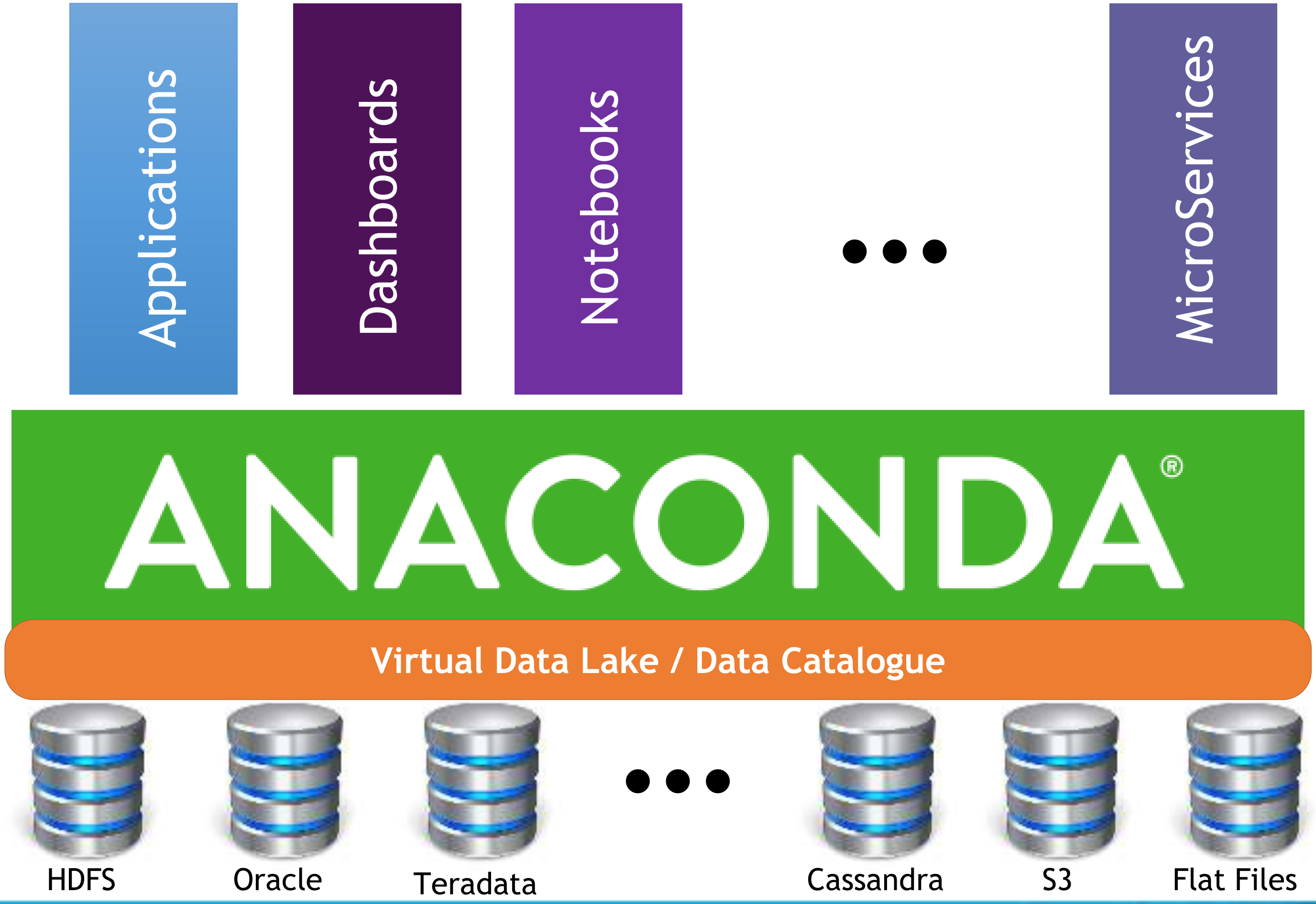
Oil & Gas
Pipeline monitoring, noise logging, seismic data processing, geophysics

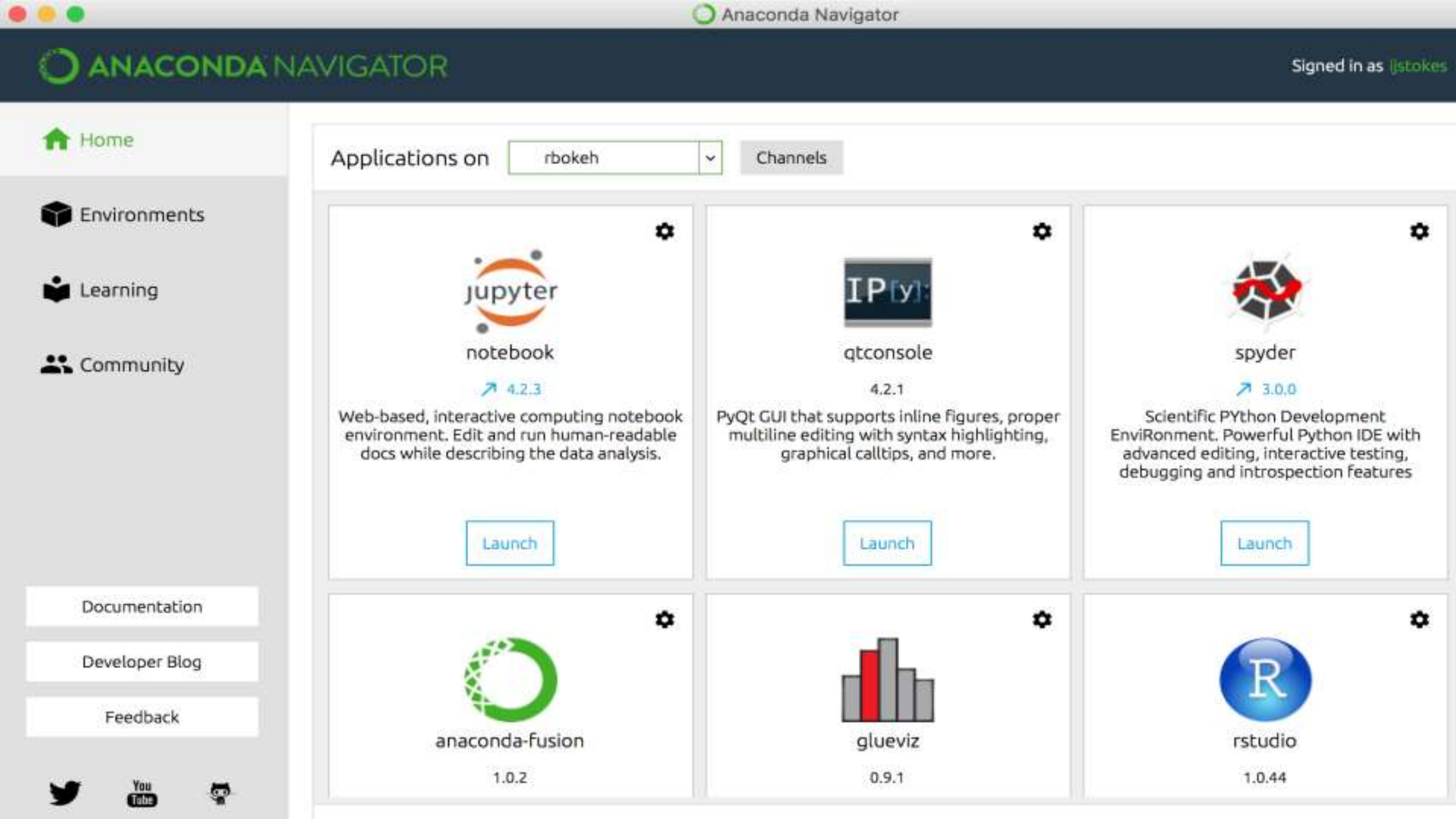
LinkedIn	Microsoft	DARPA
Kaiser Permanente	J.P.Morgan	macy's
Boeing	DuPont	Los Alamos National Laboratory EST. 1943
GEICO	Bank of America	appnexus
Cisco	Siemens	CTC Chicago Trading Company
Philips	NASA	amazon.com

Insight
Decisions
Actions
Results
Creations

Open Data Science

DATA

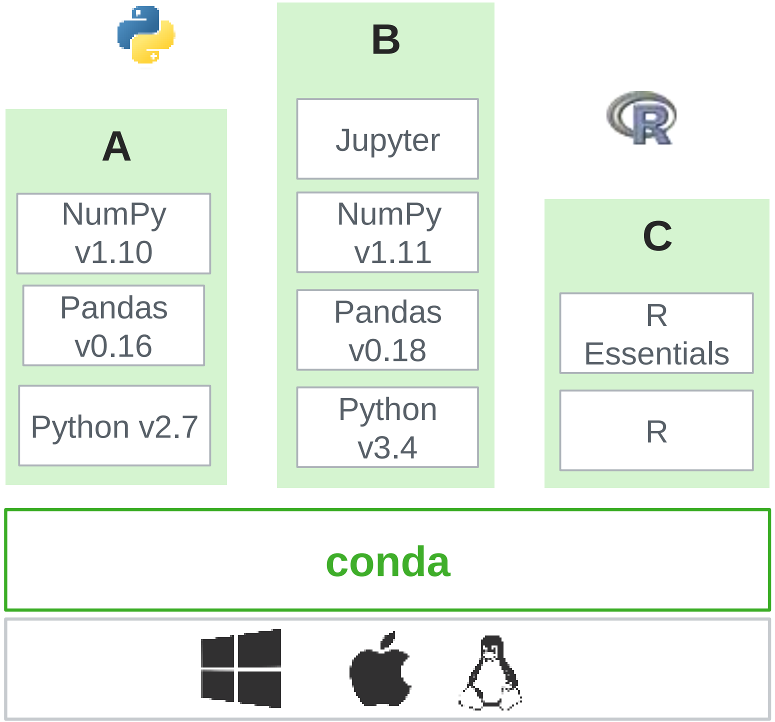



Anaconda Navigator
ANACONDA NAVIGATOR
Signed in as ljstokes
Home
Environments
Learning
Community
Documentation
Developer Blog
Feedback
Applications on
rbokeh
Channels
jupyter
notebook
4.2.3
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
IP[y]:
qtconsole
4.2.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch
spyder
3.0.0
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch
anaconda-fusion
1.0.2
glueviz
0.9.1
R
rstudio
1.0.44

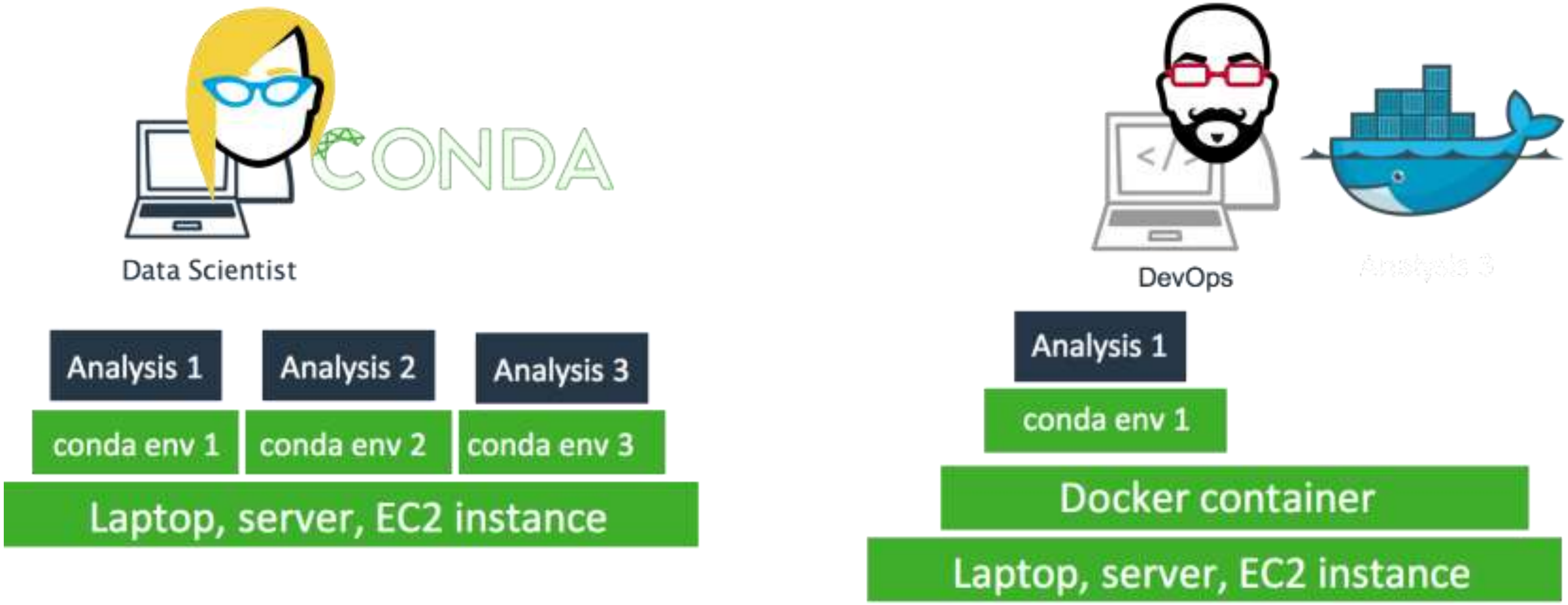
Conda Sandboxing Technology

- **Language** independent
- **Platform** independent
- No special privileges required
- No VMs or containers
- Enables:
  - **Reproducibility**
  - **Collaboration**
  - **Scaling**

"conda – package everything"



Conda for Deployment



Development → Deployment

CONDA

```
$ conda install python=2.7
$ conda install pandas
$ conda install -c r r
$ conda install -c conda-forge tensorflow
```

Install dependencies

```
name: myenv
channels:
  - chdoig
  - r
  - foo
dependecies:
  - python=2.7
  - r
  - r-ldavis
  - pandas
  - mongodb
  - spark=1.5
  - pip
  - pip:
    - flask-migrate
    - bar=1.4
```

```
$ conda env create
$ source activate myenv
```

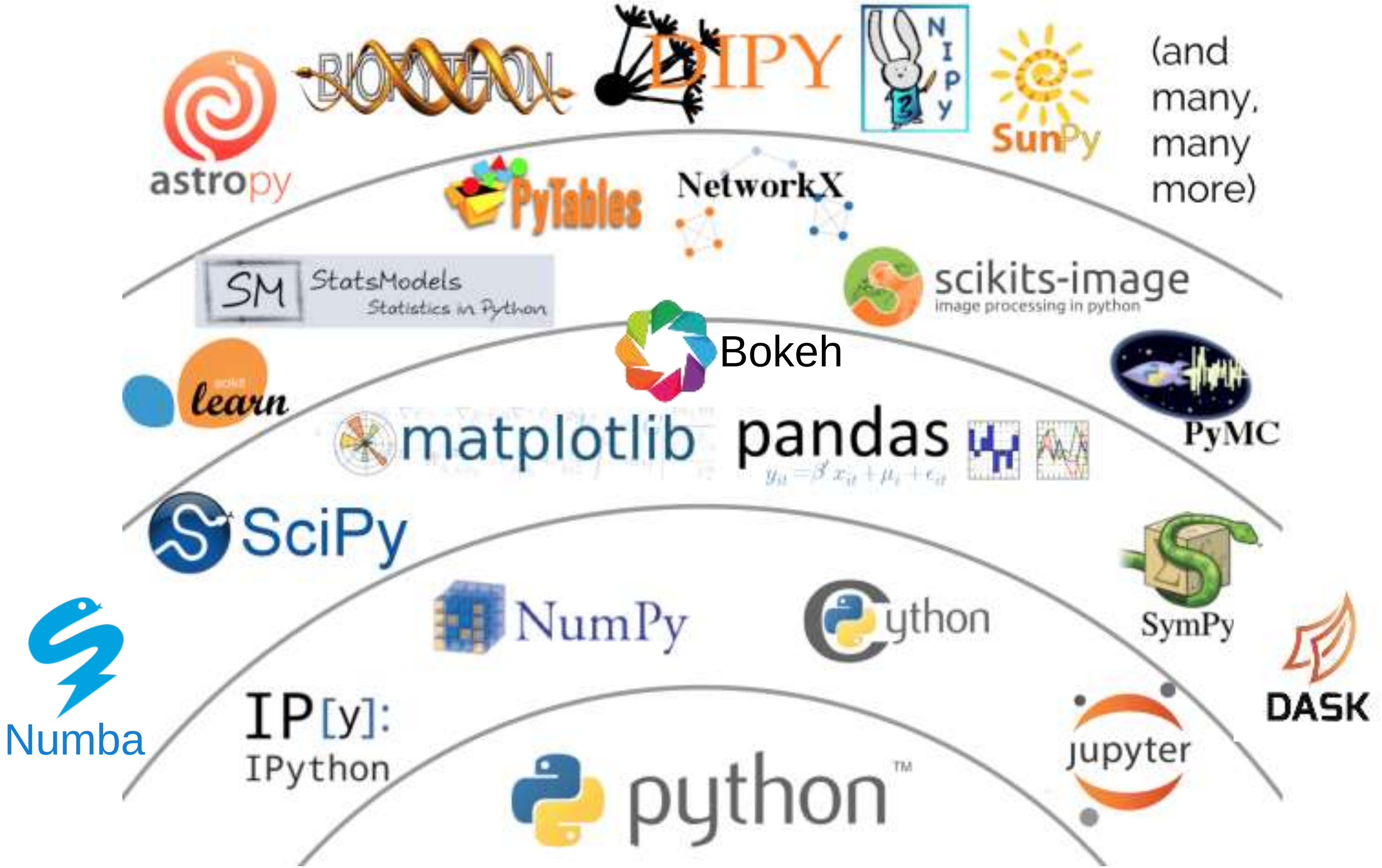
Manage multiple environments

```
$ anaconda-project run plot —show
```

Deploy an interactive visualization

ANACONDA
Powered by Continuum Analytics

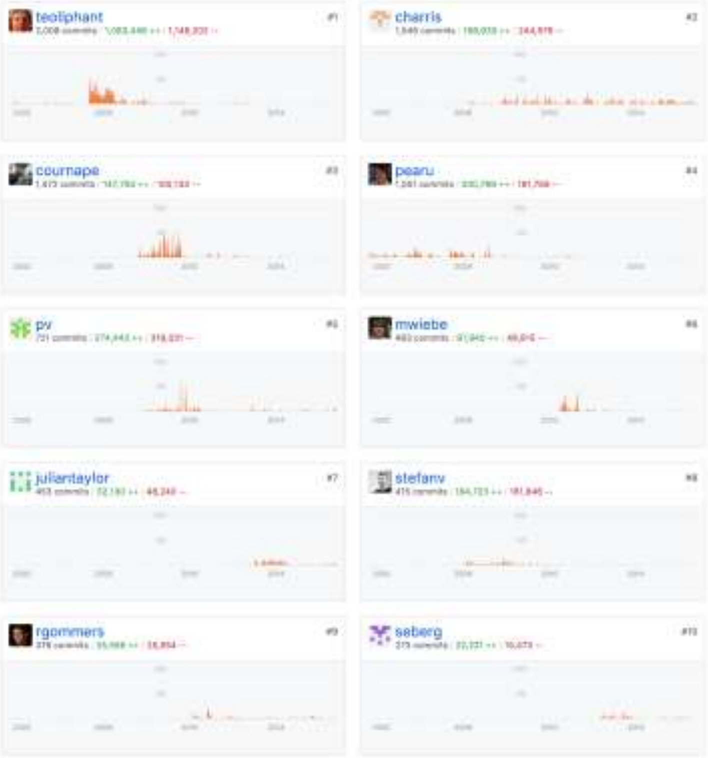
Python Ecosystem


astropy
BIOPYTHON
DIPY
NIPY
SunPy
(and many, many more)
PyTables
NetworkX
StatsModels
Statistics in Python
scikits-image
image processing in python
Bokeh
learn
matplotlib
pandas
PyMC
SciPy
NumPy
Cython
SymPy
DASK
Numba
IP[y]:
IPython
python™
jupyter

An impressive community effort

NumPy ~ 509 contributors

SciPy ~ 478 contributors



NumPy

Without NumPy

```
from math import sin, pi

def sinc(x):
    if x == 0:
        return 1.0
    else:
        pix = pi*x
        return sin(pix)/pix

def step(x):
    if x > 0:
        return 1.0
    elif x < 0:
        return 0.0
    else:
        return 0.5
```

functions.py

```
>>> import functions as f
>>> xval = [x/3.0 for x in
range(-10,10)]
>>> yval1 = [f.sinc(x) for x
in xval]
>>> yval2 = [f.step(x) for x
in xval]
```

Python is a great language but needed a way to operate quickly and cleanly over multi-dimensional arrays.

```python
from numpy import sin, pi
from numpy import vectorize
import functions as f

vsinc = vectorize(f.sinc)
def sinc(x):
    pix = pi*x
    val = sin(pix)/pix
    val[x==0] = 1.0
    return val

vstep = vectorize(f.step)
def step(x):
    y = x*0.0
    y[x>0] = 1
    y[x==0] = 0.5
    return y
```

functions2.py

With NumPy

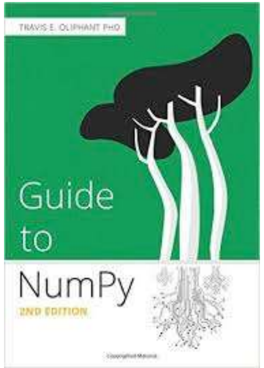
```python
>>> import functions2 as f
>>> from numpy import *
>>> x = r_[-10:10]/3.0
>>> y1 = f.sinc(x)
>>> y2 = f.step(x)
```

Offers N-D array, element-by-element functions, and basic random numbers, linear algebra, and FFT capability for Python

http://numpy.org

Fiscally sponsored by NumFOCUS

NumPy: an Array Extension of Python

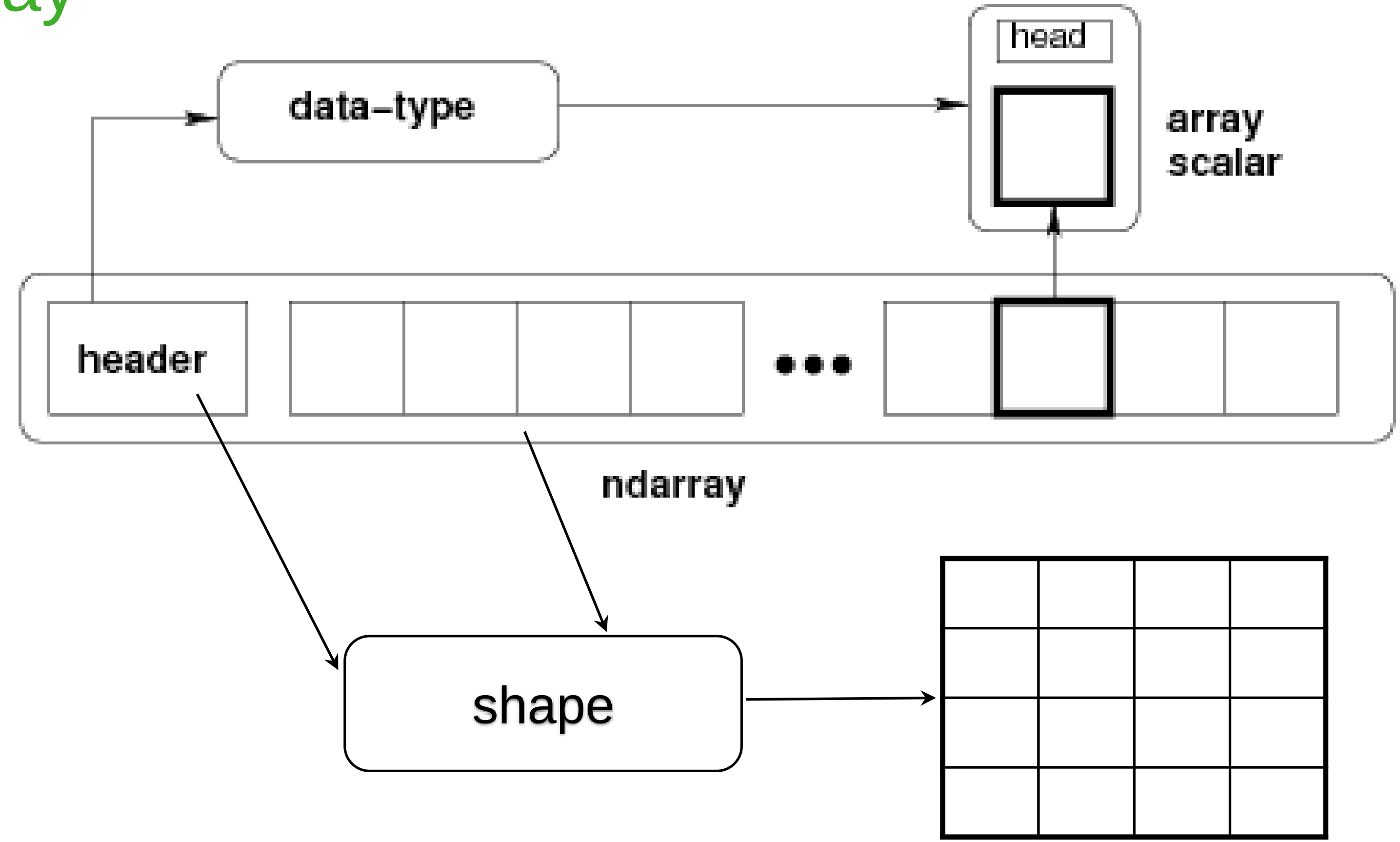


- Data: the array object
  - slicing and shaping
  - data-type map to Bytes
- Fast Math (ufuncs):
  - vectorization
  - broadcasting
  - aggregations

NumPy Array

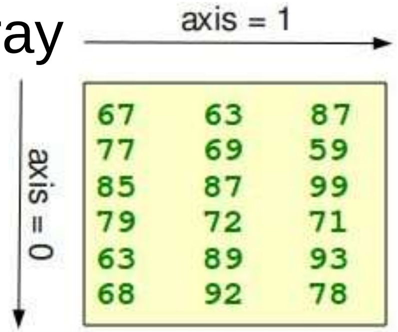
Key Attributes

- dtype
- shape
- ndim
- strides
- data



NumPy Examples

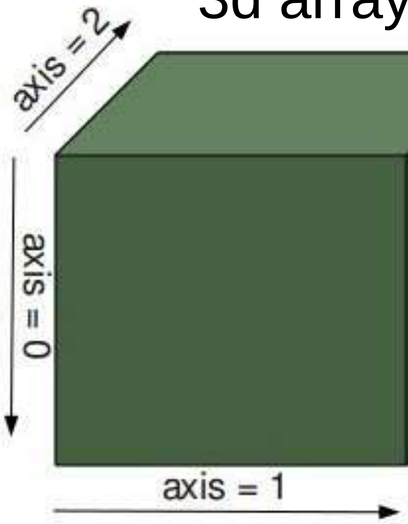
2d array




```
import numpy as np

x = np.array([[67, 63, 87],
              [77, 69, 59],
              [85, 87, 89],
              [79, 72, 71],
              [63, 89, 93],
              [68, 92, 78]])
print x.sum(axis=0), x.sum(axis=1)
```

[439 472 477]
[217 205 261 222 245 238]

3d array

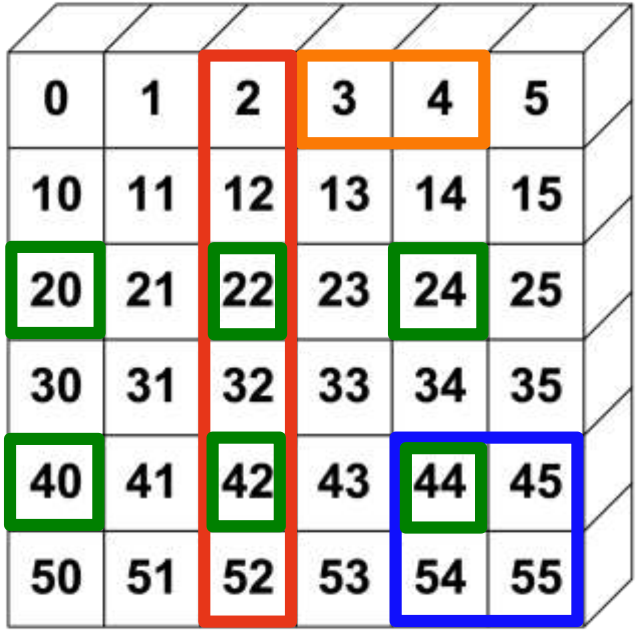



```
y = 3*np.random.randn(10,20,30)+10
print y.mean(), y.std()
```

9.98330639789 2.96677717122

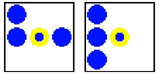
NumPy Slicing (Selection)

```
>>> a[0,3:5]
array([3, 4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20, 22, 24],
       [40, 42, 44]])
```

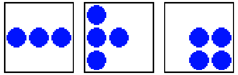


Conway's game of Life

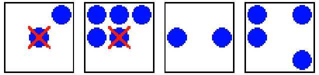
- Dead cell with exactly 3 live neighbors will come to life

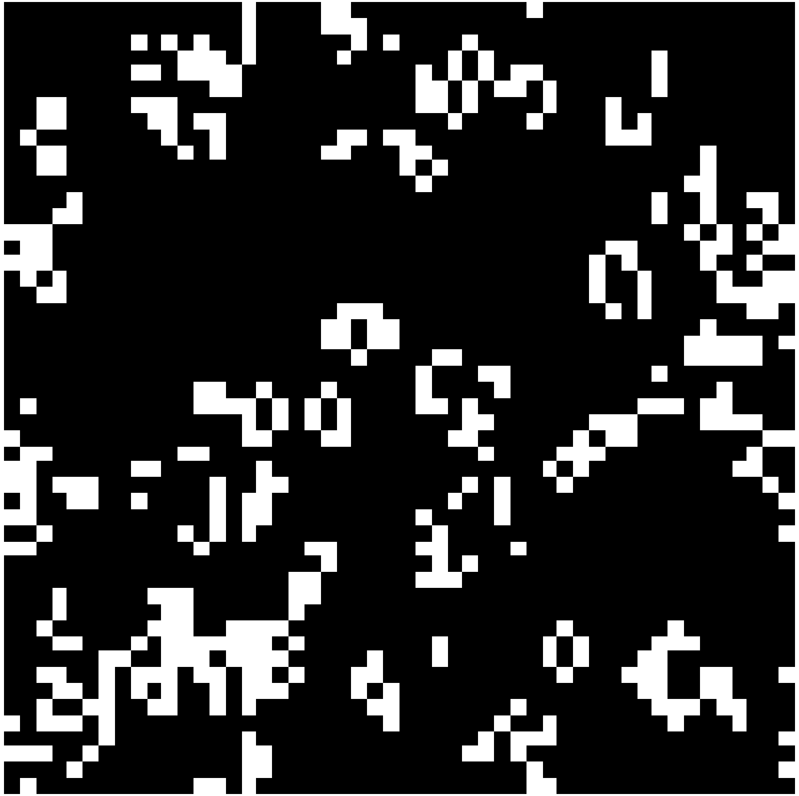


- A live cell with 2 or 3 neighbors will survive



- With too few or too many neighbors, the cell dies





Conway's Game of Life in NumPy

Initialization

```
size = 100
GRID = (rand(size,size) > 0.5).astype(uint8)
# The world is round
indx = r_[0:size]
up = roll(indx, -1)
down = roll(indx, 1)
```

Update Step

```
neighbors = GRID[up,:] + GRID[down,:] + GRID[:,up] + GRID[:,down] + \
    GRID[ix_(up,up)] + GRID[ix_(up,down)] + GRID[ix_(down,up)] + GRID[ix_(down,down)]
GRID = (neighbors == 3) | (GRID & (neighbors==2))
```

Summary

- Provides foundational N-dimensional array composed of homogeneous elements of a particular “dtype”
- The dtype of the elements is extensive (but not very extensible)
- Arrays can be sliced and diced with simple syntax to provide easy manipulation and selection.
- Provides fast and powerful math, statistics, and linear algebra functions that operate over arrays.
- Utilities for sorting, reading and writing data also provided.

SciPy

SciPy

“Distribution of Python Numerical Tools masquerading as a Library”

Name	Description
cluster	KMeans and Vector Quantization
fftpack	Discrete Fourier Transform
integrate	Numerical Integration
interpolate	Interpolation routines
io	Data Input and Output
linalg	Fast Linear algebra
misc	Utilities
ndimage	N-dimensional Image processing

Name	Description
odr	Orthogonal Distance Regression
optimize	Constrained and Unconstrained Optimization
signal	Signal Processing Tools
sparse	Sparse Matrices and Algebra
spatial	Spatial Data Structures and Algorithms
special	Special functions (e.g. Bessel)
stats	Statistical Functions and Distributions

scipy.stats --- Continuous Distributions

94 continuous distributions!

Methods on all Random Variable Objets.

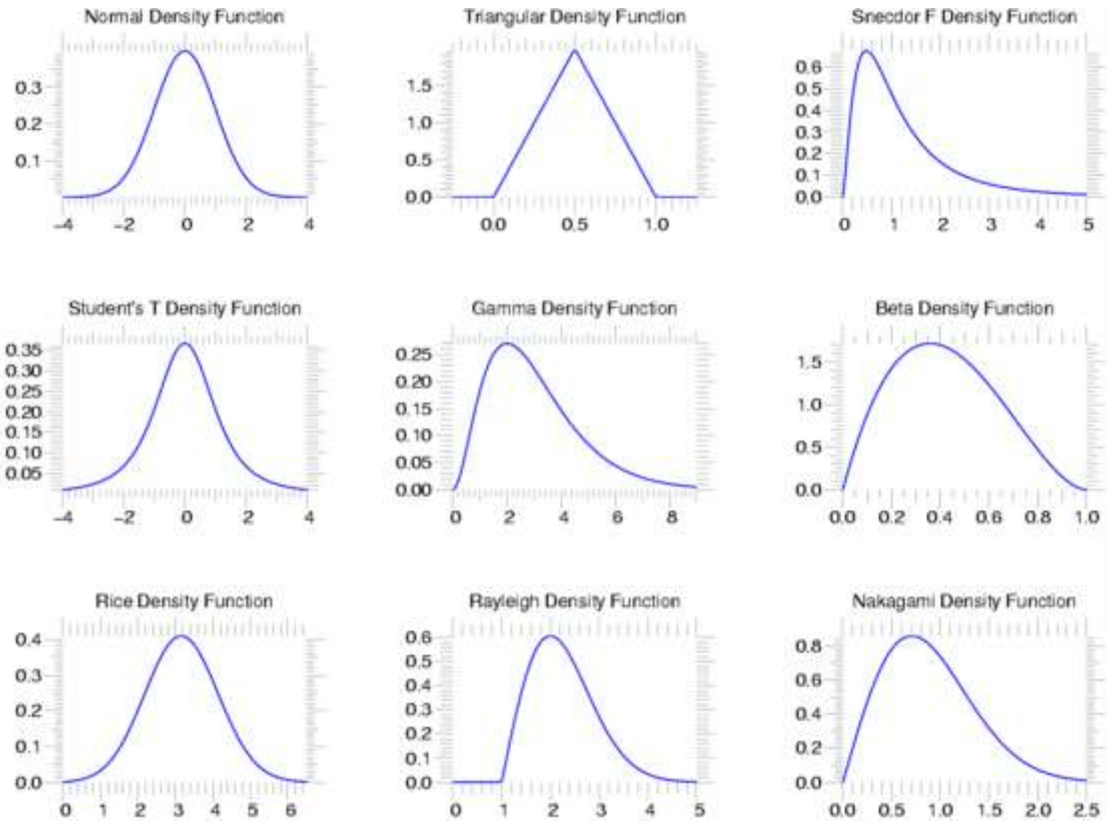
pdf

cdf

rvs

ppf

stats



scipy.stats --- Discrete Distributions

13 standard discrete distributions (plus any arbitrary finite RV)

Methods

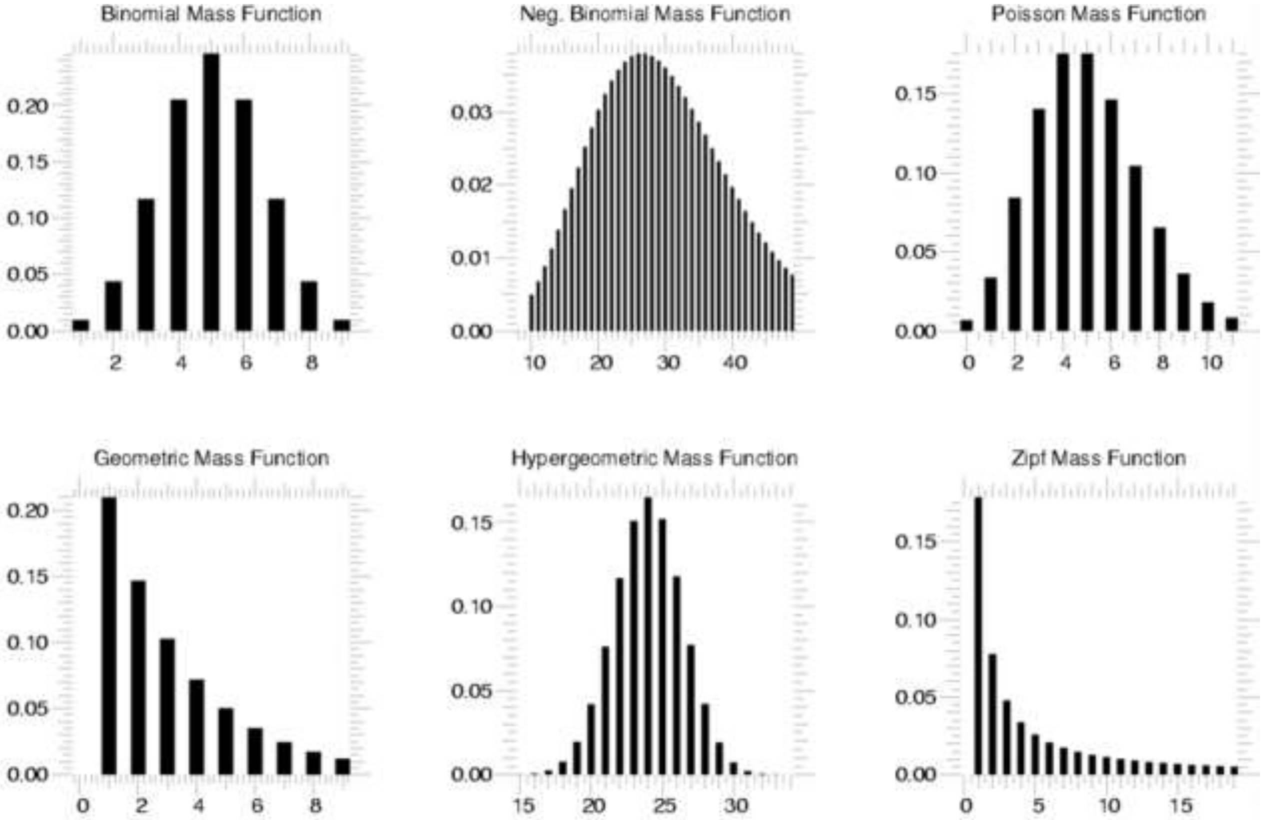
pdf

cdf

rvs

ppf

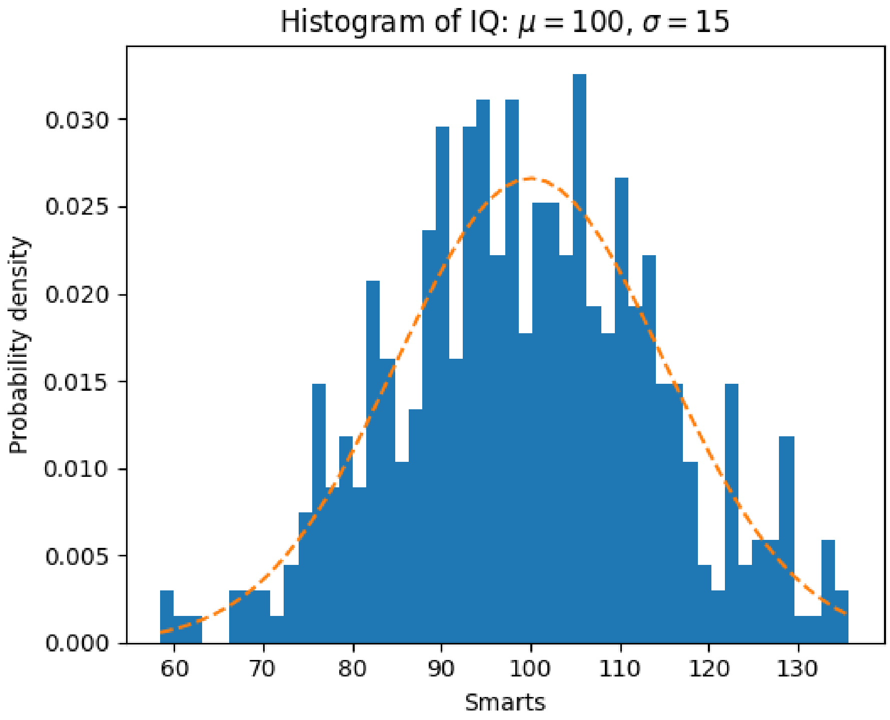
stats



Matplotlib

matplotlib a powerful plotting engine

```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

np.random.seed(0)

# example data
mu = 100  # mean of distribution
sigma = 15  # standard deviation of distribution
x = mu + sigma * np.random.randn(437)

num_bins = 50

fig, ax = plt.subplots()

# the histogram of the data
n, bins, patches = ax.hist(x, num_bins, normed=1)

# add a 'best fit' line
y = mlab.normpdf(bins, mu, sigma)
ax.plot(bins, y, '--')
ax.set_xlabel('Smarts')
ax.set_ylabel('Probability density')
ax.set_title(r'Histogram of IQ: $\mu=100$,
$\sigma=15$')

# Tweak spacing to prevent clipping of ylabel
fig.tight_layout()
plt.show()
```



a powerful plotting engine

```
import matplotlib.pyplot as plt
import scipy.misc as misc

im = misc.face()
ax = plt.imshow(im)

plt.title('Racoon Face of size
%d x %d' % im.shape[:2])

plt.savefig(‘face.png’)
```

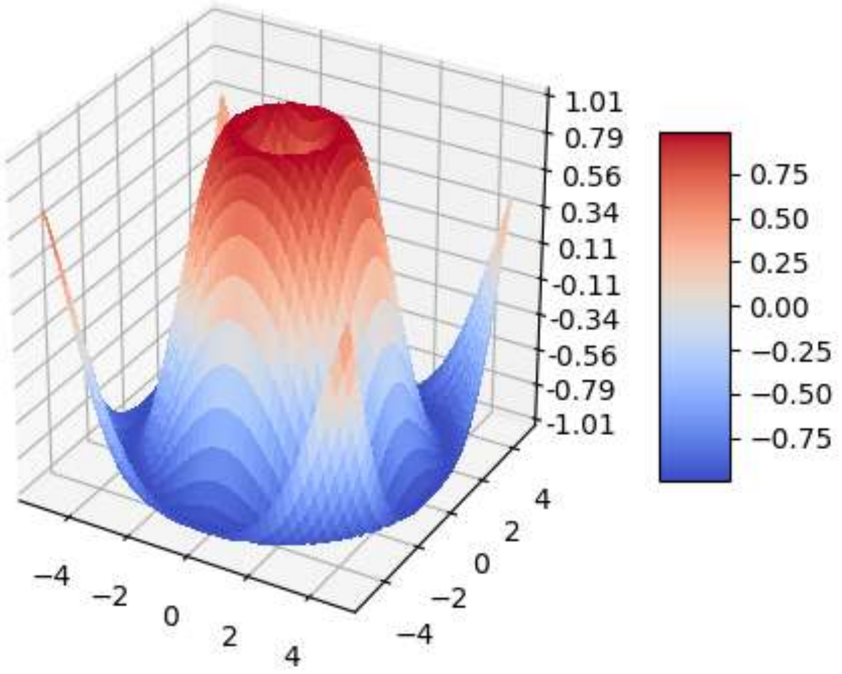
a powerful plotting engine

```
import matplotlib.pyplot as plt
import numpy as np

with plt.xkcd():
    fig = plt.figure()
    ax = fig.add_axes((0.1, 0.2, 0.8, 0.7))
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    plt.xticks([])
    plt.yticks([])
    ax.set_ylim([-30, 10])
    data = np.ones(100)
    data[70:] -= np.arange(30)

    plt.annotate('THE DAY I REALIZED\nI COULD
COOK BACON\nWHENEVER I WANTED',
       xy=(70, 1),
       arrowprops=dict(arrowstyle='->'),
       xytext=(15, -10))

    plt.plot(data)

    plt.xlabel('time')
    plt.ylabel('my overall health')
    fig.text(0.5, 0.05, '"Stove Ownership" from
xkcd by Randall Monroe',
        ha='center')
```

"STOVE OWNERSHIP" FROM XKCD BY RANDALL MONROE

matplotlib a powerful plotting engine

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator,
FormatStrFormatter
import numpy as np

fig = plt.figure()
ax = fig.gca(projection='3d')
# Make data.
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
# Plot the surface.
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
linewidth=0, antialiased=False)

# Customize the z axis.
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%
.02f'))

# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```



Pandas

Easy Data Wrangling

- Adds **indexes and labels** to 1-d and 2-d NumPy arrays (Series and DataFrame)
- Many convenience functions and methods to manipulate messy data-sets including **time-series**.
- Powerful indexing with automatic **data alignment**.
- Easy handling of **missing data**.
- Allows easy **joining and merging** Data Sets
- **Pivots** and reshaping (split-apply-combine)
- Powerful **group-by** operations with **summarization**
- Builtin visualization using labels and indexes

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Easy Data Wrangling

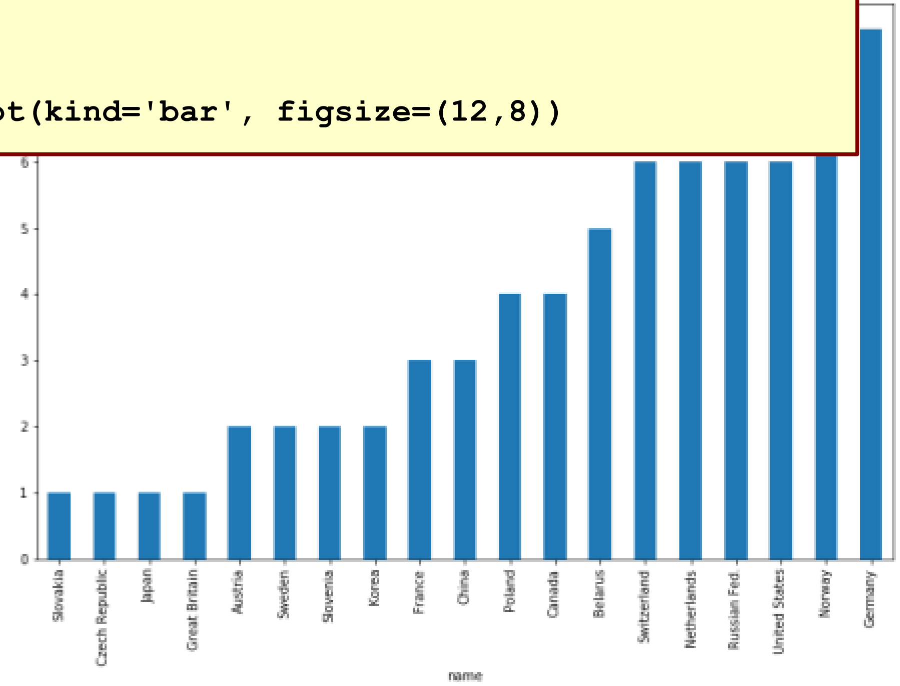
- Series Data Structure
  - built for **1-dimensional** series data
  - homogeneous data
  - Two arrays. One of data and another which is the index that can be a homogeneous array of any type like integers, objects, or date-times.
- DataFrame
  - built for **2-dimensional** collections of tabular data (think Excel sheet)
  - heterogeneous data comprised of multiple Series
  - includes an index column allowing sophisticated selection and alignment

pandas

Easy Data Wrangling

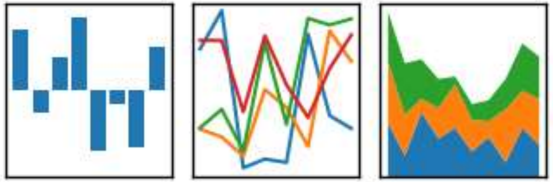
```python
medals = pd.read_csv('data/medals.csv', index_col='name')
medals.head()
gold = medals['medal'] == 'gold'
won = medals['count'] > 0
medals.loc[gold & won, 'count'].sort_values().plot(kind='bar', figsize=(12,8))
```

	count	medal	country
name			
Australia	1	bronze	AUS
Australia	2	silver	AUS
Australia	0	gold	AUS
Austria	1	bronze	AUT
Austria	6	silver	AUT



pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Easy Data Wrangling

```
google = pd.read_csv('data/goog.csv', index_col='Date', parse_dates=True)
google.info()
google.head()
google.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1257 entries, 2010-01-04 to 2014-12-31
Data columns (total 5 columns):
Open      1257 non-null float64
High      1257 non-null float64
Low       1257 non-null float64
Close     1257 non-null float64
Volume    194 non-null f
dtypes: float64(5)
memory usage: 58.9 KB
```

	Open	High	Low	Close	Volume
Date					
2010-01-04	313.16	314.44	311.81	313.06	NaN
2010-01-05	313.28	313.61	310.46	311.68	NaN
2010-01-06	312.62	312.62	302.88	303.83	NaN
2010-01-07	304.40	304.70	296.03	296.75	NaN
2010-01-08	295.70	301.32	294.26	300.71	NaN

	Open	High	Low	Close	Volume
count	1257.000000	1257.000000	1257.000000	1257.000000	1.940000e+02
mean	375.275593	378.450247	372.132474	375.327064	1.937264e+06
std	115.684354	116.288827	114.935742	115.664301	9.842775e+05
min	218.940000	220.920000	216.600000	217.820000	7.040350e+05
25%	285.790000	288.760000	283.060000	285.450000	1.338451e+06
50%	318.330000	320.800000	315.180000	317.260000	1.684634e+06
75%	452.540000	456.020000	449.740000	452.830000	2.164369e+06
max	612.790000	613.830000	608.690000	609.470000	6.795393e+06

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Easy Data Wrangling

```
df = pd.read_excel("data/pbpython/salesfunnel.xlsx")
df.head()
table = pd.pivot_table(df,
                       index=["Manager","Rep","Product"],
                       values=["Price","Quantity"],
                       aggfunc=[np.sum,np.mean])
```

	Account	Name	Rep	Manager	Product	Quantity	Price	Status
0	714466	Trantow-Barrows	Craig Booker	Debra Henley	CPU	1	30000	presented
1	714466	Trantow-Barrows	Craig Booker	Debra Henley	Software	1	10000	presented
2	714466	Trantow-Barrows	Craig Booker	Debra Henley	Maintenance	2	5000	pending
3	737550	Fritsch, Russel and Anderson	Craig Booker	Debra Henley	CPU	1	35000	declined
4	146832	Kiehn-Spinka	Daniel Hilton	Debra Henley	CPU	2	65000	won

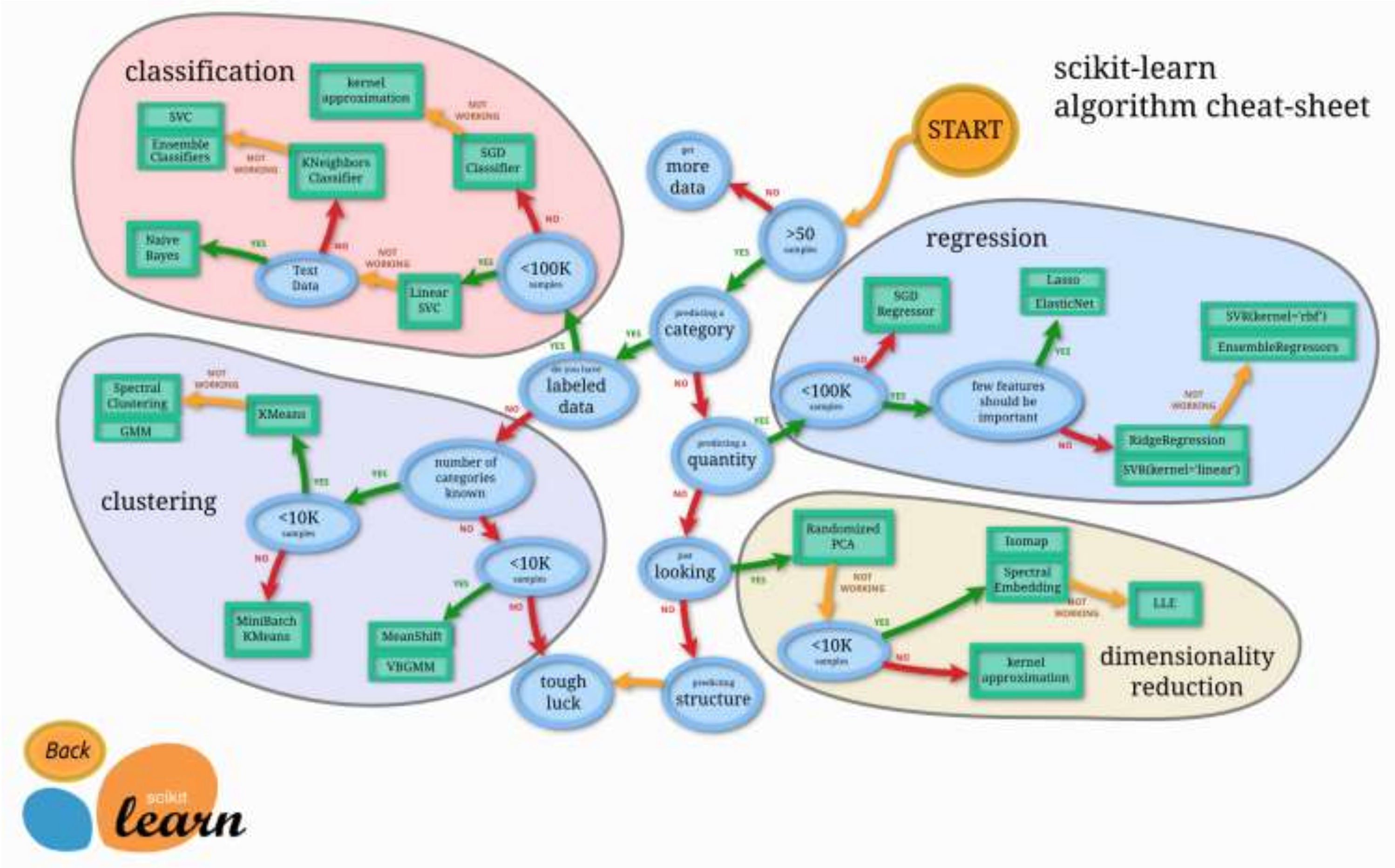
			sum		mean	
			Price	Quantity	Price	Quantity
Manager	Rep	Product				
Debra Henley	Craig Booker	CPU	65000	2	32500.0	1.0
		Maintenance	5000	2	5000.0	2.0
		Software	10000	1	10000.0	1.0
	Daniel Hilton	CPU	105000	4	52500.0	2.0
		Software	10000	1	10000.0	1.0
	John Smith	CPU	35000	1	35000.0	1.0
		Maintenance	5000	2	5000.0	2.0
Fred Anderson	Cedric Moss	CPU	95000	3	47500.0	1.5
		Maintenance	5000	1	5000.0	1.0
		Software	10000	1	10000.0	1.0
	Wendy Yule	CPU	165000	7	82500.0	3.5
		Maintenance	7000	3	7000.0	3.0
		Monitor	5000	2	5000.0	2.0

Scikit-Learn

Machine Learning made easy

- **Supervised Learning** — uses “labeled” data to train a model
  - Regression — predicted variable is continuous
  - Classification — predicted variable is discrete
- **Unsupervised Learning**
  - Clustering — discover categories in the data
  - Density Estimation — determine representation of data
  - Dimensionality Reduction — represent data with fewer variables or feature vectors

- Reinforcement Learning — “goal-oriented” learning (e.g. drive a car)
- **Deep Learning** — neural networks with many layers
- Semi-supervised Learning (use some labeled data for training)


scikit-learn
algorithm cheat-sheet
START
classification
regression
clustering
dimensionality
reduction
get
more
data
>50
samples
predicting a
category
do you have
labeled
data
predicting a
quantity
just
looking
predicting
structure
tough
luck
<100K
samples
Text
Data
Linear
SVC
Naive
Bayes
KNeighbors
Classifier
SVC
Ensemble
Classifiers
SGD
Classifier
kernel
approximation
number of
categories
known
<10K
samples
KMeans
Spectral
Clustering
GMM
MiniBatch
KMeans
MeanShift
VBGMM
SGD
Regressor
Lasso
ElasticNet
few features
should be
important
RidgeRegression
SVR(kernel='linear')
SVR(kernel='rbf')
EnsembleRegressors
Randomized
PCA
Isomap
Spectral
Embedding
LLE
kernel
approximation
YES
NO
NOT
WORKING
Back
scikit
learn

Basic scikit-learn experience

1) Create or Load Data

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> digits = datasets.load_digits()
```

A Scikit-learn dataset is a “dictionary-like” object with input data stored as the .data attribute and labels stored as the .target attribute

.data attribute is always a 2D array **`(n_samples, n_features)`**

May need to extract features from raw data to produce data scikit-learn can use

Basic scikit-learn experience

2) Choose Model (or Build Pipeline of Models)

```
>>> from sklearn import svm
>>> clf = svm.SVC(gamma=0.001, C=100.)
```

Most models are model-families and have “hyper-parameters” that specify the specific model function.   Good values for these can be found via grid-search and cross-validation (easy target for parallelization).

Here “gamma” and “C” are hyper-parameters.

Many choices of models: http://scikit-learn.org/stable/supervised_learning.html

Basic scikit-learn experience

3) Train the Model

```
>>> clf.fit(data[:-1], labels[:-1])
```

Models have a “fit” method which updates the parameters-to-be-estimated in the model in-place so that after fitting the model is “trained”

For validation and scoring you need to leave out some of the data to use later. cross-validation (e.g. k-fold) techniques can also be parallelized easily.

Here we “leave-one-out” (or n-fold)

Basic scikit-learn experience

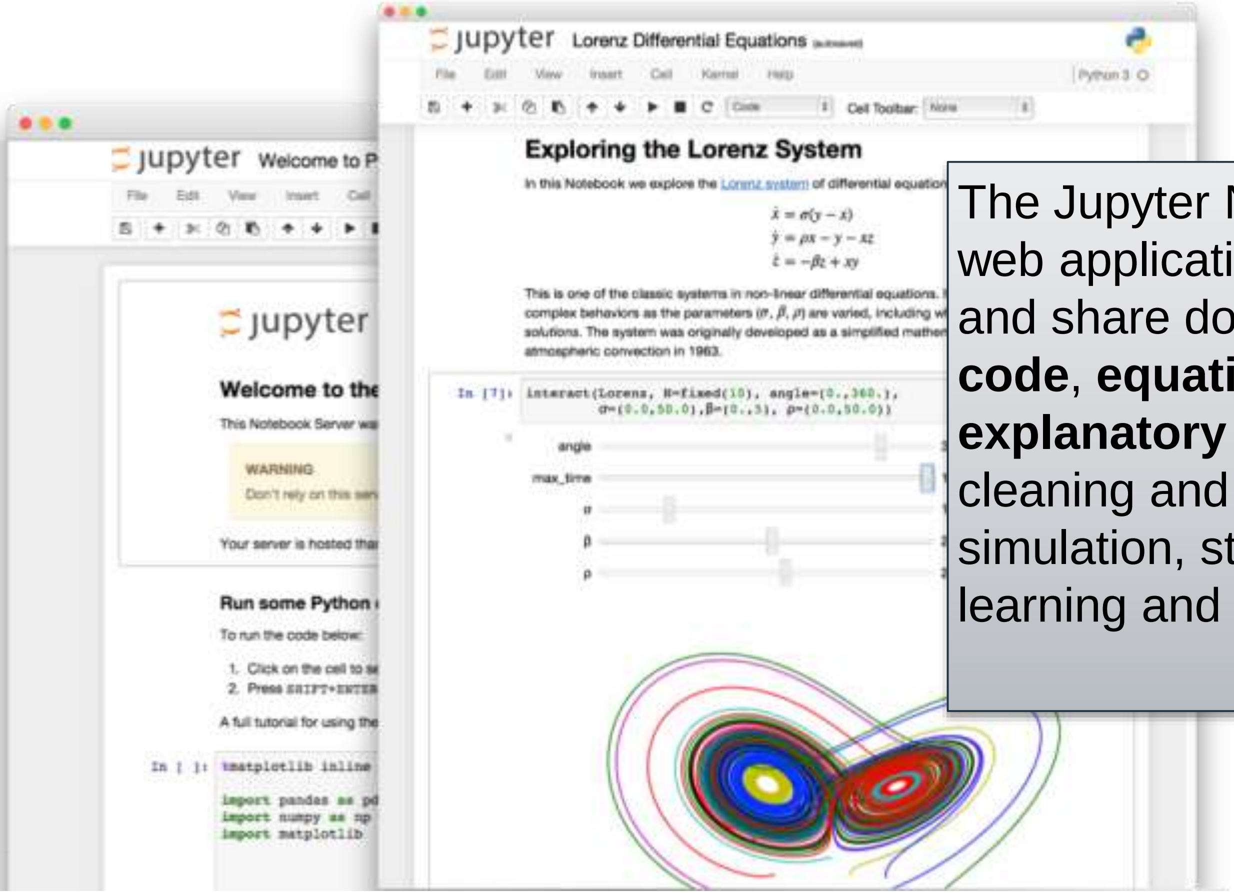
4) Predict new values

```
>>> clf.predict(data[-1:])
```

Prediction of new data uses the trained parameters of the model. Cross-validation can be used to understand how sensitive the model is to different partitions of the data.

```
>>> from sklearn.model_selection import cross_val_score
>>> scores = cross_val_score(clf, data, target, cv=10)
array([ 0.96…,  1.   …,  0.96… , 1.         ])
```

Jupyter



The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live **code**, **equations**, **visualizations** and **explanatory text**. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, machine learning and much more.

jupyter Flight Delays Modling (autosaved) R Logout

File Edit View Insert Cell Kernel Help R

Markdown CellToolbar

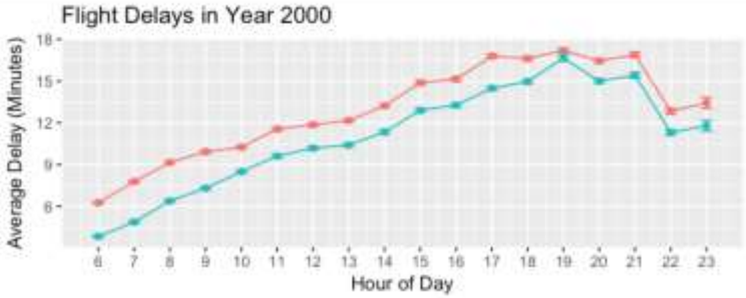
```
    se=sqrt(var(newdelay,na.rm=TRUE)/length(na.omit(newdelay
    obs=length(na.omit(newdelay)))
```

In [7]:
```
head(plot_data, n=c(3L))
```

departure_hour	delay_type	mu	se	obs
6	Arrival Delay	6.258174	0.09168138	56981
6	Departure Delay	3.840884	0.08537210	57210
7	Arrival Delay	7.777764	0.09093139	62987

Using the R kernel allows us to make visualizations using ggplot natively.

In [10]:
```
p=ggplot(plot_data,aes(x=departure_hour,y=mu,min=mu-se,max=mu+se,group=dela
  geom_line() +
  geom_point() +
  geom_errorbar(width=.33) +
  scale_x_continuous(breaks=seq(6,23)) +
  labs(x="Hour of Day",y="Average Delay (Minutes)",title="Flight Delays in
  theme(legend.position="bottom") +
  scale_color_discrete(name="Delay Type")

p
```



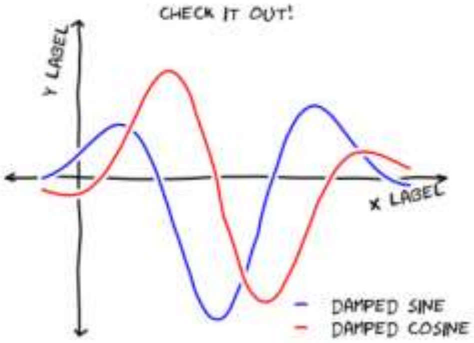
jupyter XKCD_plots (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Python 3

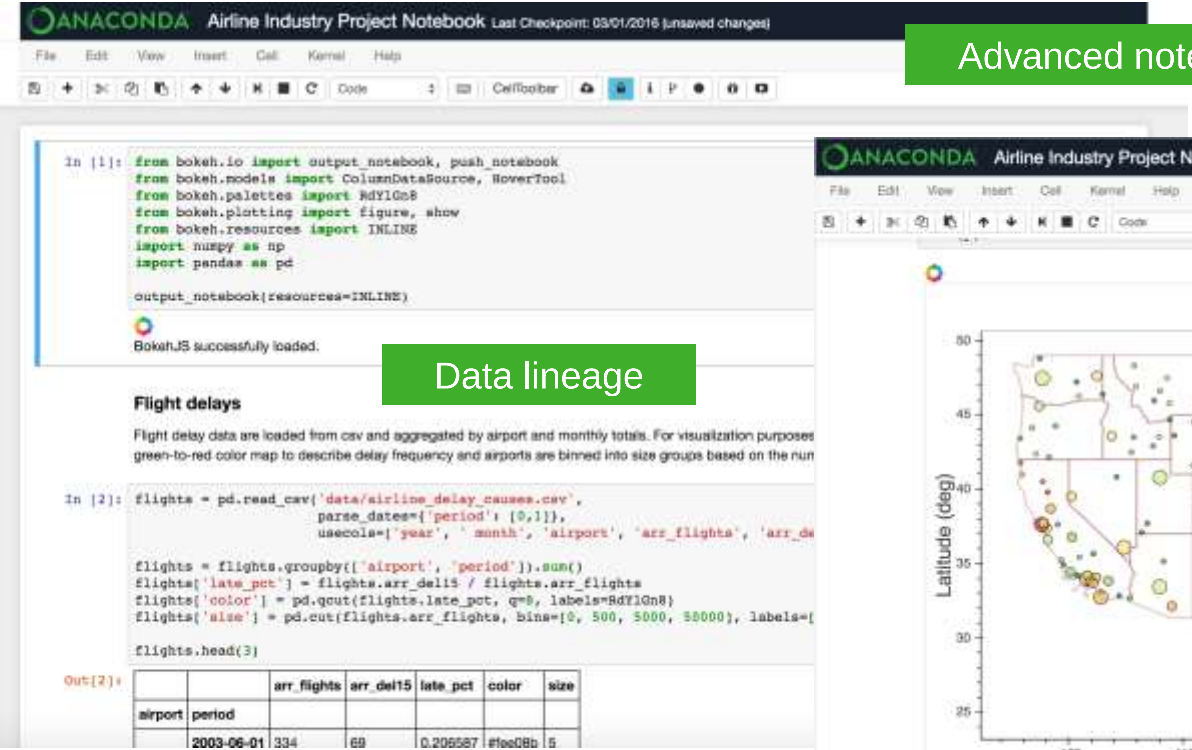
Markdown CellToolbar

In [5]:
```
np.random.seed(0)

ax = pylab.axes()

x = np.linspace(0, 10, 100)
ax.plot(x, np.sin(x) * np.exp(-0.1 * (x - 5) ** 2), 'b', lw=1, label='dampe
ax.plot(x, -np.cos(x) * np.exp(-0.1 * (x - 5) ** 2), 'r', lw=1, label='damp

ax.set_title('check it out!')
ax.set_xlabel('x label')
ax.set_ylabel('y label')

ax.legend(loc='lower right')

ax.set_xlim(0, 10)
ax.set_ylim(-1.0, 1.0)

#XKCDify the axes -- this operates in-place
XKCDify(ax, xaxis_loc=0.0, yaxis_loc=1.0,
        xaxis_arrow='+-', yaxis_arrow='+-',
        expand_axes=True)
```

Out[5]: <matplotlib.axes.AxesSubplot at 0x2fecbd0>

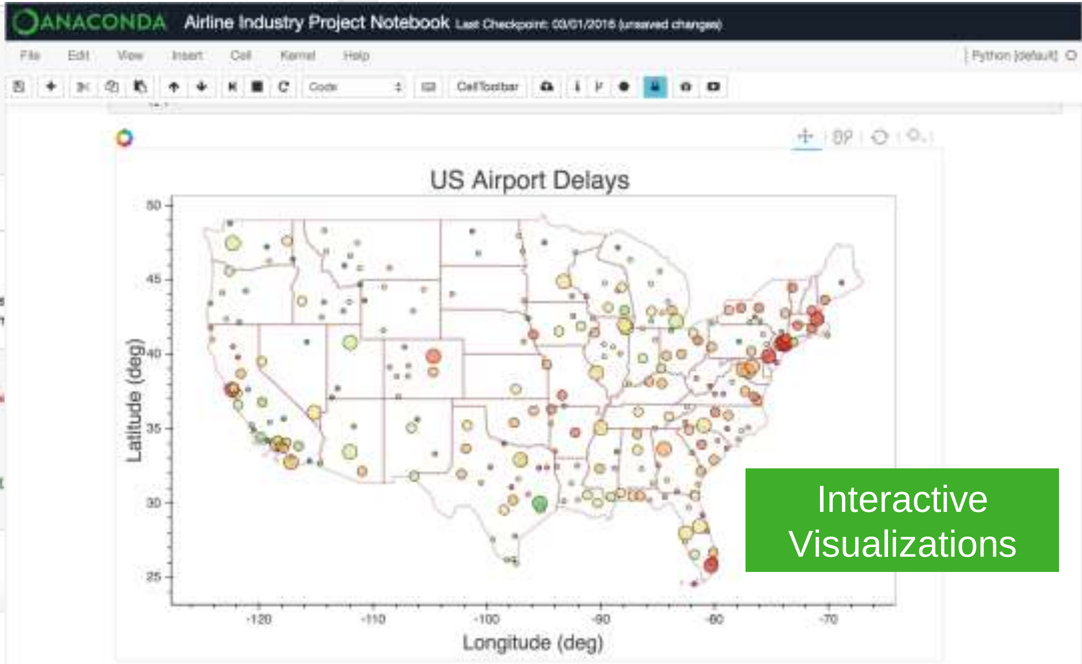


Collaborative Executable Notebooks



Advanced notebook extensions

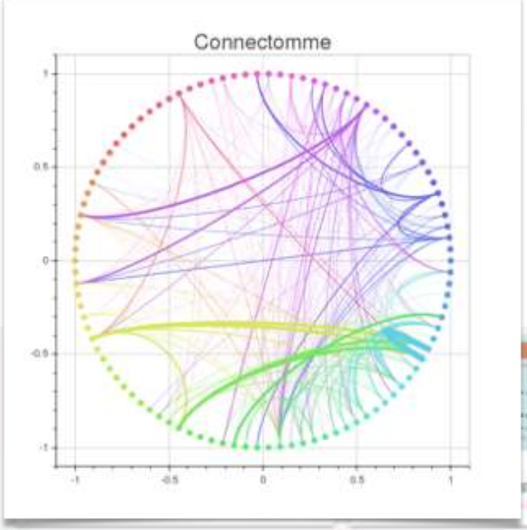
Data lineage

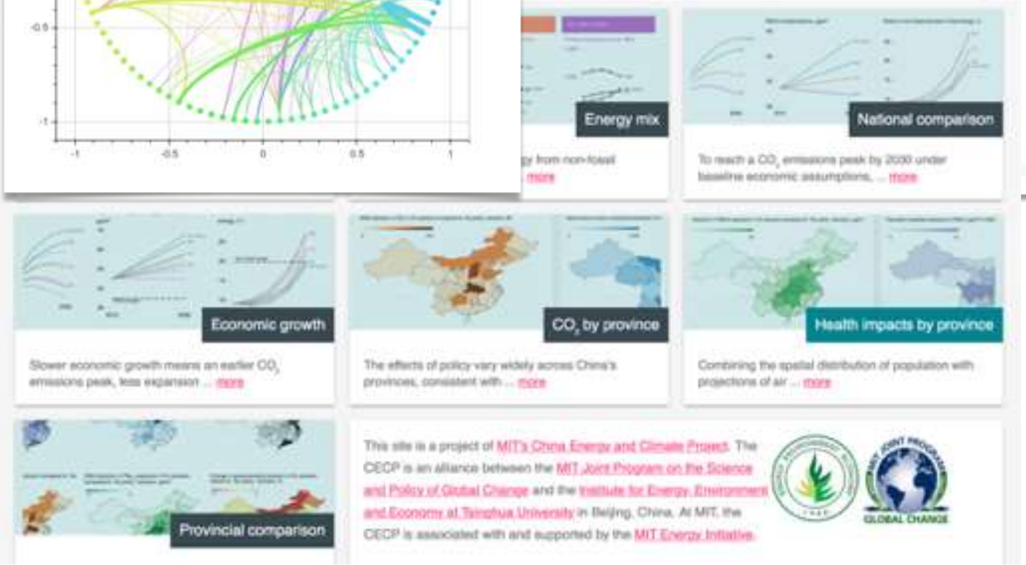


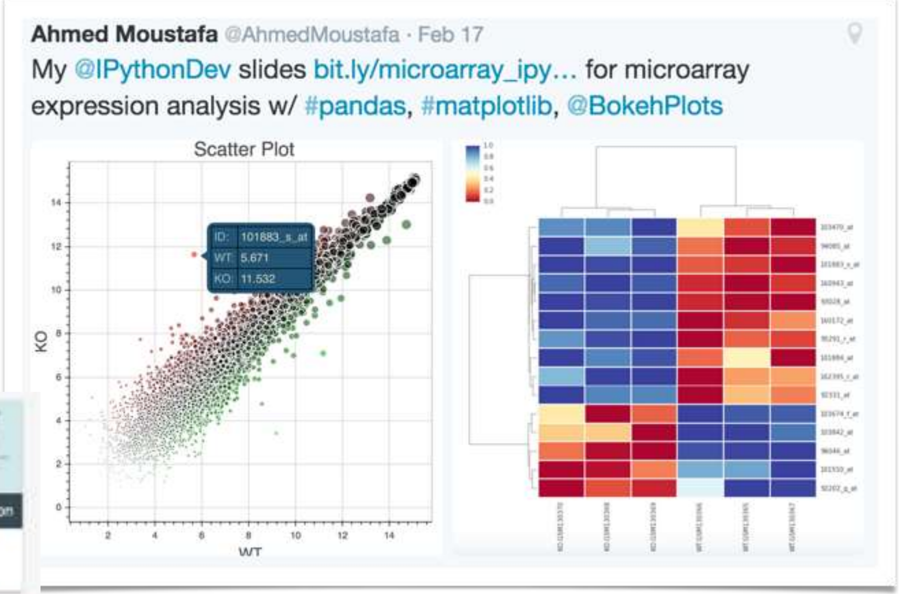
Interactive Visualizations

Interactive Data Vizualization Apps with Bokeh

Interactive Data Visualization

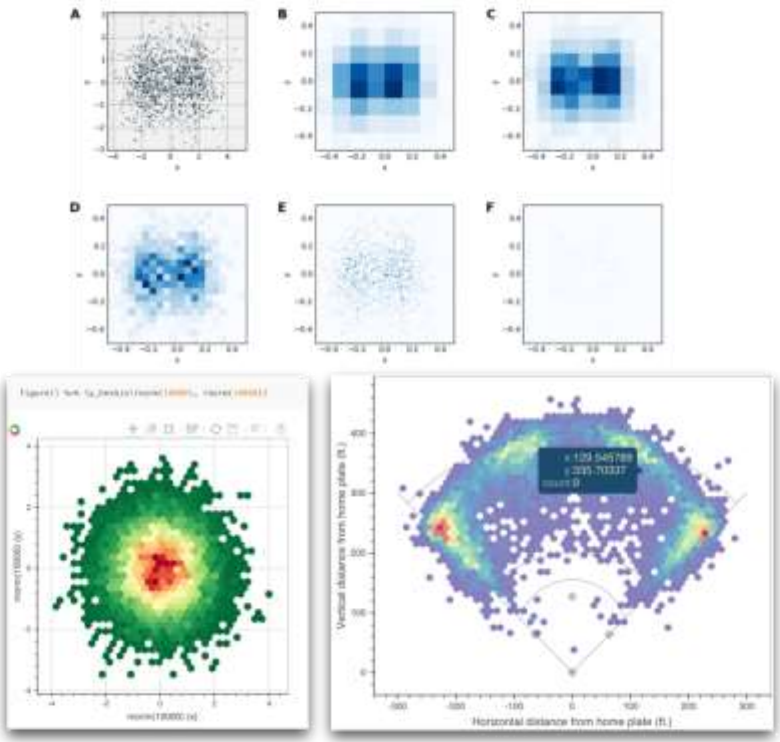






- Interactive viz, widgets, and tools
- Versatile high level graphics
- Streaming, dynamic, large data
- Optimized for the browser
- No Javascript
- With or without a server

Rapid Prototyping Visual Apps

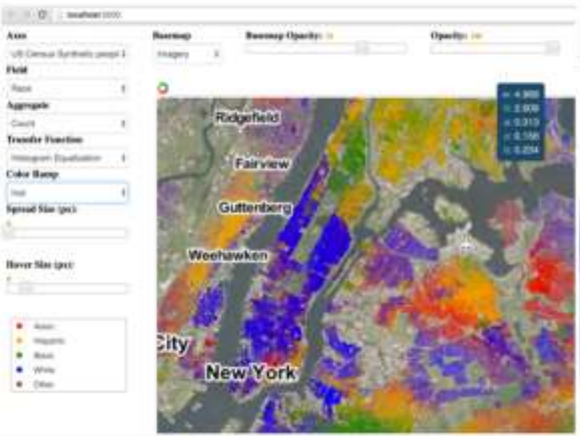


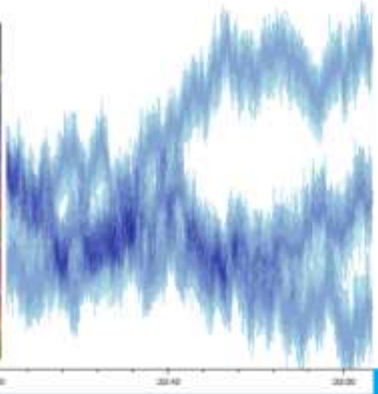
- Python interface
- R interface
- Smart plotting

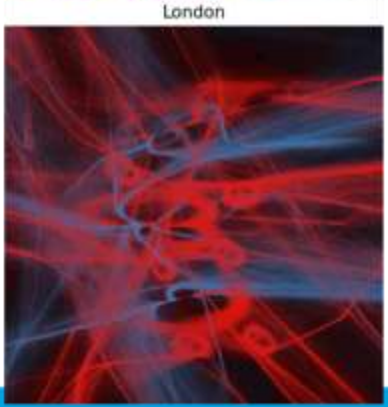
Plotting Billions of Points and Map Integration with Datashader

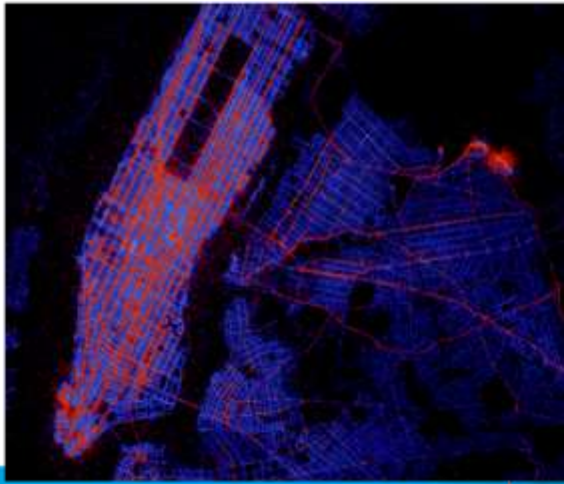
Datashader: Rendering a Billion Points of Data

- `datashader` provides a fast, configurable visualization pipeline for faithfully revealing even very large datasets
- Each of these visualizations requires just a few lines of code and no magic numbers to adjust by trial and error.



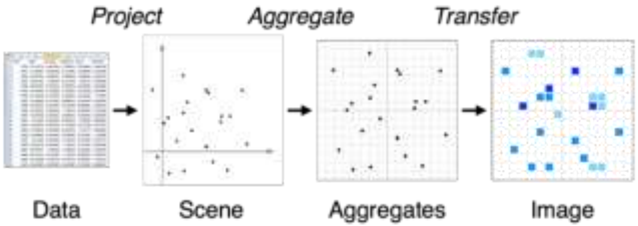






Datashader

Configurable visualization pipeline:



Simple code, with no magic numbers:

```
cvs = ds.Canvas(plot_width=900, plot_height=600)
agg = cvs.points(df,'longitude','latitude',ds.count())
img = tf.shade(agg, cmap=hot, how='eq_hist')
```

Powerful computations on the data:

E.g. show only top 10% highest-count pixels,
or pixels where blacks outnumber whites:

```
agg2 = agg.where(agg>=np.percentile(agg,90))
agg3 = agg.where(agg.sel(race='w') < agg.sel(race='b'))
```

High performance

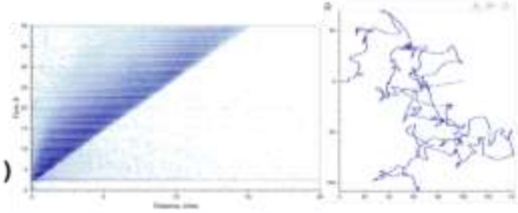
Only the first stage depends on the dataset size — visualizes arbitrarily large datasets, without downsampling, in your browser.

Fast on ordinary machines, and scales up easily for distributed, out of core processing

E.g. 10 million points in 0.5 seconds on a laptop, or 3 billion points in 10 seconds on a cluster.

Flexible datatypes

- Scatterplots/heatmaps
- Time series
- Connected points (trajectories)
- Rasters (via Rasterio)



Highly interactive:

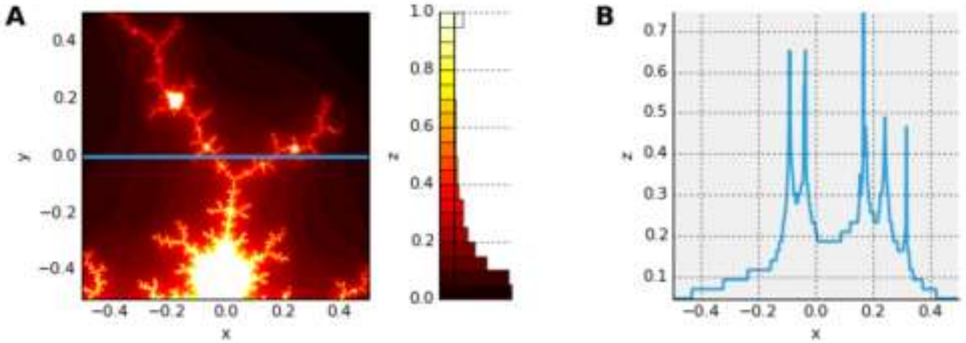
Output can be embedded into Bokeh plots with pan,zoom,overlays,axes

Data Visualization and Applications made easy with Holoviews

HoloViews: Stop plotting your data

HoloViews makes it simple to create beautiful interactive Bokeh or Matplotlib visualizations of complex data.

```
import numpy as np
import holoviews as hv
hv.notebook_extension('matplotlib')
fractal = hv.Image(np.load('mandelbrot.npy'))

((fractal * hv.HLine(y=0)).hist() + fractal.sample(y=0))
```

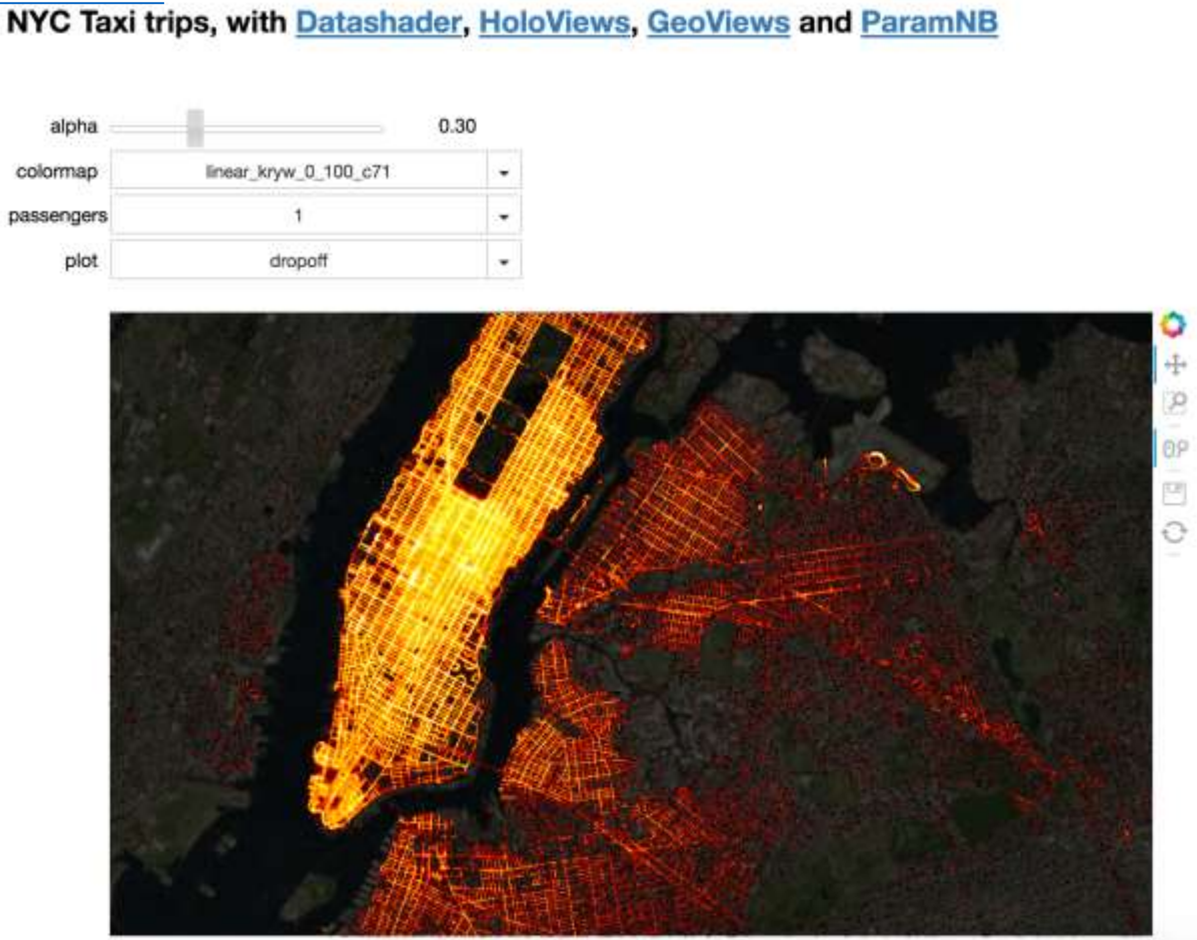


- Exploring data can be tedious if you use a plotting library directly, because you will need to specify details about your data (units, dimensions, names, etc.) every time you construct a new type of plot.
- With HoloViews, you instead annotate your data once, and then flexible plotting comes for free — HoloViews objects just display themselves, alone or in any combination.
- It’s now easy to lay out subfigures, overlay traces, facet or animate a multidimensional dataset, sample or aggregate to reduce dimensionality, preserving the metadata each time so the results visualize themselves.

https://anaconda.org/jbednar/nyc_taxi-paramnb/notebook

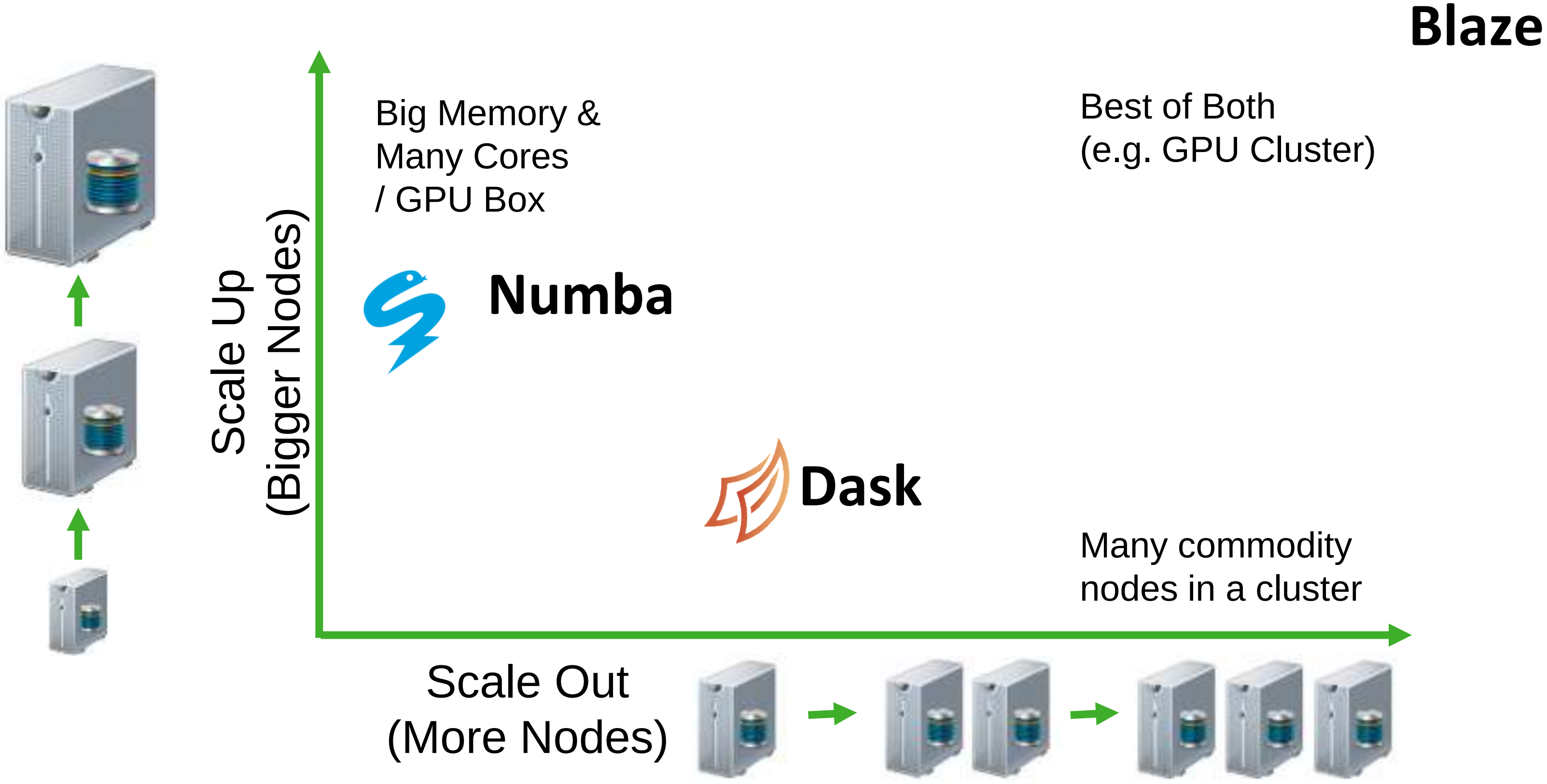
Data Widgets and Applications from Jupyter Notebooks!

```
tiles = gv.WMTS(WMTSTileSource(url='https://server.arcgisonline.com/Ar
                                   'World_Imagery/MapServer/tile/{Z}/{
tile_options = dict(width=800,height=475,xaxis=None,yaxis=None,bgcolor

passenger_counts = sorted(df.passenger_count.unique().tolist())

class Options(hv.streams.Stream):
    alpha      = param.Magnitude(default=0.75, doc="Alpha value for t
    colormap   = param.ObjectSelector(default=cm["fire"], objects=cm.
    plot       = param.ObjectSelector(default="pickup",   objects=["p
    passengers = param.ObjectSelector(default=1,          objects=pas

    def make_plot(self, x_range=None, y_range=None, **kwargs):
        map_tiles = tiles(style=dict(alpha=self.alpha), plot=tile_opti

        df_filt = df[df.passenger_count==self.passengers]
        points = hv.Points(gv.Dataset(df_filt, kdims=[self.plot+'_x', 
        taxi_trips = datashade(points, width=800, height=475, x_sampli
                               cmap=self.colormap, element_type=gv.Ima
                               dynamic=False, x_range=x_range, y_range

        return map_tiles * taxi_trips

selector = Options(name="")
paramnb.Widgets(selector, callback=selector.update)
hv.DynamicMap(selector.make_plot, kdims=[], streams=[selector, RangeXY
```



Scaling Up and Out with Numba and Dask

Scale Up vs Scale Out



Scaling Up! Optimized Python with JIT compilation from Numba

Numba

- Started in 2012
- Release 31 (0.31) in February
- Version 1.0 coming in 2017
- Particularly suited to Numeric computing
- Lots of features!
- Ahead of Time Compilation
- Wide community adoption and use

```
conda install numba
```

Example: Filter an array

Numba decorator (nopython=True not required)

Array Allocation

Looping over ndarray x as an iterator

Using numpy math functions

Returning a slice of the array

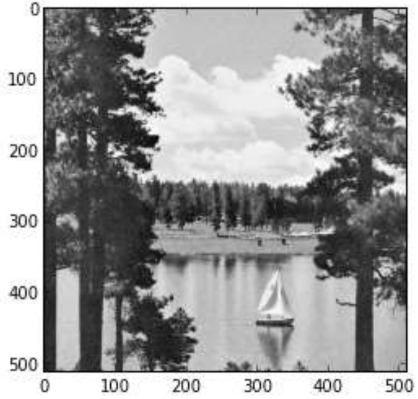
```
In [87]: @jit(nopython=True)
         def nan_compact(x):
             out = np.empty_like(x)
             out_index = 0
             for element in x:
                 if not np.isnan(element):
                     out[out_index] = element
                     out_index += 1
             return out[:out_index]
```

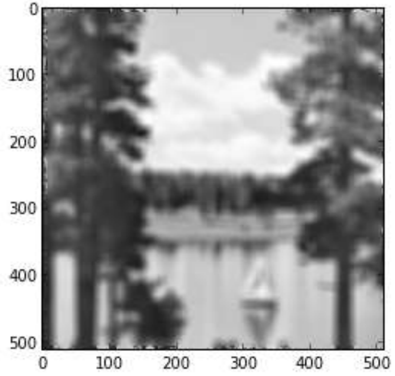
```
In [88]: a = np.random.uniform(size=10000)
         a[a < 0.2] = np.nan
         np.testing.assert_equal(nan_compact(a), a[~np.isnan(a)])
```

```
In [89]: %timeit a[~np.isnan(a)]
         %timeit nan_compact(a)

10000 loops, best of 3: 52 µs per loop
100000 loops, best of 3: 19.6 µs per loop
```

2.7x Speedup over NumPy!

Image Processing





~1500x speed-up

```
@jit('void(f8[:,:],f8[:,:],f8[:,:])')
def filter(image, filt, output):
    M, N = image.shape
    m, n = filt.shape
    for i in range(m//2, M-m//2):
        for j in range(n//2, N-n//2):
            result = 0.0
            for k in range(m):
                for l in range(n):
                    result += image[i+k-m//2,j+l-n//2]*filt[k, l]
            output[i,j] = result
```

Numba Compatibility

Does *not* replace the standard Python interpreter

OS	HW	SW
Windows (7 and later)	32 and 64-bit x86 CPUs	Python 2 and 3
OS X (10.9 and later)	CUDA-capable NVIDIA GPUs	NumPy 1.7 through 1.11
Linux (~RHEL 5 and later)	HSA-capable AMD GPUs	

Scaling Out with Dask (integrates with but doesn't depend on Hadoop)

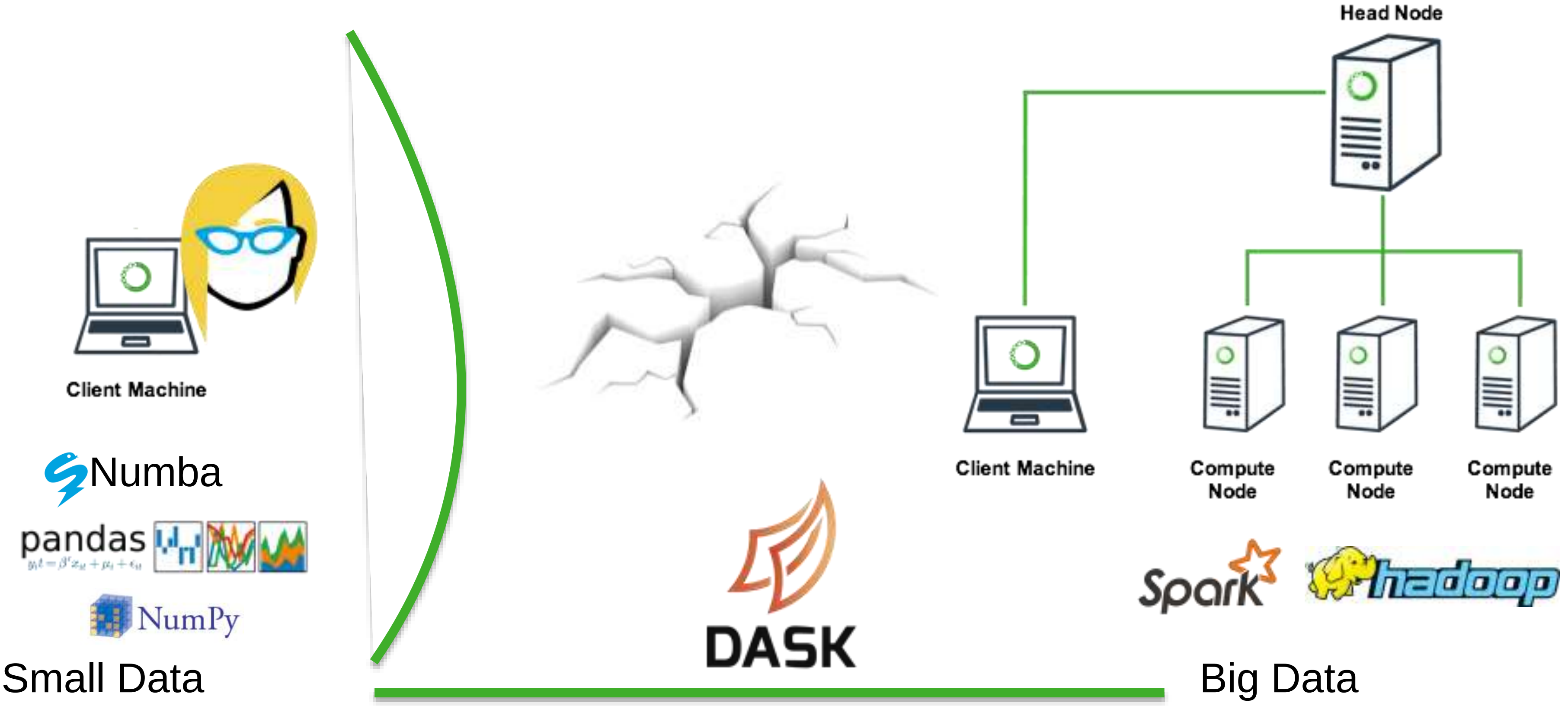
Dask

- Started as part of Blaze in early 2014.
- General parallel programming engine
- Flexible and therefore highly suited for
  - Commodity Clusters
  - Advanced Algorithms
- Wide community adoption and use

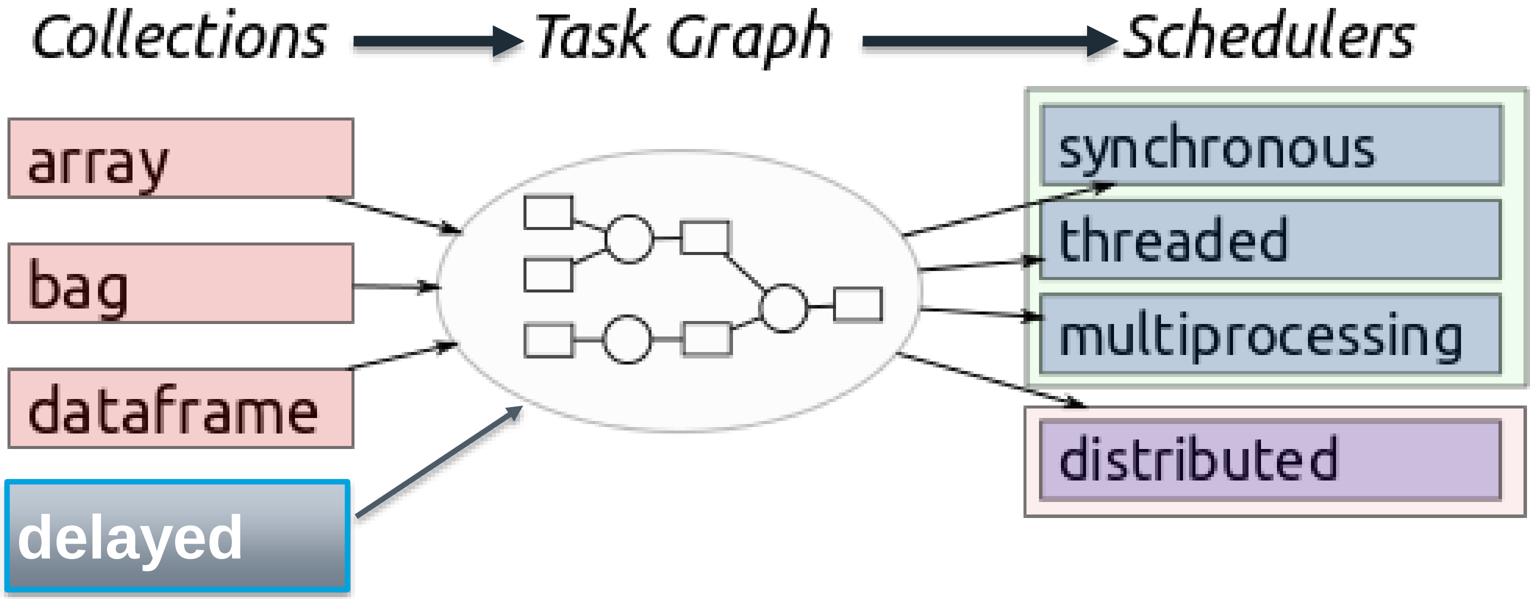
```
conda install -c conda-forge dask
```

```
pip install dask[complete] distributed --upgrade
```

Moving from small data to big data

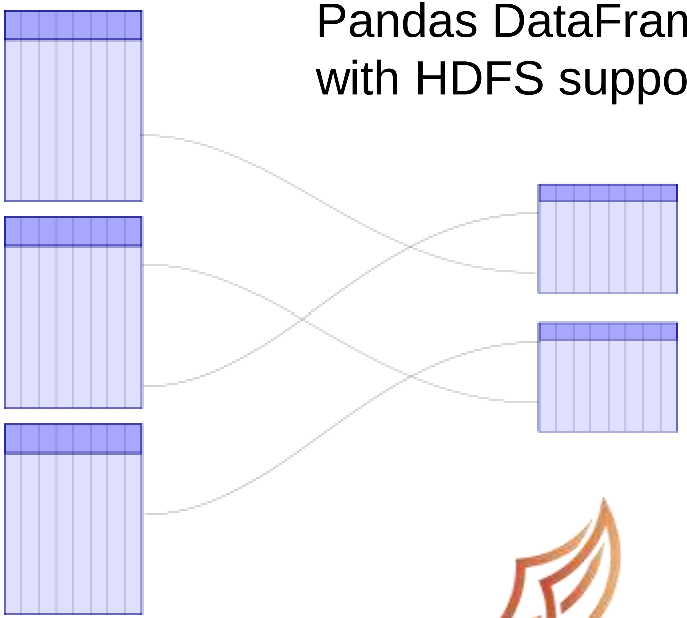


Dask: From User Interaction to Execution



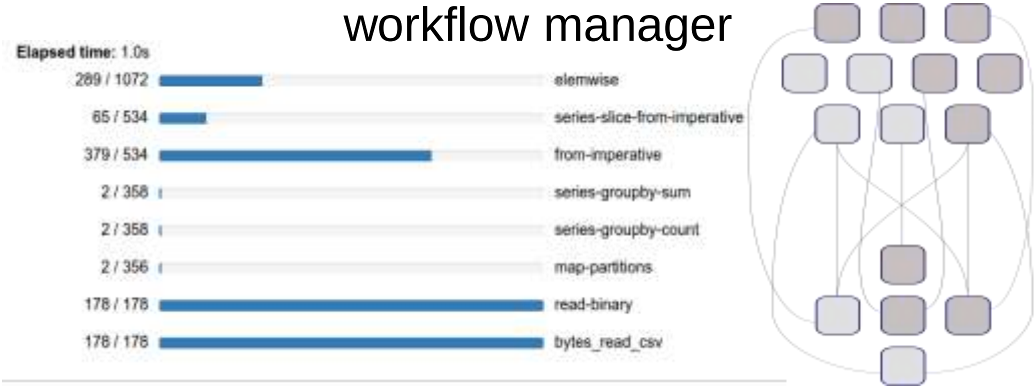
Dask: Parallel Data Processing

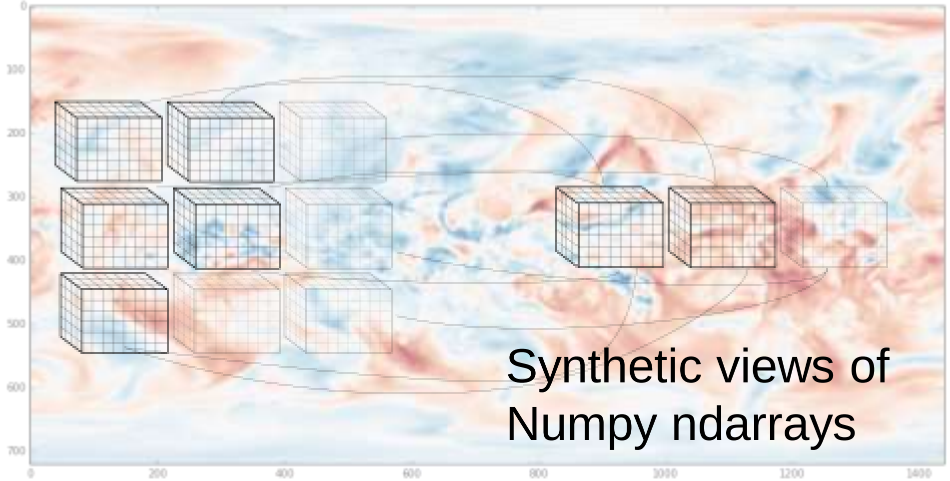
Synthetic views of Pandas DataFrames with HDFS support



DASK

DAG construction and workflow manager



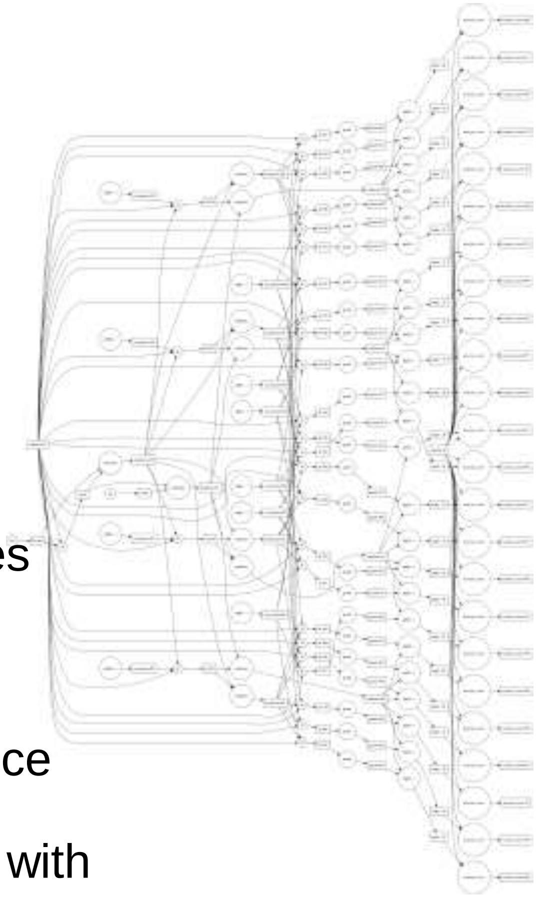


Synthetic views of Numpy ndarrays

Overview of Dask

Dask is a Python parallel computing library that is:

- **Familiar**: Implements parallel NumPy and Pandas objects
- **Fast**: Optimized for demanding for numerical applications
- **Flexible**: for sophisticated and messy algorithms
- **Scales up**: Runs resiliently on clusters of 100s of machines
- **Scales down**: Pragmatic in a single process on a laptop
- **Interactive**: Responsive and fast for interactive data science



Dask **complements** the rest of Anaconda. It was developed with NumPy, Pandas, and scikit-learn developers.

Dask Collections: Familiar Expressions and API

Dask array (mimics NumPy)

```
x.T - x.mean(axis=0)
```

Dask bag (collection of data)

```
b.map(json.loads).foldby(...)
```

Dask dataframe (mimics Pandas)

```
df.groupby(df.index).value.mean()
```

Dask delayed (wraps custom code)

```
def load(filename):
def clean(data):
def analyze(result):
```

Dask Dataframes

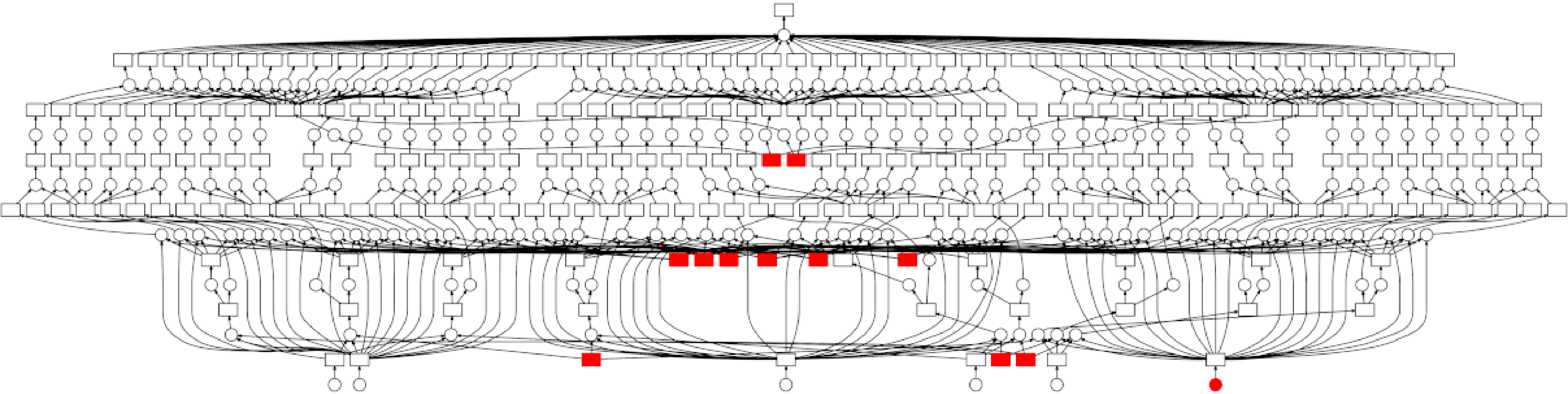
pandas

DASK

```
>>> import pandas as pd

>>> df = pd.read_csv('iris.csv')

>>> df.head()

   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa

>>> max_sepal_length_setosa = df[df.species
== 'setosa'].sepal_length.max()

5.7999999999999998
```

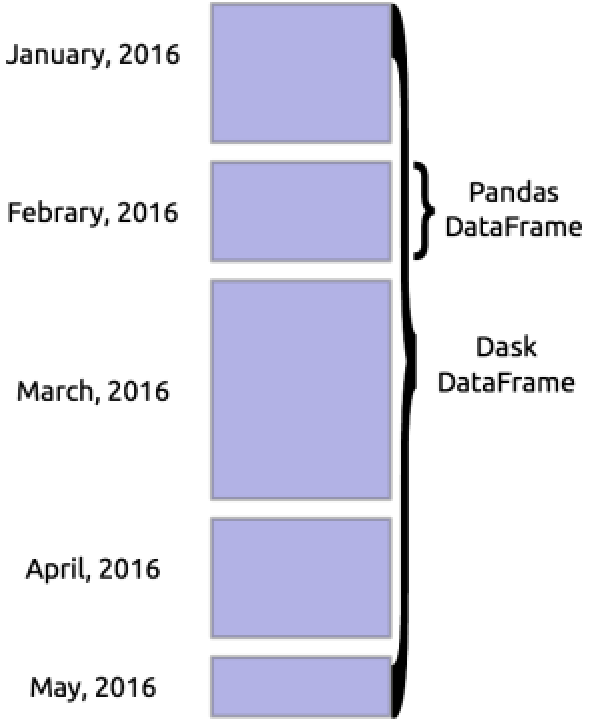
```
>>> import dask.dataframe as dd

>>> ddf = dd.read_csv('*.csv')

>>> ddf.head()

   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
…

>>> d_max_sepal_length_setosa = ddf[ddf.species
== 'setosa'].sepal_length.max()

>>> d_max_sepal_length_setosa.compute()

5.7999999999999998
```

Dask Graphs: Example Machine Learning Pipeline



Example 1: Using Dask DataFrames on a cluster with CSV data

- Built from Pandas DataFrames
- Match Pandas interface
- Access data from HDFS, S3, local, etc.
- Fast, low latency
- Responsive user interface



Dask Arrays

NumPy

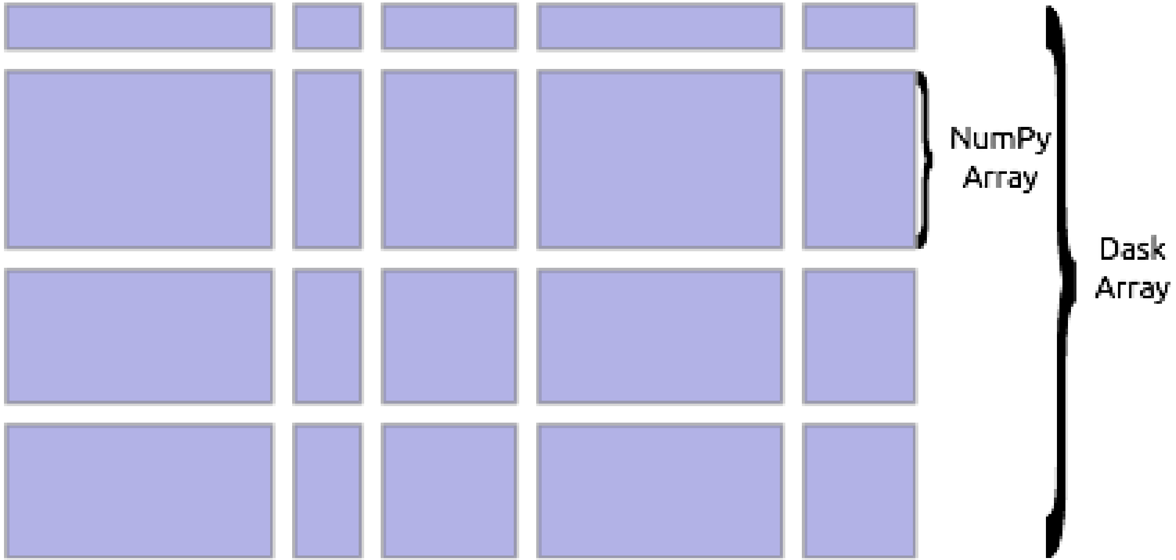
```
>>> import numpy as np
>>> np_ones = np.ones((5000, 1000))
>>> np_ones
array([[ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       ...,
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.]])

>>> np_y = np.log(np_ones + 1)[:5].sum(axis=1)
>>> np_y
array([ 693.14718056,  693.14718056,
693.14718056,  693.14718056, 693.14718056])
```

DASK

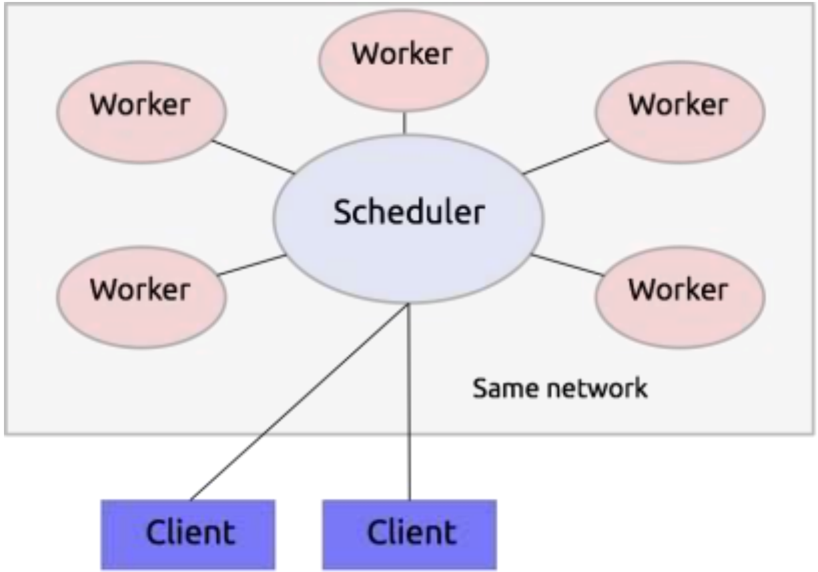
```
>>> import dask.array as da
>>> da_ones = da.ones((5000000, 1000000),
                      chunks=(1000, 1000))
>>> da_ones.compute()
array([[ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       ...,
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.]])

>>> da_y = da.log(da_ones + 1)[:5].sum(axis=1)
>>> np_da_y = np.array(da_y) #fits in memory
array([ 693.14718056,  693.14718056,
693.14718056,  693.14718056, …, 693.14718056])

# If result doesn't fit in memory
>>> da_y.to_hdf5('myfile.hdf5', 'result')
```

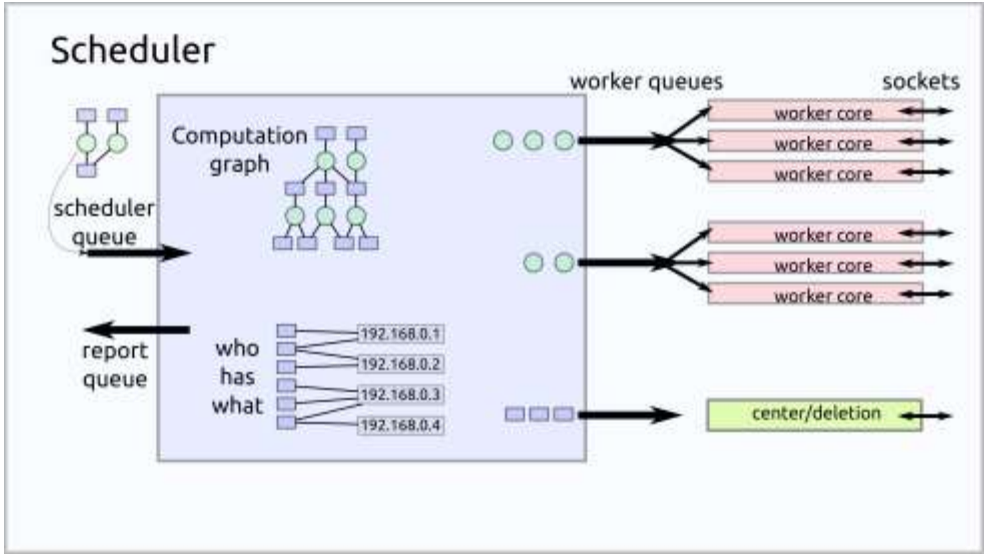
Example 3: Using Dask Arrays with global temperature data

- Built from NumPy n-dimensional arrays
- Matches NumPy interface (subset)
- Solve medium-large problems
- Complex algorithms

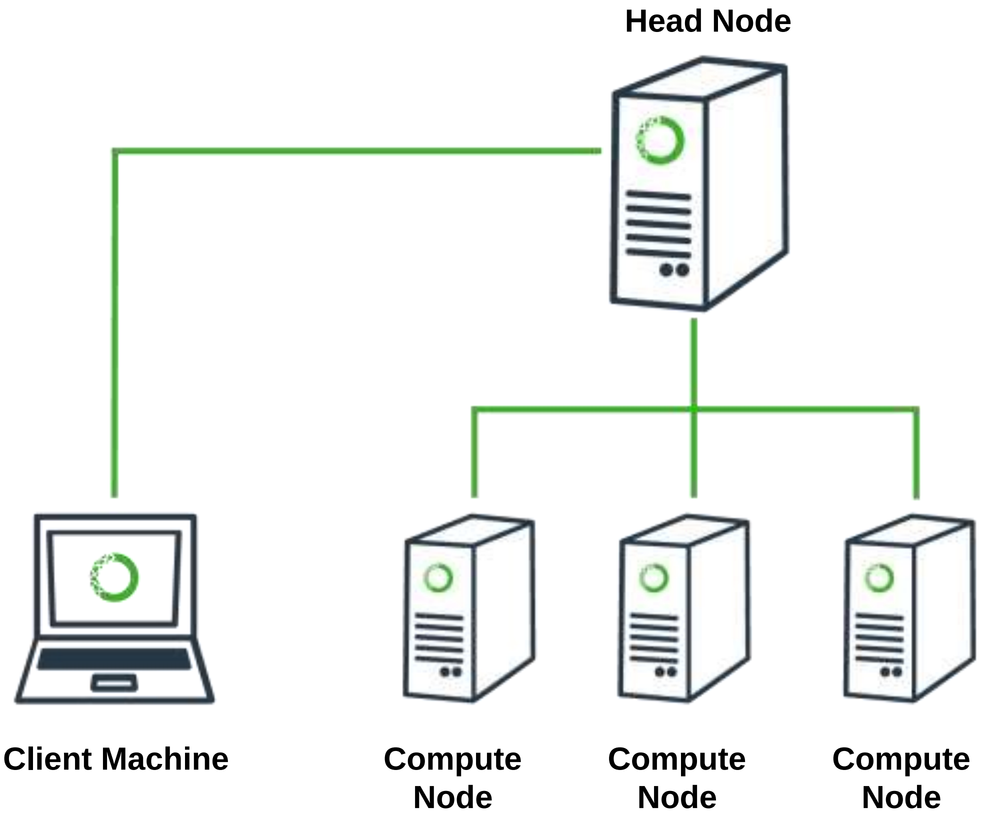


Dask Schedulers: Distributed Scheduler





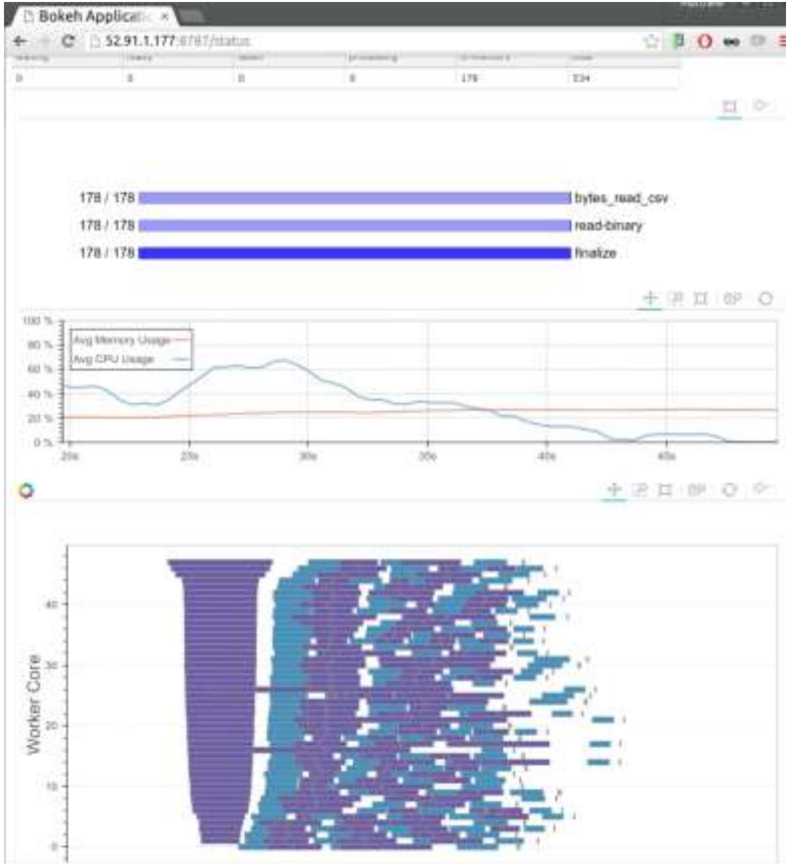
Cluster Architecture Diagram

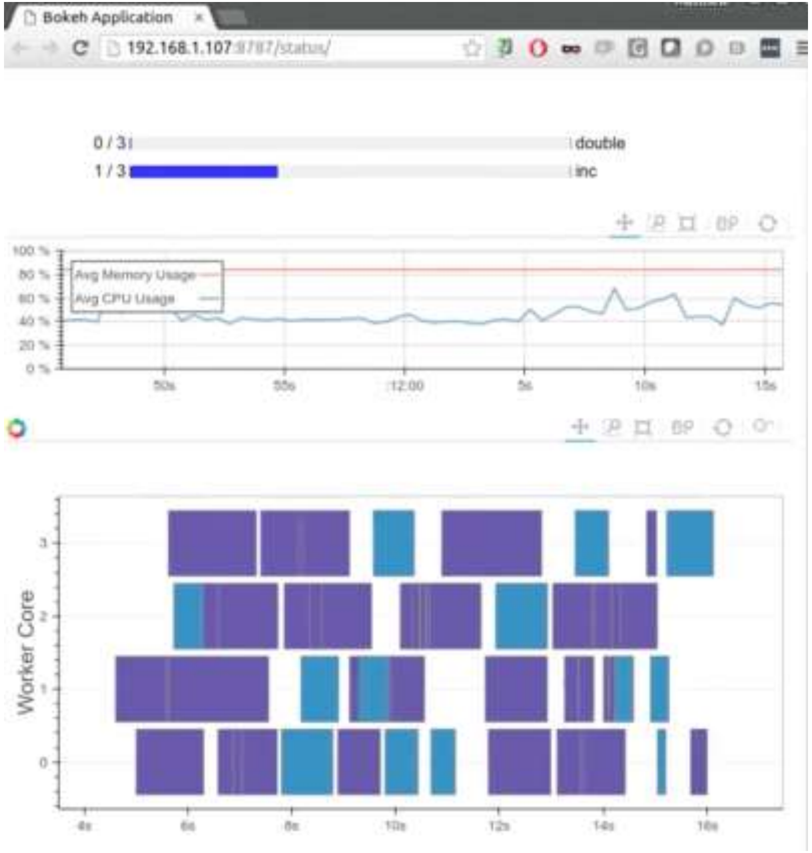


Using Anaconda and Dask on your Cluster

- Single machine with multiple threads or processes
- On a cluster with **SSH** (dcluster)
- Resource management: **YARN** (knit), **SGE**, **Slurm**
- On the cloud with **Amazon EC2** (dec2)
- On a cluster with **Anaconda for cluster management**
  - Manage multiple conda environments and packages on bare-metal or cloud-based clusters

Scheduler Visualization with Bokeh





Numba + Dask

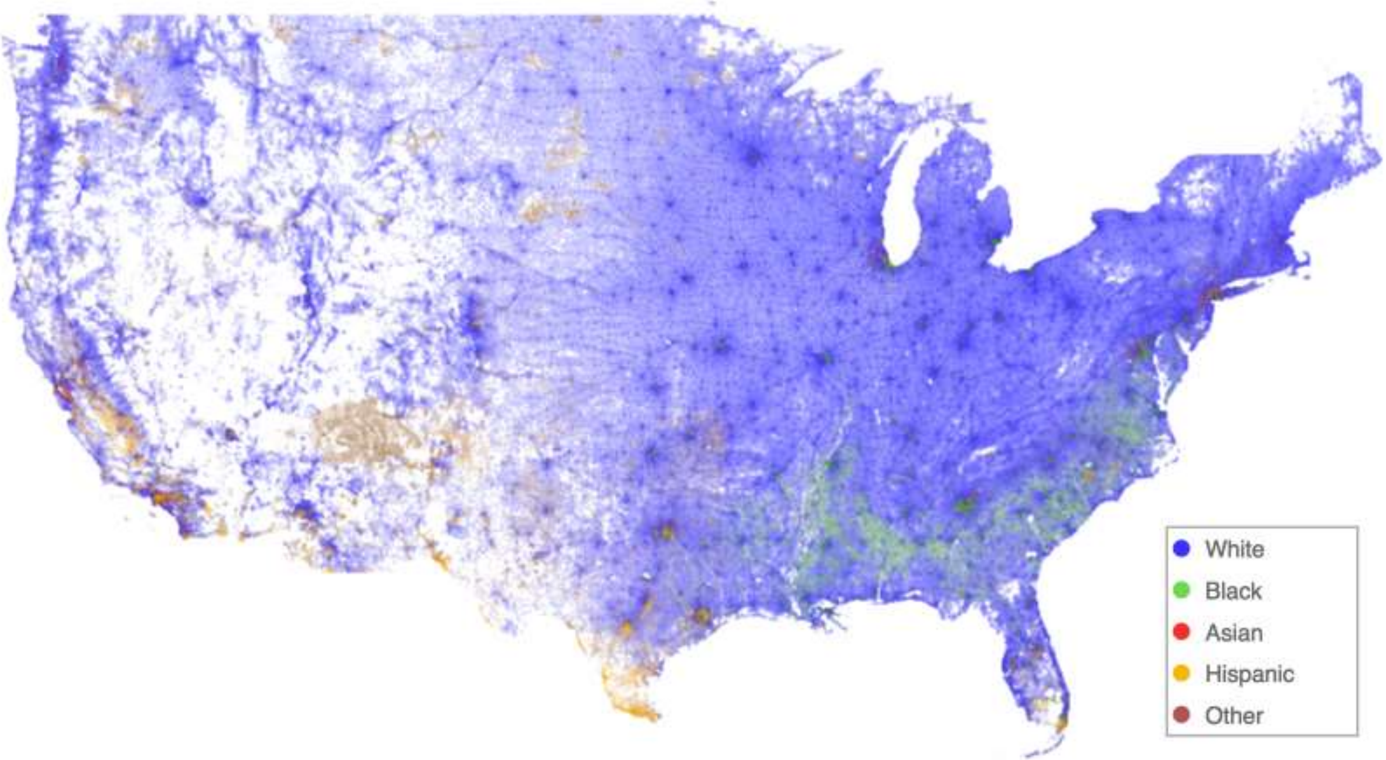
Look at **all** of the data with **Bokeh's datashader.**
Decouple the data-processing from the visualization. Visualize arbitrarily large data

E.g. Open Street Map data:

- About **3 billion** GPS coordinates
- https://blog.openstreetmap.org/2012/04/01/bulk-gps-point-data/.
- This image was rendered in one minute on a standard MacBook with 16 GB RAM
- Renders in less than a second on several 128GB Amazon EC2 instances

ANACONDA.

Categorical data: 2010 US Census



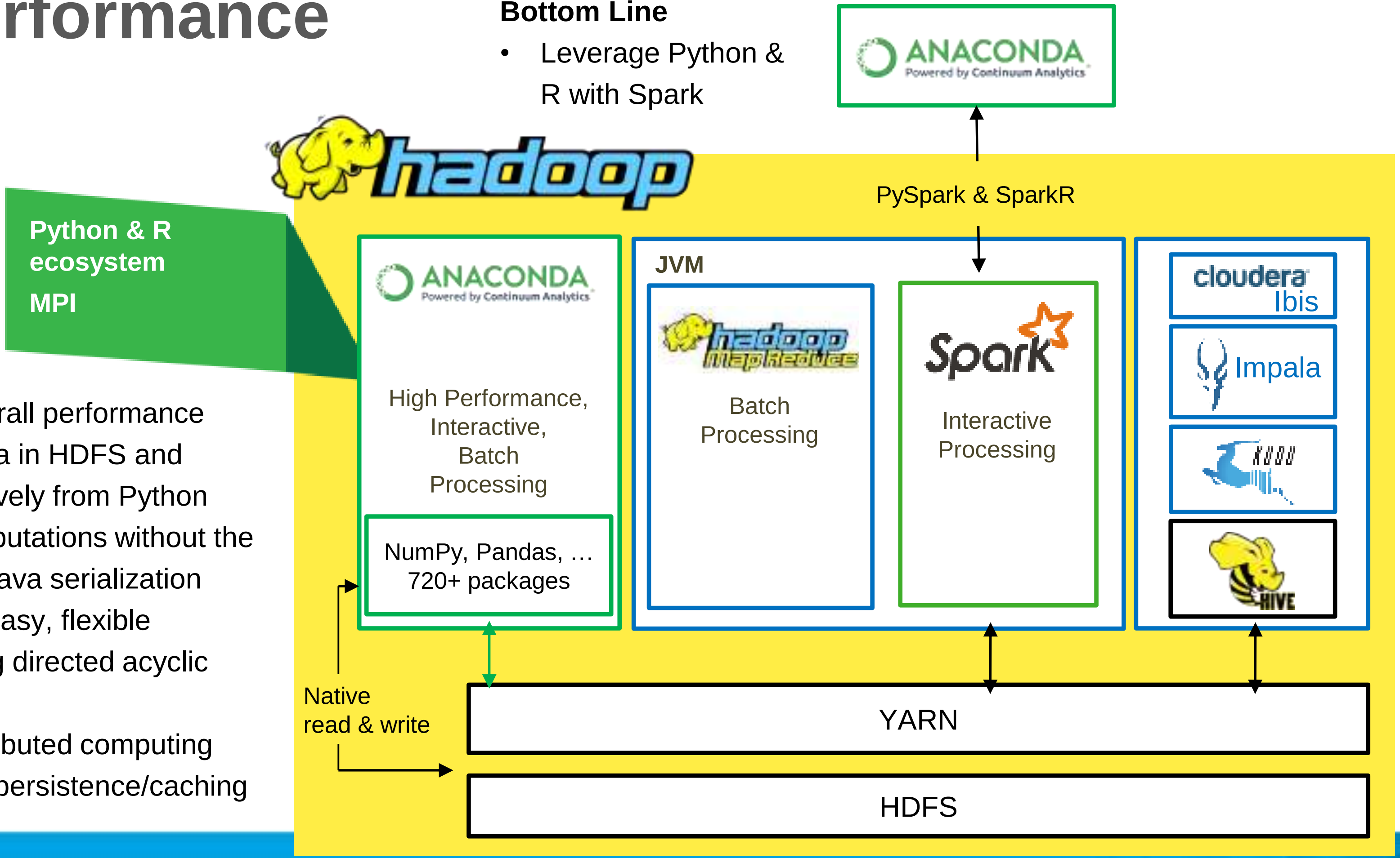
- One point per person
- 300 million total
- Categorized by race
- Interactive rendering with Numba+Dask
- No pre-tiling

Anaconda and Hadoop

High Performance Hadoop

Bottom Line

- Leverage Python & R with Spark



Bottom Line

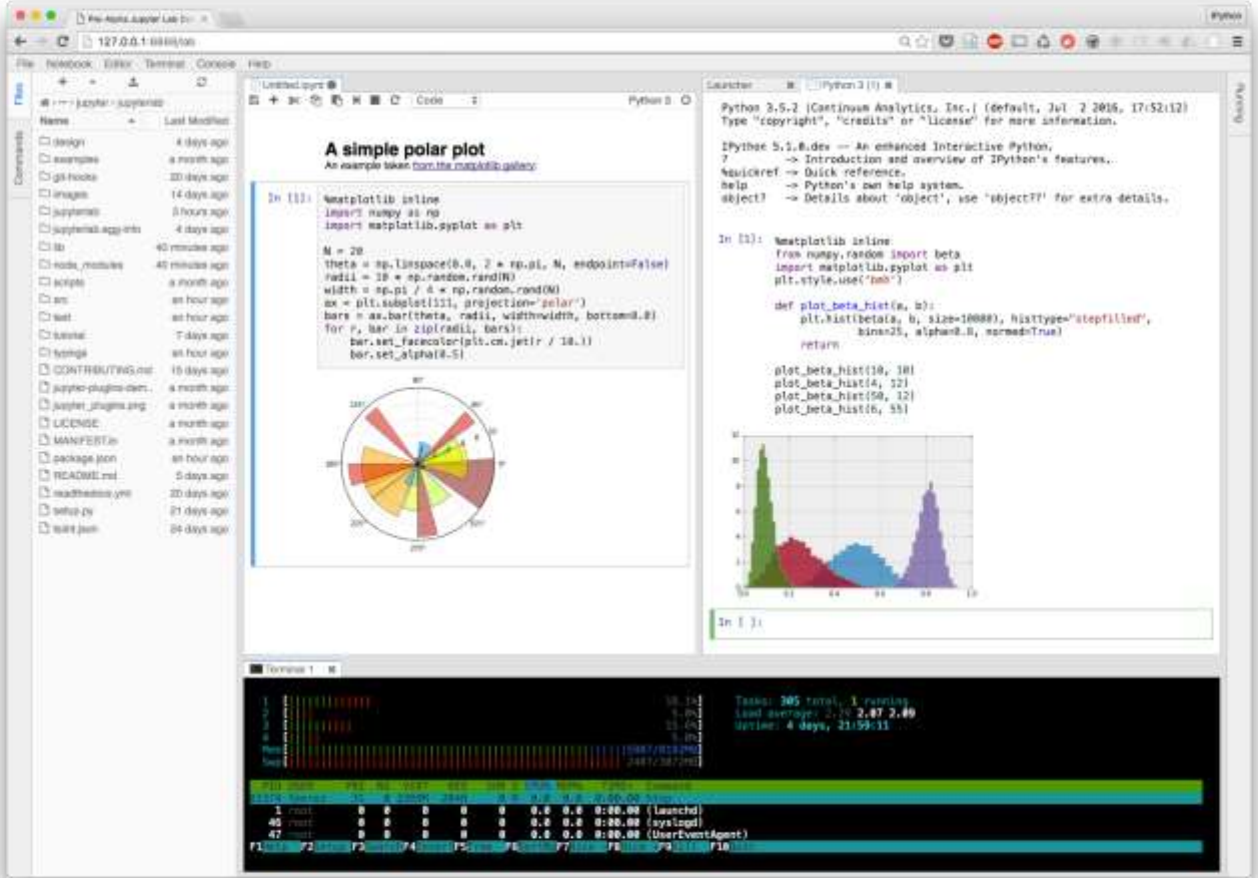
2X-100X faster overall performance

- Interact with data in HDFS and Amazon S3 natively from Python
- Distributed computations without the JVM & Python/Java serialization
- Framework for easy, flexible parallelism using directed acyclic graphs (DAGs)
- Interactive, distributed computing with in-memory persistence/caching

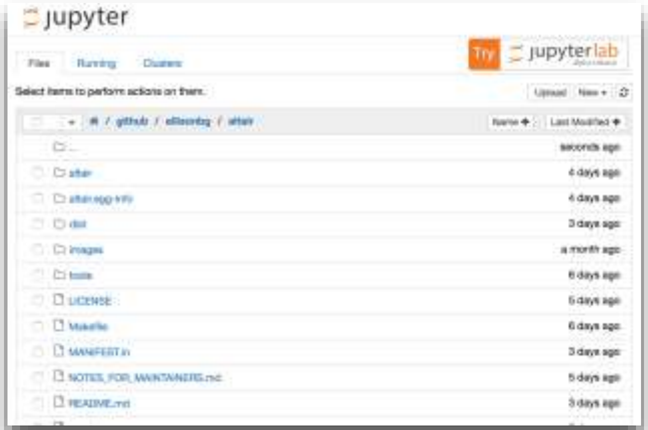
Jupyterlab (Where Jupyter is heading)

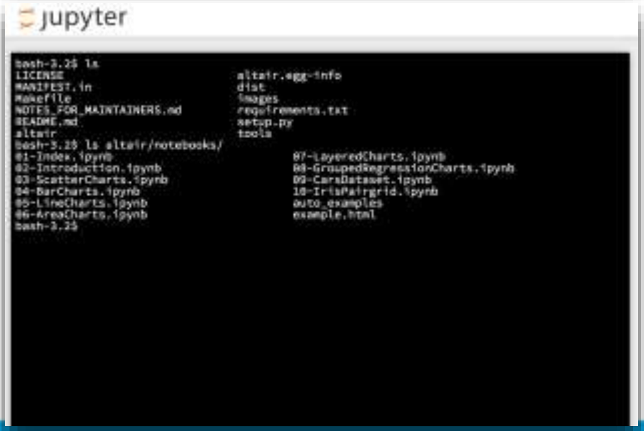
JupyterLab

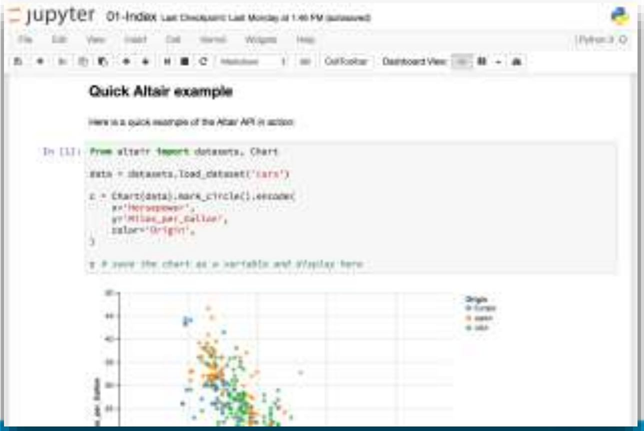
- IDE
- Extensible
- Notebook -> Applications



More Than Just Notebooks







Building Blocks

File Browser

Notebooks

Text Editor

Widgets

Output

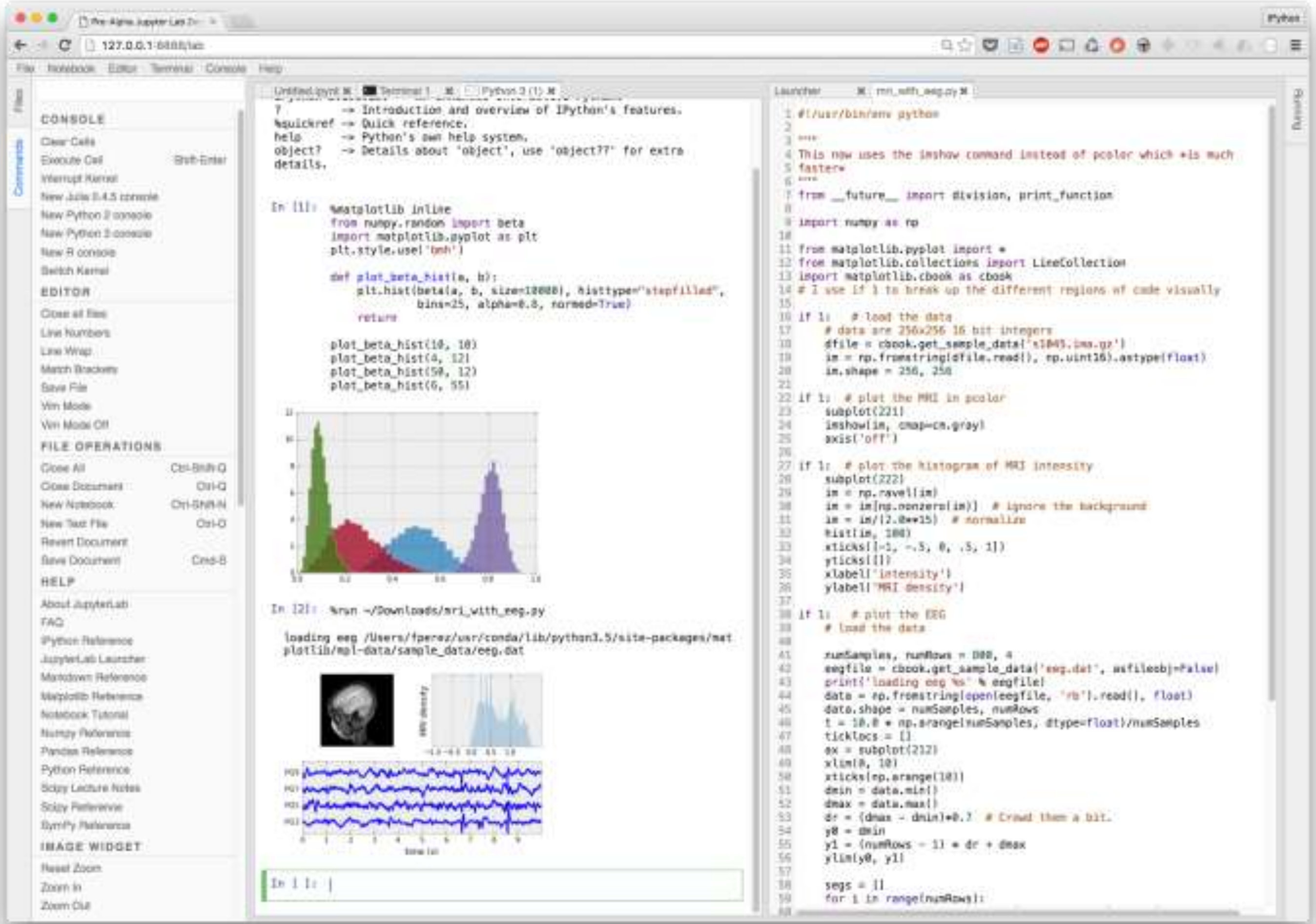
Terminal

2015 User Experience Survey

- Mostly daily/weekly users
- Love the notebook workflow and user experience
- Top needs:
  - Integration with version control systems (git/GitHub)
  - Code/text editing
  - Layout/integration of building blocks
  - Debugger, profiler, variable inspector, etc.

https://github.com/jupyter/design/blob/master/surveys/2015-notebook-ux/analysis/report_dashboard.ipynb

A completely modular architecture

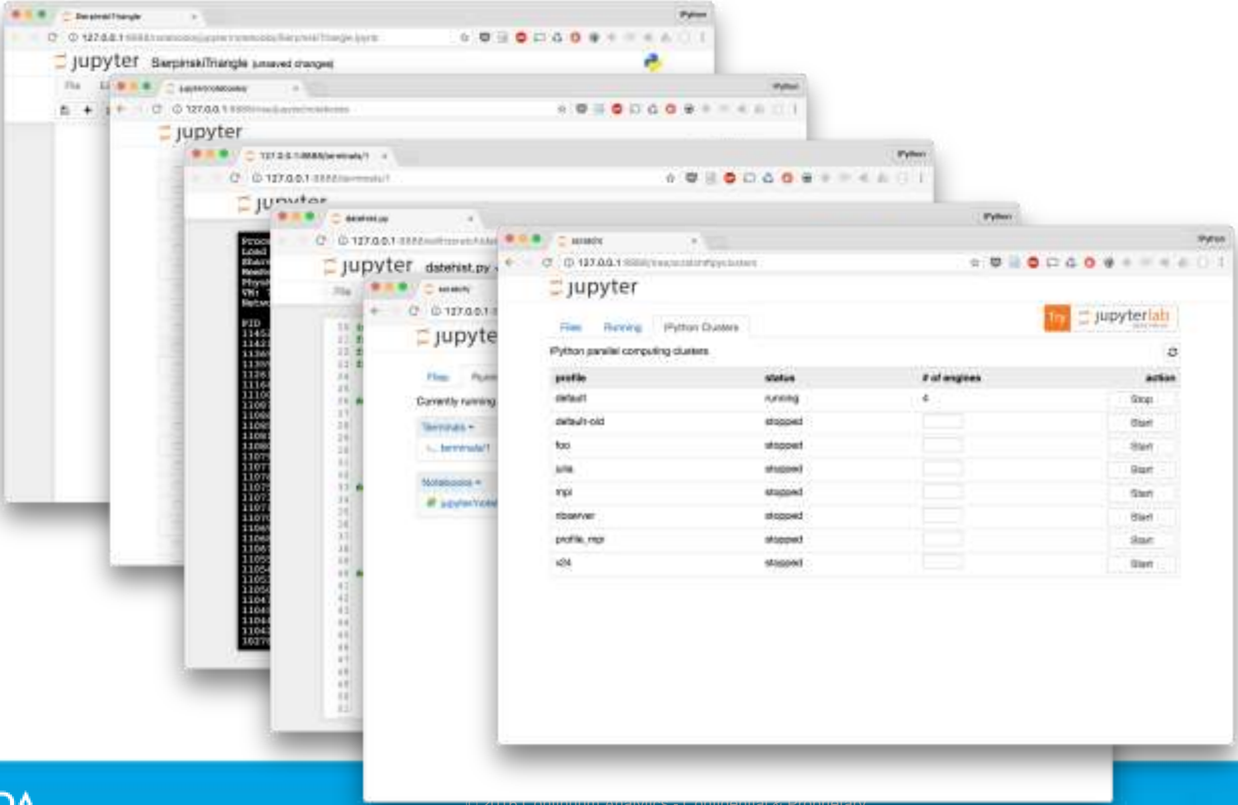


JupyterLab

- JupyterLab is the natural evolution of the Jupyter Notebook user interface
- JupyterLab is an extensible IDE: *Interactive* Data Environment
- Flexible user interface for assembling the fundamental building blocks of interactive computing
- Modernized JavaScript architecture based on npm/webpack, plugin system, model/view separation
- Built using PhosphorJS (http://phosphorjs.github.io/)
- Design-driven development process

https://github.com/jupyter/jupyterlab

The “Notebook”?



Data Science Workflow