
python
TM

What is Python?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

What is Python?

Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

History of Python

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum in the Netherlands as a successor of a language called ABC. Guido remains Pythons principal author, although it includes many contributions from others.

WORK

Python Displacing R As The Programming Language For Data Science

R remains popular with the PhDs of data science, but as data moves mainstream, Python is taking over.

MATT ASAY · NOV 25, 2013


PyData

PyData : Conference Mission

- PyData is a gathering of users and developers of data analysis tools in Python.
- The goals are to provide Python enthusiasts a place to share ideas and learn from each other about how best to apply our language and tools to ever-evolving challenges in the vast realm of data management, processing, analytics, and visualization.

PyData

- We aim to be an accessible, community-driven conference, with tutorials for novices, advanced topical workshops for practitioners, and opportunities for package developers and users to meet in person.
- A major goal of the conference is to provide a venue for users across all the various domains of data analysis to share their experiences and their techniques, as well as highlight the triumphs and potential pitfalls of using Python for certain kinds of problems.

CONTINUUM

ANALYTICS

PyData
Berlin 2015

Register | CFP | Venue | News | Speakers | Schedule

PyData Berlin 2015

May 29-30

PyData is the home for all things related to the use of Python in data management and analysis. It brings together Python enthusiasts at a novice level and includes Tutorials and corresponding talks as well as advanced talks by experts and package developers. The conference not only focuses on the application of data science tools but also on underlying algorithms and patterns. After a very successful PyData Berlin 2014 hosted at the BCC we are thrilled to announce PyData Berlin 2015, this time taking place at Betahaus Berlin.

Registration is now open using Eventbrite.

We have now opened our CFP and are looking forward to your contributions.

For upcoming news please follow pydataberlin on Twitter.

PyCon Ireland 2014

Home PyCon 2014 Location Sponsorship Conference Sprints

About

Introduction

This years PyCon Ireland 2014 will be held on **Sat 11th - Sun 12th October** in the **Ballsbridge Hotel**. Followed by two days of **Sprints on Mon 13th and Tuesday 14th October** also in the Ballsbridge Hotel.

PyData London Meetup

Home Members Sponsors Photos Discussions More My profile

PyData London

London, United Kingdom

Founded Apr 23, 2014

About us...

members	1,201
Group reviews	8
Past Meetups	10
Our calendar	

Welcome!

+ SUGGEST A NEW MEETUP

Upcoming Past Calendar

We're still planning a Meetup

Want to come?

Contact the Organizer

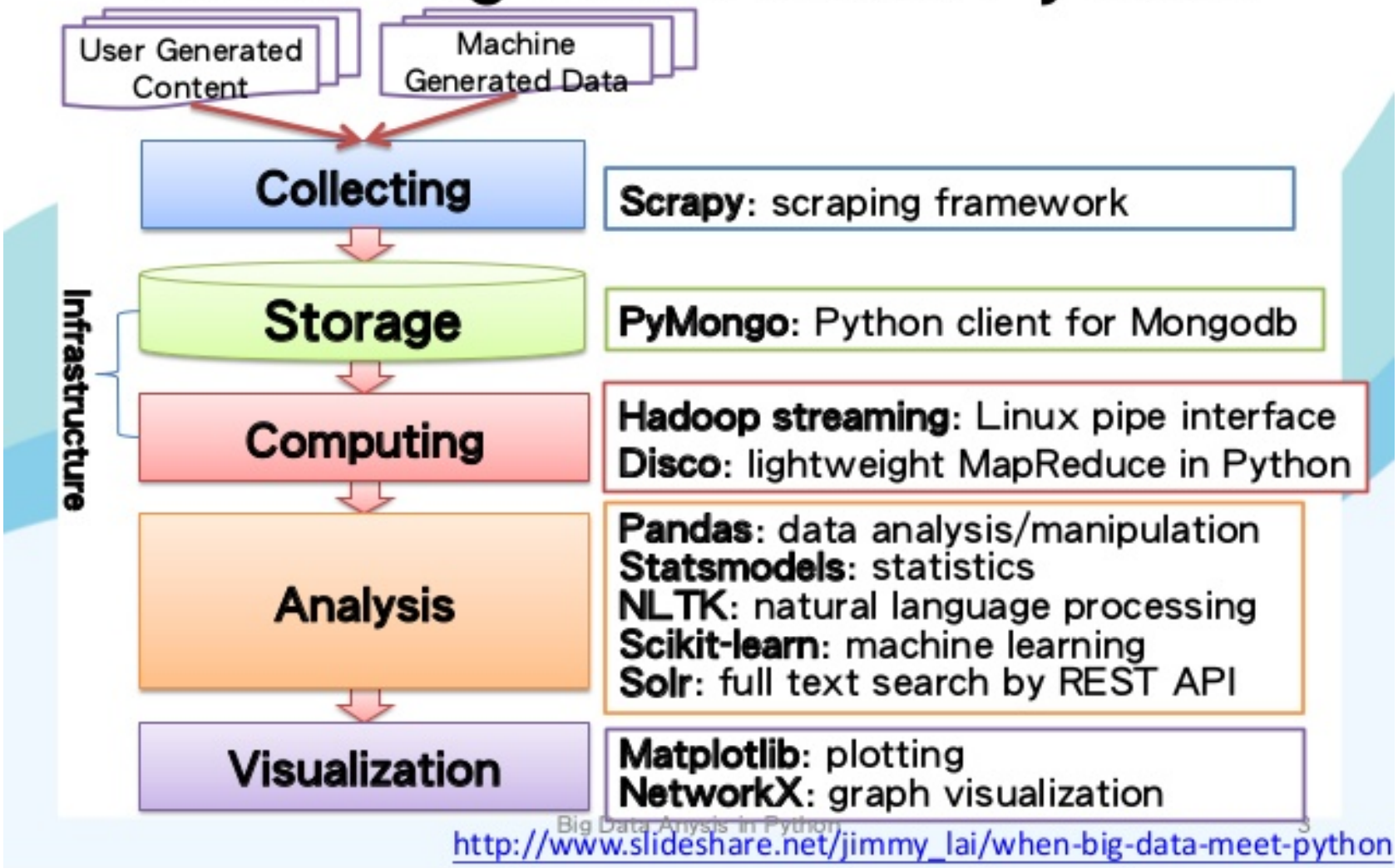
Recent Meetups

6 days ago · 7:00 PM

What's new

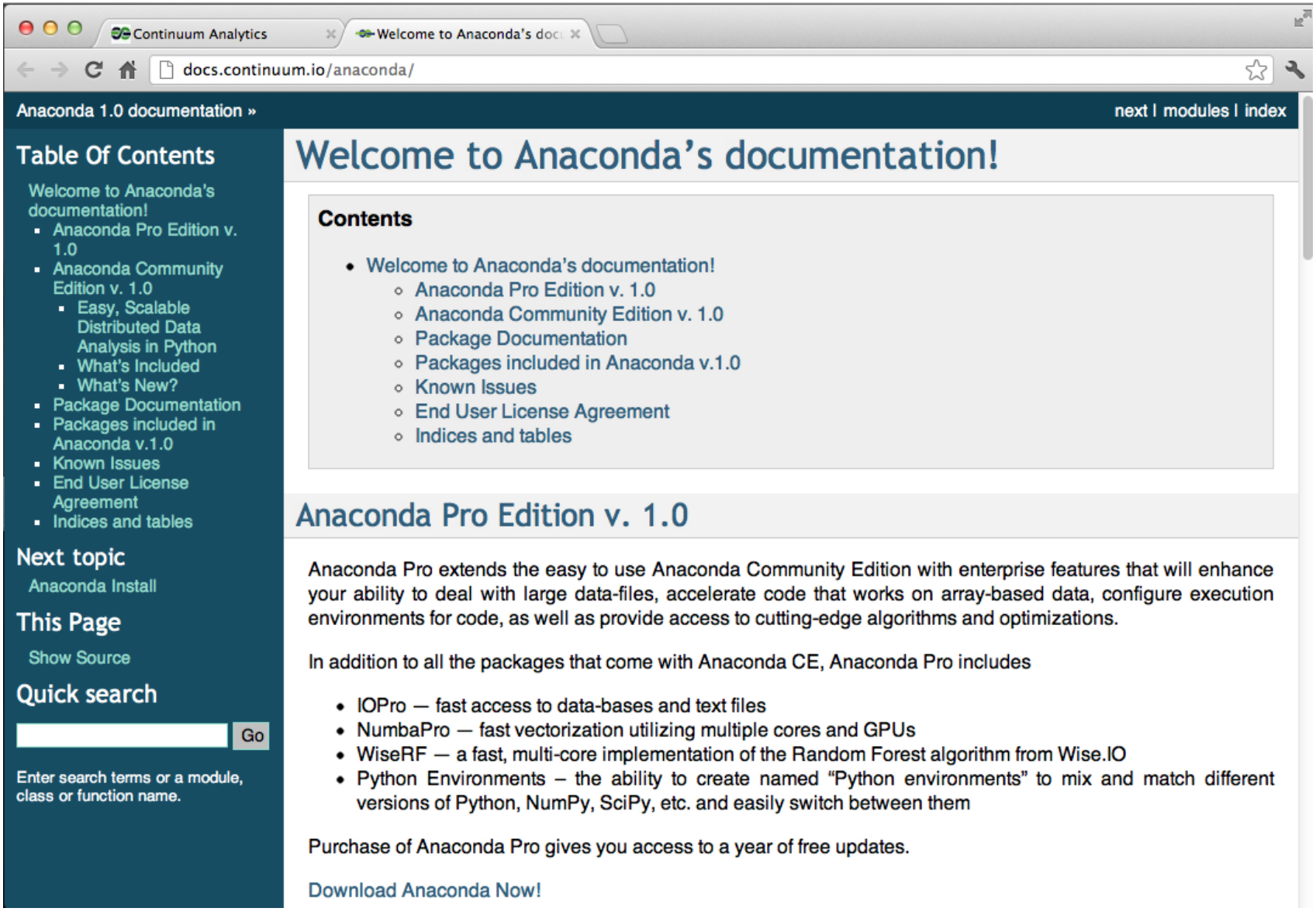
Important Components of the Python Scientific Stack

When Big Data meet Python



http://www.slideshare.net/jimmy_lai/when-big-data-meet-python

Continuum Analytics Anaconda



Continuum Analytics Anaconda

- Anaconda, a free product of Continuum Analytics (www.continuum.io), is a virtually complete scientific stack for Python.
- It includes both the core Python interpreter and standard libraries as well as most modules required for data analysis.

Continuum Analytics Anaconda

- Anaconda is free to use and modules for accelerating the performance of linear algebra on Intel processors using the **Math Kernel Library** (MKL) are available (free to academic users and for a small cost to non-academic users).
- Continuum Analytics also provides other high-performance modules for reading large data files or using the GPU to further accelerate performance for an additional, modest charge.

Installing Anaconda

Most importantly, installation is extraordinarily easy onWindows, Linux and OS X. Anaconda is also simple to update to the latest version using

```
conda update conda
conda update anaconda
```

NumPy and SciPy

- **NumPy** provides a set of array and matrix data types which are essential for statistics and econometrics.
- **SciPy** contains a large number of routines needed for analysis of data.The most important include a wide range of random number generators, linear algebra routines and optimizers.
- Remark: SciPy depends on NumPy.
- More on them later.

IPython

IPython provides an interactive Python environment which enhances productivity when developing code or performing interactive data analysis.

IPy IPython Dashboard | IPy IPython Dashboard | IPy Untitled0

127.0.0.1:8889/a5989266-c452-4c65-b962-d921b364b3e0

IP[y]: Notebook Untitled0 Last saved: Jun 02 2:02 PM

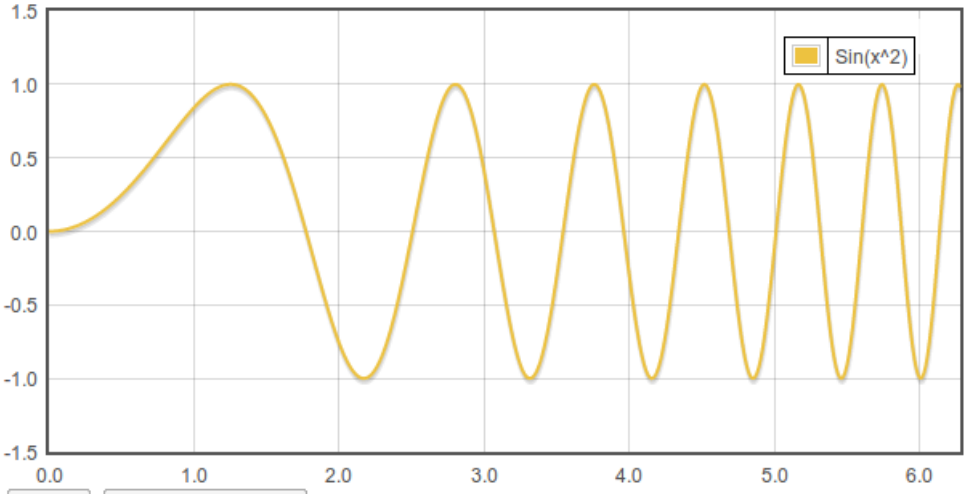
File Edit View Insert Cell Kernel Help

Code

In [8]:
```
import flot
```

In [9]:
```
fp = flot.Plot()
fplot = fp.plot_figure
```

In [10]:
```
x = linspace(0, 2*pi, 500)
y = sin(x**2)
fplot(x, y, label='Sin(x^2)')
```



Ipython Notebook / Jupyter

Evolved from the IPython Project

The language-agnostic parts of IPython are getting a new home in Project Jupyter

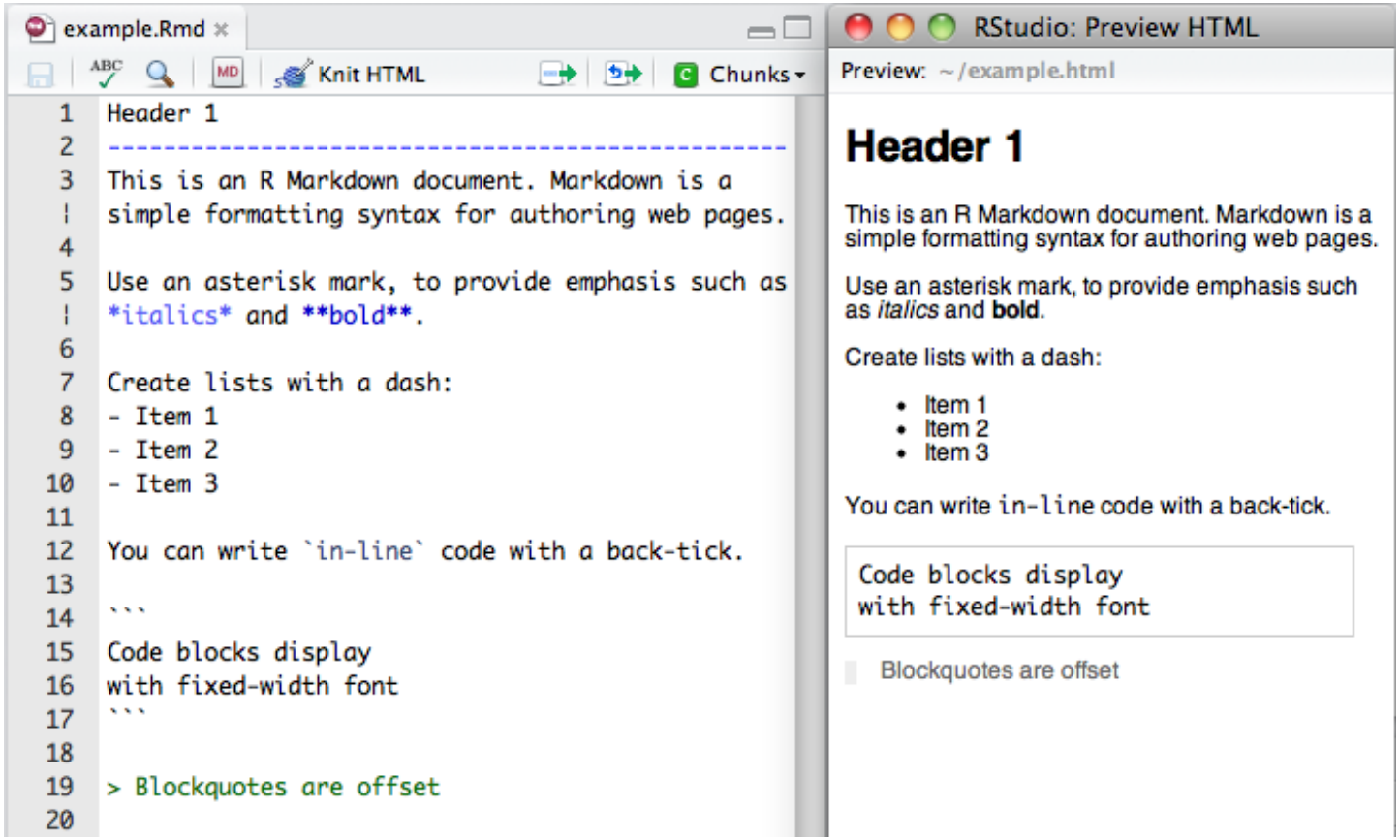
IPython

- Interactive Python shell
- Python kernel for Jupyter
- Interactive Parallel Python

Jupyter

- Rich REPL Protocol
- Notebook (format, environment, conversion)
- JupyterHub (multi-user notebook server)
- More...

Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML).



matplotlib and seaborn

Graphics Packages

- **matplotlib** provides a plotting environment for 2D plots, with limited support for 3D plotting.
- **seaborn** is a Python package that improves the default appearance of matplotlib plots without any additional code.

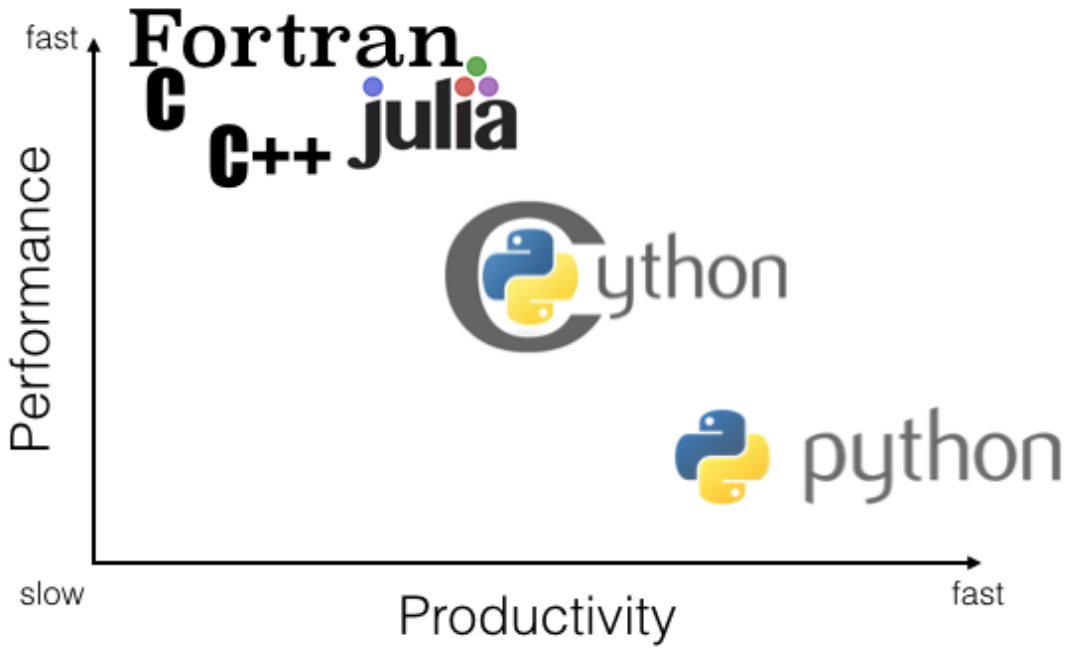
pandas

- *pandas* is a high-performance module that provides a comprehensive set of structures for working with data.
- *pandas* excels at handling structured data, such as data sets containing many variables, working with missing values and merging across multiple data sets.

pandas

- While extremely useful, *pandas* is not an essential component of the Python scientific stack unlike NumPy, SciPy or matplotlib, and so while *pandas* doesnt make data analysis possible in Python, it makes it much easier.
- *pandas* also provides high-performance, robust methods for importing from and exporting to a wide range of formats.
- - example `read.csv()`


Cython


fast
Performance
Fortran
C
C++
julia
Cython
python
slow
Productivity
fast

Performance Modules : Cython and Numba

A number of modules are available to help with performance. These include Cython and Numba.

Cython Cython is a Python module which facilitates using a simple Python-derived creole to write functions that can be compiled to native (C code) Python extensions.

Numba Numba uses a method of just-in-time compilation to translate a subset of Python to native code using *Low-Level Virtual Machine* (LLVM).

Versions of Python

Two Main Versions of Python

- Version 2.7
- Version 3

Python Coding Conventions

There are a number of common practices which can be adopted to produce Python code which looks more like code found in other modules:

- Use 4 spaces to indent blocks avoid using tab, except when an editor automatically converts tabs to 4 spaces
- Avoid more than 4 levels of nesting, if possible
- Limit lines to 79 characters. The \ symbol can be used to break long lines 219
- Use two blank lines to separate functions, and one to separate logical sections in a function.

Python Coding Conventions

- Use ASCII mode in text editors, not UTF-8
- One module per import line
- Avoid from module `import *` (for any module). Use either `from module import func1, func2` or `import module as shortname`.
- Follow the NumPy guidelines for documenting functions

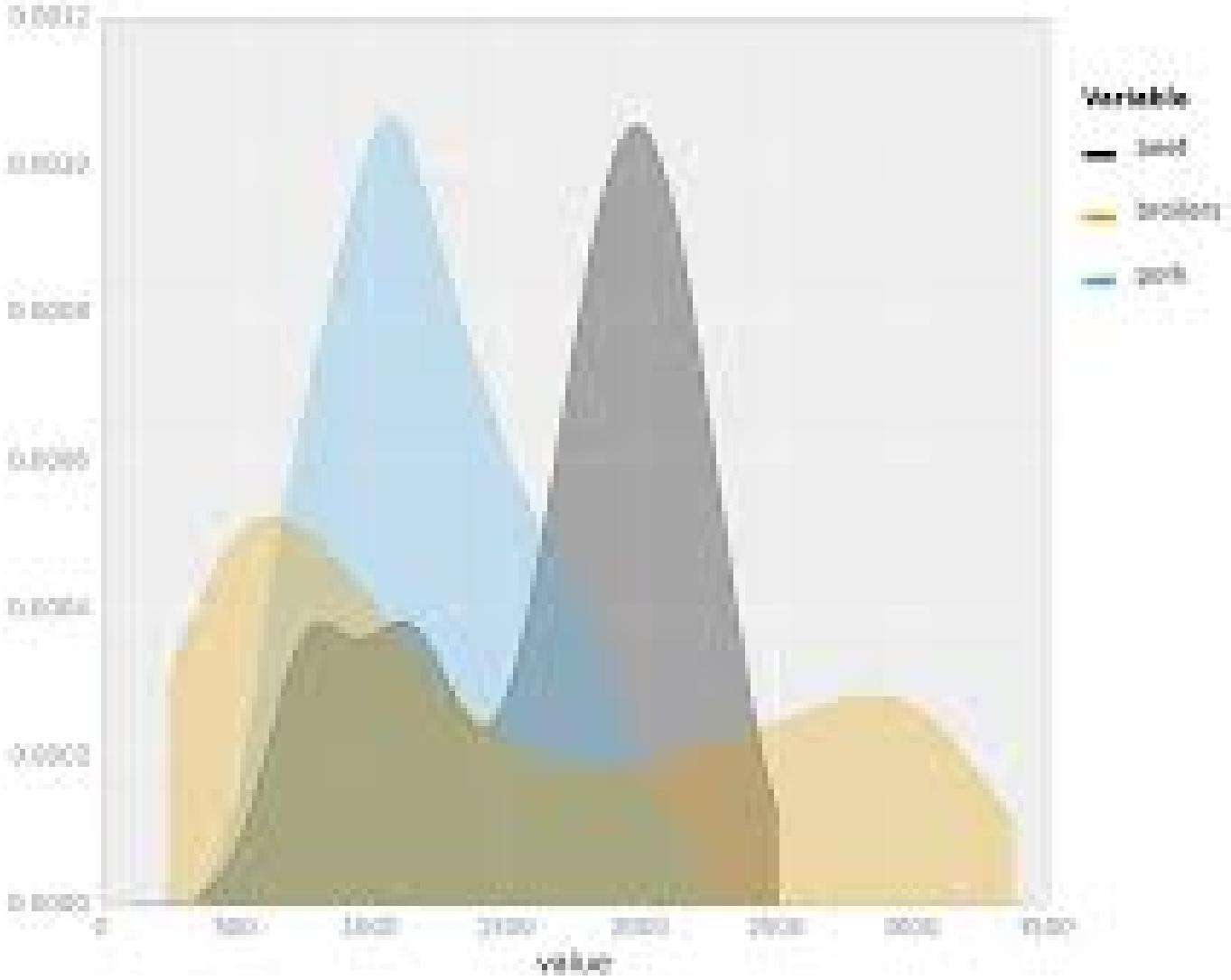
Other Interesting Python Packages

statsmodels

- `statsmodels` provides a large range of cross-sectional models aswell assometime-series models.
- statsmodels uses a model descriptive language (provided via the Python package patsy) to formulate the model when working with pandas `DataFrames`.
- Models supported include linear regression, generalized linear models, limited dependent variable models, ARMA and VAR models.

Bokeh Data Visualization

Bokeh Data Visualization



Bokeh Data Visualization

Bokeh Data Visualization

- interactive graphics for the web
- designed for large data sets
- Designed for streaming data
- Native interface in pythin
- Fast javascript components
- DARPA funded
- v.01 relase imminent


scikit
learn

scikit.learn

- scikit-learn is an open source machine learning library for the Python programming language.
- scikit-learn features various classification, regression and clustering algorithms including support vector machines, logistic regression, naive Bayes, random forests, gradient boosting, k-means and DBSCAN.
- scikit-learn is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Sci-Kit Learn Site info

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: *SVM*, *nearest neighbors*, *random forest*, ... — *Examples*

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: *SVR*, *ridge regression*, *Lasso*, ... — *Examples*

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: *k-Means*, *spectral clustering*, *mean-shift*, ... — *Examples*

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: *PCA*, *feature selection*, *non-negative matrix factorization*. — *Examples*

Model selection

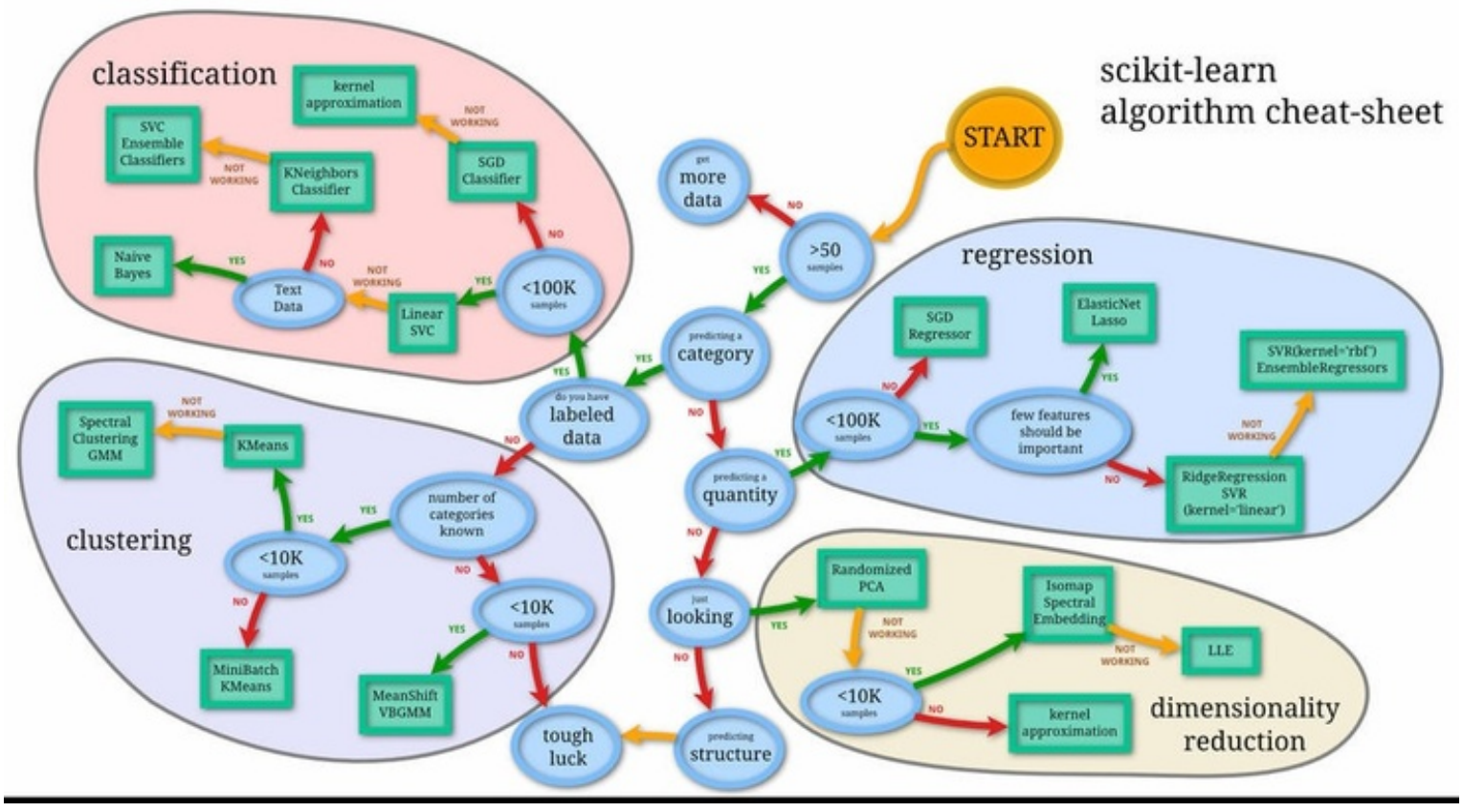
Comparing, validating and choosing parameters and models.

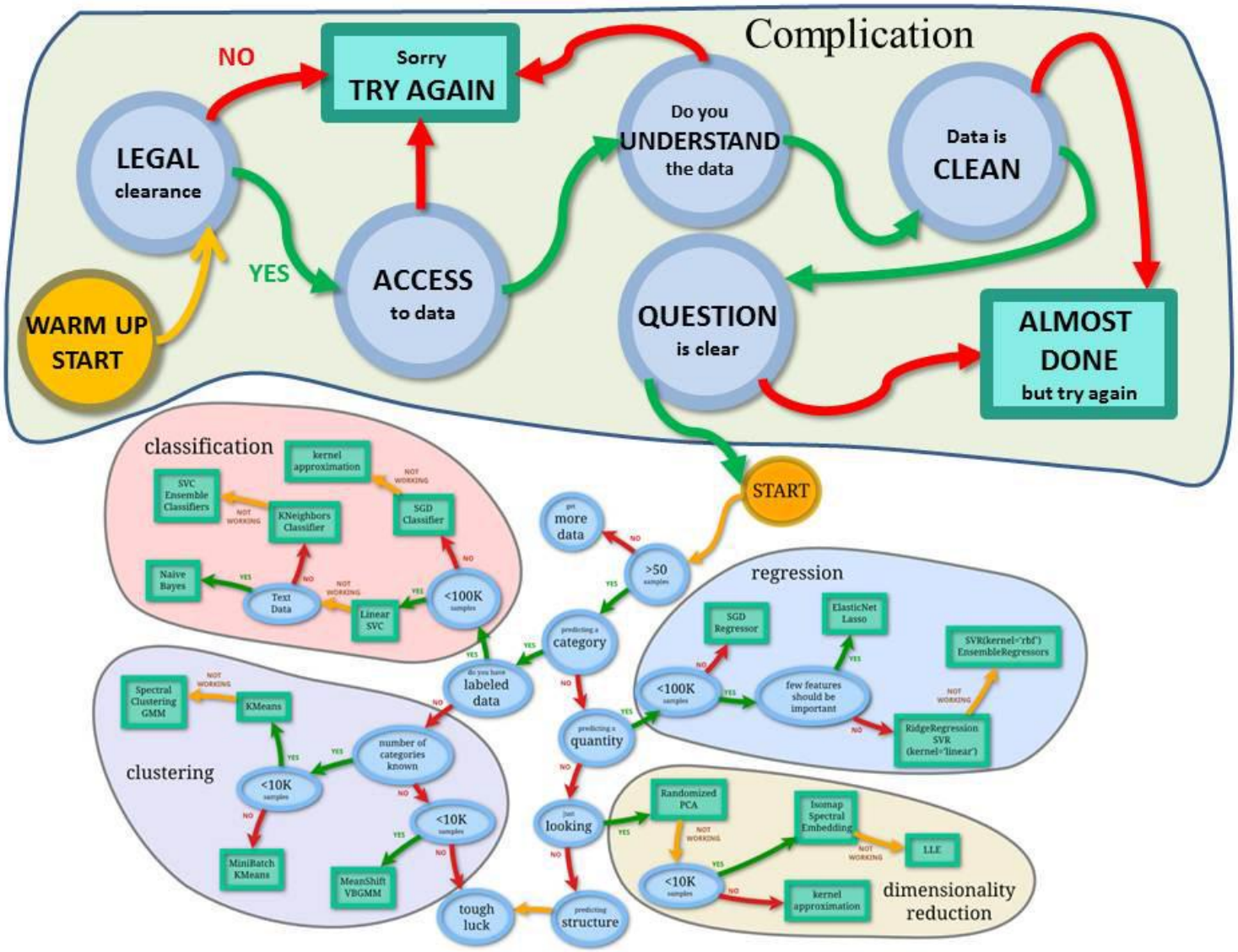
Goal: Improved accuracy via parameter tuning
Modules: *grid search*, *cross validation*, *metrics*. — *Examples*

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: *preprocessing*, *feature extraction*. — *Examples*


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


Complication
NO
Sorry
TRY AGAIN
LEGAL
clearance
YES
ACCESS
to data
Do you
UNDERSTAND
the data
Data is
CLEAN
WARM UP
START
QUESTION
is clear
ALMOST
DONE
but try again
START
classification
kernel approximation
SVC Ensemble Classifiers
KNeighbors Classifier
SGD Classifier
Naive Bayes
Text Data
Linear SVC
<100K
get more data
>50
predicting a category
labeled data
regression
SGD Regressor
ElasticNet Lasso
SVR(kernel='rbf') EnsembleRegressors
<100K
few features should be important
RidgeRegression SVR (kernel='linear')
clustering
Spectral Clustering GMM
KMeans
number of categories known
<10K
MiniBatch KMeans
<10K
MeanShift VBGMM
predicting a quantity
just looking
Randomized PCA
Isomap Spectral Embedding
LLE
<10K
kernel approximation
dimensionality reduction
tough luck
predicting structure

Simon Blomberg:

> From R's fortunes package: To paraphrase provocatively, 'machine learning is statistics minus any checking of models and assumptions'. -- Brian D. Ripley (about the difference between machine learning and statistics) useR! 2004, Vienna (May 2004) :-) Season's Greetings!

Andrew Gelman:

> In that case, maybe we should get rid of checking of models and assumptions more often. Then maybe we'd be able to solve some of the problems that the machine learning people can solve but we can't!

Machine Learning is statistics minus any checking of models or assumptions

The Data Science Profession

Data Science Retreat (Berlin)

> *MOOC have not decreased the barrier of entry to machine-learning.*
> *Nowadays, you cannot be 'the guy who knows how to run (insert off-the-shelf-algo-here)'.*
> *In dataland, that's the equivalent to being a code monkey. MOOCs and superb libraries (scikit-learn, R's ecosystem) made sure there is plenty of people who can throw say a random forest to a problem. In the modern world, this is not adding that much value.*

Other Packages

pytz and babel
ptyz and babel provide extended support for time zones and formatting information.

rpy2
rpy2 provides an interface for calling R 3.0.x in Python, as well as facilities for easily moving data between the two platforms.

PyTables and h5py
PyTables and h5py both provide access to HDF5 files, a flexible data storage format optimized for numeric data.

Three main packages

1. numpy
2. pandas
3. scipy

The numpy package

- The Python programming language was not initially designed for numerical computing, but attracted the attention of the scientific/engineering community early on.
- NumPy is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

- The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers.
- In 2005, Travis Oliphant created NumPy by incorporating features of Numarray into Numeric with extensive modifications.

The numpy package

- NumPy is open source and has many contributors.
- **Website** http://www.numpy.org/

Useful Commands for simulation exercises

- `random.randint(a, b)` - Return a random integer N such that $a \leq N \leq b$.
- `random.choice(seq)` - return a random element from the non-empty sequence `seq`. If `seq` is empty, raises `IndexError`.
- `random.sample(population, k)` - Return a k length list of unique elements chosen from the population sequence. Used for random *sampling without replacement*.

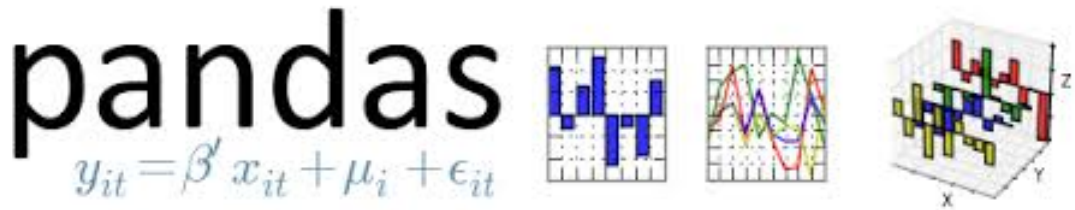
Array Creation

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x
array([1, 2, 3])
>>> y = np.arange(10)  # like Python's range, but returns an arra
>>> y
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Basic Operations

```
>>> a = np.array([1, 2, 3, 6])
>>> b = np.linspace(0, 2, 4)  # create an array with 4 equally spaced poi
>>> c = a - b
>>> c
array([ 1.        ,  1.33333333,  1.66666667,  4.        ])
>>> a**2
array([ 1,  4,  9, 36])
```

pandas



- pandas is a high-performance module that provides a comprehensive set of structures for working with data.
- pandas excels at handling structured data, such as data sets containing many variables, working with missing values and merging across multiple data sets.

Data Wrangling with Pandas, NumPy, and IPython

Python for Data Analysis

pandas

- While extremely useful, pandas is not an essential component of the Python scientific stack unlike NumPy, SciPy or matplotlib, and so while pandas doesnt make data analysis possible in Python, it makes it much easier.
- pandas also provides high-performance, robust methods for importing from and exporting to a wide range of formats.

pandas

Data Structures

pandas provides a set of data structures which include Series, DataFrames and Panels.

- **Series** are 1-dimensional arrays.
- **DataFrames** are collections of Series and so are 2-dimensional,
- **Panels** are collections ofDataFrames, and so are 3-dimensional.

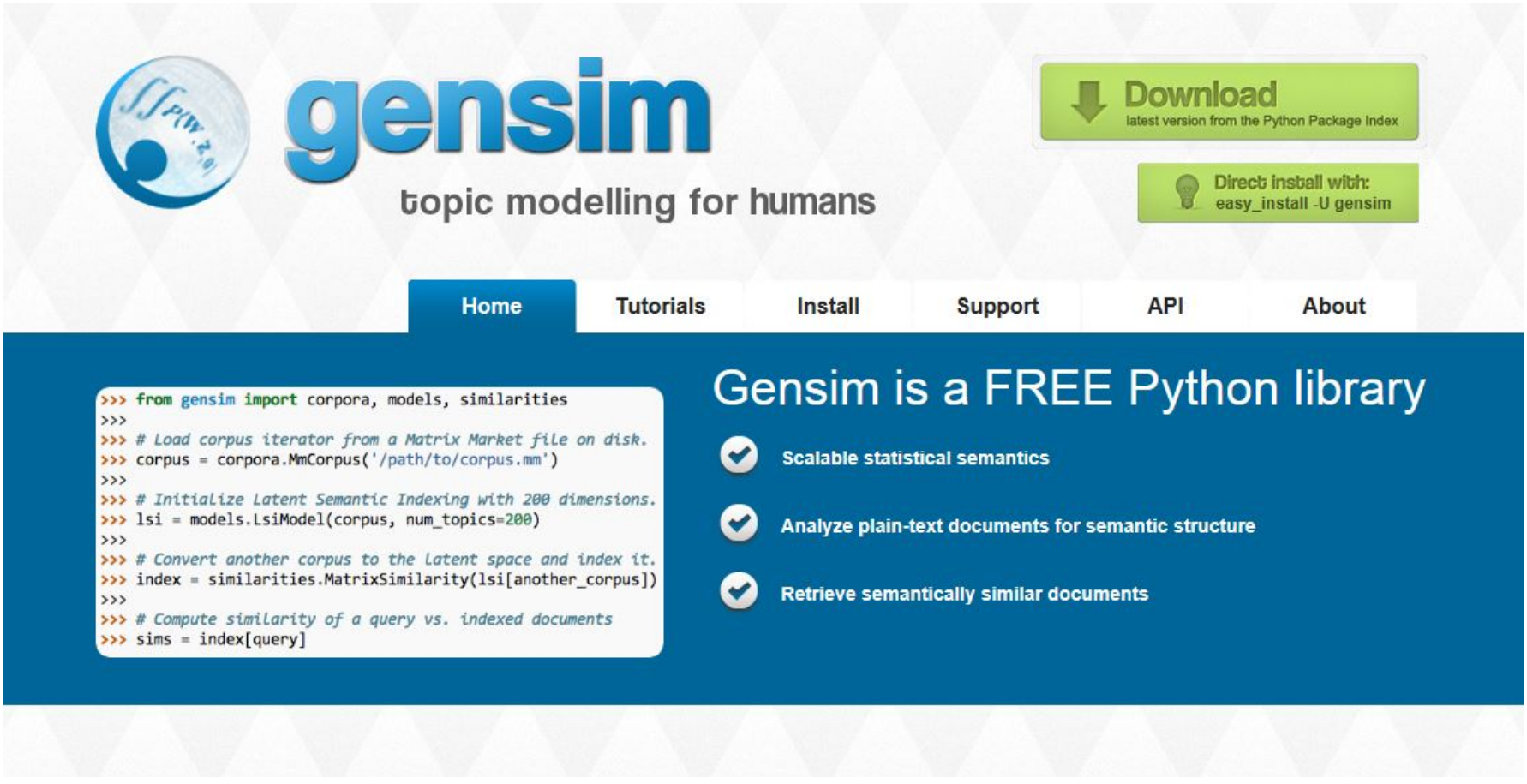
SciPy

sciPy

- SciPy (pronounced Sigh Pie) is an open source Python library used by scientists, analysts, and engineers doing scientific computing and technical computing.
- SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.


Text
Mining
Analysis
Data
Information
Software
Research
Word
Analytics
Biomedical
Entity
Business
Pattern
Such
New
Application
Extract
Document
Search
Process
Need
Intelligent
Content
Describe
Sentiment

gensim


gensim
topic modelling for humans
Download
latest version from the Python Package Index
Direct install with:
easy_install -U gensim
Home
Tutorials
Install
Support
API
About
>>> from gensim import corpora, models, similarities
>>>
>>> # Load corpus iterator from a Matrix Market file on disk.
>>> corpus = corpora.MmCorpus('/path/to/corpus.mm')
>>>
>>> # Initialize Latent Semantic Indexing with 200 dimensions.
>>> lsi = models.LsiModel(corpus, num_topics=200)
>>>
>>> # Convert another corpus to the latent space and index it.
>>> index = similarities.MatrixSimilarity(lsi[another_corpus])
>>>
>>> # Compute similarity of a query vs. indexed documents
>>> sims = index[query]
Gensim is a FREE Python library
Scalable statistical semantics
Analyze plain-text documents for semantic structure
Retrieve semantically similar documents

gensim

The tutorials are organized as a series of examples that highlight various features of *gensim*. It is assumed that the reader is familiar with the **Python language**, has ***installed gensim*** and read the ***introduction***.

The examples are divided into parts on:

Beautiful Soup

Table Of Contents

Beautiful Soup Documentation

Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

These instructions illustrate all major features of Beautiful Soup 4, with examples. I show you what the library is good for, how it works, how to use it, how to make it do what you want, and what to do when it violates your expectations.

The examples in this documentation should work the same way in Python 2.7 and Python 3.2.

You might be looking for the documentation for Beautiful Soup 3. If so, you should know that Beautiful Soup 3 is no longer being developed, and that Beautiful Soup 4 is recommended for all new projects. If you want to learn about the differences between Beautiful Soup 3 and Beautiful Soup 4, see Porting code to BS4.

This documentation has been translated into other languages by Beautiful Soup users:

- 这篇文档当然还有中文版.

Beautiful Soup

Beautiful Soup

- Beautiful Soup is a Python library for pulling data out of HTML and XML files.
- It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree.
- It commonly saves programmers hours or days of work.

Data Structures

pandas introduces two new data structures to Python - **Series** and **DataFrame**, both of which are built on top of NumPy.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
pd.set_option('max_columns', 50)
```

Series

Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index. The basic method to create a Series is to call:

```
s = Series(data, index=index)
```

Here, data can be many different things:

- a Python `dict`
- an `ndarray`
- a scalar value (like 5)

- A Series is a one-dimensional object similar to an array, list, or column in a table.
- It will assign a labeled index to each item in the Series.
- By default, each item will receive an index label from 0 to N, where N is the length of the Series minus one.

```
# create a Series with an arbitrary list
s = pd.Series([7, 'Heisenberg', 3.14, -1789710578,
     'Happy Eating!'])
s
```

Series

Output from Previous Slide

```
0                   7
1        Heisenberg
2              3.14
3       -1789710578
4    Happy Eating!
dtype: object
```

Alternatively, you can specify an index to use when creating the Series.

```
s = pd.Series([7, 'Heisenberg', 3.14, -1789710578,
   'Happy Eating!'],
index=['A', 'Z', 'C', 'Y', 'E'])
s
```

```
A                 7
Z       Heisenberg
C             3.14
Y      -1789710578
E    Happy Eating!
dtype: object
```

Series

The Series constructor can convert a dictonary as well, using the keys of the dictionary as its index.

```
d = {'Chicago': 1000, 'New York': 1300, 'Portland': 900,
'Austin': 450, 'Boston': None}
cities = pd.Series(d)
cities
Out[4]:
Austin            450
Boston            NaN
Chicago          1000
New York         1300
Portland          900
San Francisco    1100
dtype: float64
```

Series

You can use the index to select specific items from the Series ...

```
cities['Chicago']
Out[5]:
1000.0
```

Series

```
cities[['Chicago', 'Portland', 'San Francisco']]
Out[6]:
Chicago          1000
Portland          900
San Francisco    1100
dtype: float64
```

Series

You can use **boolean indexing** for selection.

```
cities[cities < 1000]
Out[7]:
Austin       450
Portland     900
dtype: float64
```

That last one might be a little strange, so let's make it more clear - `cities < 1000` returns a Series of `True/False` values, which we then pass to our Series cities, returning the corresponding `True` items.

```
less_than_1000 = cities < 1000
print less_than_1000
print '\n'
print cities[less_than_1000]
Austin            True
Boston           False
Chicago          False
New York         False
Portland          True
San Francisco    False
dtype: bool


Austin      450
Portland    900
dtype: float64
```

You can also change the values in a Series on the fly.

```
# changing based on the index

print 'Old value:', cities['Chicago']

cities['Chicago'] = 1400
print 'New value:', cities['Chicago']

Old value: 1000.0
New value: 1400.0
```

Changing values using boolean logic

```
print cities[cities < 1000]
print '\n'
cities[cities < 1000] = 750

print cities[cities < 1000]
Austin      450
Portland    900
dtype: float64


Austin      750
Portland    750
dtype: float64
```

Working with Series

What if you aren't sure whether an item is in the Series? You can check using idiomatic Python.

```
print 'Seattle' in cities
print 'San Francisco' in cities
False
True
```

Mathematical operations can be done using scalars and functions.

```
# divide city values by 3
cities / 3
Out[12]:
Austin             250.000000
Boston                    NaN
Chicago            466.666667
New York           433.333333
Portland           250.000000
San Francisco      366.666667
dtype: float64
```

```
# square city values
np.square(cities)
Out[13]:
Austin              562500
Boston                 NaN
Chicago            1960000
New York           1690000
Portland            562500
San Francisco      1210000
dtype: float64
```

You can add two Series together, which returns a union of the two Series with the addition occurring on the shared index values. Values on either Series that did not have a shared index will produce a NULL/NaN (not a number).

```
print cities[['Chicago', 'New York', 'Portland']]
print'\n'
print cities[['Austin', 'New York']]
print'\n'
print cities[['Chicago', 'New York', 'Portland']] + citie
```

```
Chicago      1400
New York     1300
Portland      750
dtype: float64
```

```
Austin        750
New York     1300
dtype: float64
```

```
Austin        NaN
Chicago       NaN
New York     2600
Portland      NaN
dtype: float64
```

Working with Series

NULL Checking

- Notice that because Austin, Chicago, and Portland were not found in both Series, they were returned with NULL/NaN values.
- NULL checking can be performed with `isnull()` and `notnull()`.

Return a boolean series indicating which values aren't NULL

```
cities.notnull()

Austin              True
Boston             False
Chicago             True
New York            True
Portland            True
San Francisco       True
dtype: bool
```

Using boolean logic to grab the NULL cities

```
print cities.isnull()
print '\n'
print cities[cities.isnull()]
Austin           False
Boston            True
Chicago          False
New York         False
Portland         False
San Francisco    False
dtype: bool

Boston   NaN
dtype: float64
```

Special Arrays

Functions are available to construct a number of useful, frequently encountered arrays.

ones

ones generates an array of 1s and is generally called with one argument, a tuple, containing the size of each dimension. ones takes an optional second argument (dtype) to specify the data type. If omitted, the data type is float.

```
>>> M, N = 5, 5
>>> x = ones((M,N)) # M by N array of 1s
>>> x = ones((M,M,N)) # 3D array
>>> x = ones((M,N), dtype=int32) # 32bit integers
```

zeros

zeros produces an array of 0s in the same way ones produces an array of 1s, and commonly used to initialize an array to hold values generated by another procedure. zeros takes an optional second argument (dtype) to specify the data type. If omitted, the data type is float.

```
>>> x = zeros((M,N)) # M by N array of 0s
>>> x = zeros((M,M,N)) # 3D array of 0s
>>> x = zeros((M,N),dtype=int64) # 64 bit integers
```

ones

`ones_like` creates an array with the same shape and data type as the input. Calling `ones_like(x)` is equivalent to calling `ones(x.shape,x.dtype)`. zeros_like creates an array with the same size and shape as the input. Calling zeros_like(x) is equivalent to calling zeros(x.shape,x.dtype).

empty

empty produces an empty (uninitialized) array to hold values generated by another procedure. empty takes an optional second argument (dtype) which specifies the data type. If omitted, the data type is float.

```
>>> x = empty((M,N)) # M by N empty array
>>> x = empty((N,N,N,N)) # 4D empty array
>>> x = empty((M,N),dtype=float32) # 32bit
```

floats (single precision)

- Using empty is slightly faster than calling zeros since it does not assign 0 to all elements of the array the empty array created will be populated with (essential random) non-zero values.
- empty_like creates an array with the same size and shape as the input.
- Calling empty_like(x) is equivalent to calling empty(x.shape,x.dtype).

eye, identity

eye generates an identity array an array with ones on the diagonal, zeros everywhere else. Normally, an identity array is square and so usually only 1 input is required. More complex zero-padded arrays containing an identity matrix can be produced using optional inputs.

```
>>> In = eye(N)
```

identity is a virtually identical function with similar use, In = identity(N).

The Normal Distribution - `normal`

The main commands

- `normal()` generates a set of random numbers from a standard Normal distribution.
- `normal(mu, sigma)` generates draws from a Normal distribution with mean μ and standard deviation σ.
- `normal(mu, sigma, (10,10))` generates a 10 by 10 array of draws from a Normal with mean μ and standard deviation σ.
- `normal(mu, sigma)` is equivalent to `mu + sigma * standard_normal()`.

The Poisson Distribution - `poisson`

- `poisson()` generates a set of random numbers from a Poisson distribution with $\lambda = 1$.
- `poisson(lambda)` generates a draw from a Poisson distribution with expectation λ.
- poisson(lambda, (10,10)) generates a 10 by 10 array of draws from a Poisson distribution with expectation λ.

standard_t

`standard_t(nu)` generates a set of random numbers from a Students t with shape parameter ν.
`standard_t(nu, (10,10))` generates a 10 by 10 array of draws from a Students t with shape parameter ν.

uniform

`uniform()` generates a uniform random variable on (0, 1).
`uniform(low, high)` generates a uniform on (l , h).
`uniform(low, high, (10,10))` generates a 10 by 10 array of uniforms on (l , h).

Continuous Random Variables

SciPy contains a large number of functions for working with continuous random variables. Each function resides in its own class (e.g. norm for Normal or gamma for Gamma), and classes expose methods for random number generation, computing the PDF, CDF and inverse CDF, fitting parameters using MLE, and computing various moments. The methods are listed below, where dist is a generic placeholder for the distribution name in SciPy.

- `dist.rvs`
  Pseudo-randomnumbergeneration. Generically, rvs is called using dist.rvs(*args, loc=0, scale=1, size=size) where size is an n-element tuple containing the size of the array to be generated.
- `dist.pdf`
  Probability density function evaluation for an array of data (element-by-element). Generically, pdf is called using `dist.pdf(x, *args, loc=0, scale=1)` where x is an array that contains the values to use when evaluating PDF.

- `dist.cdf`
  Cumulative distribution function evaluation for an array of data (element-by-element). Generically, cdf is called using `dist.cdf(x, *args, loc=0, scale=1)` where x is an array that contains the values to use when evaluating CDF.
- `dist.ppf`
  Inverse CDF evaluation (also known as percent point function) for an array of values between 0 and 1. Generically, ppf is called using `dist.ppf(p, *args, loc=0, scale=1)` where p is an array with all elements between 0 and 1 that contains the values to use when evaluating inverse CDF.

- `dist.fit`
  Estimate shape, location, and scale parameters from data by maximum likelihood using an array of data.
  Generically, fit is called using `dist.fit(data, *args, floc=0, fscale=1)` where data is a data array used to estimate the parameters.
  floc forces the location to a particular value (e.g. floc=0).
  `fscale` similarly forces the scale to a particular value (e.g. `fscale=1`) .
  It is necessary to use floc and/or fscale when computing MLEs if the distribution does not have a location and/or scale.
  For example, the gamma distribution is defined using 2 parameters, often referred to as shape and scale.
  In order to useMLto estimate parameters from a gamma, floc=0 must be used.

- `dist.median`
  Returns the median of the distribution. Generically, median is called using dist.median(*args, loc=0, scale=1).
- `dist.mean`
  Returns the mean of the distribution. Generically, mean is called using dist.mean(*args, loc=0, scale=1).
- `dist.moment`
  nth non-centralmomentevaluation of the distribution. Generically, moment is called using dist.moment(r, *args, loc=0, scale=1) where r is the order of the moment to compute.
- `dist.var`
  Returns the variance of the distribution. Generically, var is called using `dist.var(*args, loc=0, scale=1)`.
- `dist.std`
  Returns the standard deviation of the distribution. Generically, std is called using dist.std(*args, loc=0, scale=1).

Example

The gamma distribution is used as an example.
The gamma distribution takes 1 shape parameter a (a is the only element of *args), which is set to 2 in all examples.

```
>>> import scipy.stats as stats
>>> gamma = stats.gamma

>>> gamma.mean(2), gamma.median(2)
>>> gamma.std(2), gamma.var(2)
(2.0, 1.678346900166608, 1.4142135623730951, 2.0)

>>> gamma.moment(2,2) gamma.
moment(1,2)**2 # Variance
```

```
>>> gamma.cdf(5, 2), gamma.pdf(5, 2)
(0.9595723180054873, 0.03368973499542734)

>>> gamma.ppf(.9595723180054873, 2)
5.000000000000018

>>> log(gamma.pdf(5, 2)) gamma.
logpdf(5, 2)
0.0
```

```
>>> gamma.rvs(2, size=(2,2))
array([[ 1.83072394, 2.61422551],
       [ 1.31966169, 2.34600179]])

>>> gamma.fit(gamma.rvs(2, size=(1000)), floc = 0)
    # a, 0, shape
(2.209958533078413, 0, 0.8918726284546031 3)
```