File Edit View Insert Cell Kernel Help | Python 2

Introduction to Blaze

In this tutorial we'll learn how to use Blaze to discover, migrate, and query data living in other databases. Generally this tutorial will have the following format

1. odo - Move data to database
2. `blaze` - Query data in database

Install

This tutorial uses many different libraries that are all available with the Anaconda Distribution. Once you have Anaconda install, please run these commands from a terminal:

```
$ conda install -y blaze
$ conda install -y bokeh
$ conda install -y odo
$ conda install -y into # required because of recent renaming of in
to that collides inside blaze
```

The last command is there because of an issue with `odo` which was recently renamed (formerly `into`).

Goal: Accessible, Interactive, Analytic Queries

Chrome File Edit View History Bookmarks Window People Help Sun 1:35 PM

Ecosystem — | Welcome to | Bokeh Plot | Bokeh Plot | Bokeh Plot | Andy

blaze.pydata.org/en/latest/

Blaze

Overview

Install

Quickstart

Basic Queries

Split-Apply-Combine – Grouping

Pandas to Blaze

URI strings

Tips for working with CSV files

Interacting with SQL Databases

Out of Core Processing

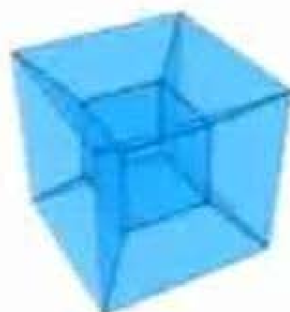
Server

Datashape

Read the Docs v: latest

Docs » Ecosystem

Edit on GitHub



Blaze

Blaze translates a subset of modified NumPy and Pandas-like syntax to databases and other computing systems. Blaze allows Python users a familiar interface to query data living in other data storage systems.

Ecosystem

Several projects have come out of Blaze development other than the Blaze project itself.

Chrome File Edit View History Bookmarks Window People Help Sun 1:35 PM

Ecosystem — | Welcome to B | Bokeh Plot S | Bokeh Plot S | Bokeh Plot S | Andy

bokeh.pydata.org/en/latest/

Quickstart
Installation
Bokeh Tutorial
Gallery
User Guide
Developer Guide
Contributing
Frequently Asked Questions
Release Notes and Roadmap
Reference Guide
Source

Search

Welcome to Bokeh

Bokeh is a Python interactive visualization library that targets modern web browsers for presentation. Its goal is to provide elegant, concise construction of novel graphics in the style of D3.js, but also deliver this capability with high-performance interactivity over very large or streaming datasets. Bokeh can help anyone who would like to quickly and easily create interactive plots, dashboards, and data applications.

For more information about the goals and direction of the project, please see the Technical Vision.

To get started quickly, follow the Quickstart.

To see examples of how you might use Bokeh with your own data, check out the Gallery.

For questions and technical assistance, come join the Bokeh mailing list.

Visit the GitHub source repository.

Be sure to follow us on Twitter @bokehplots, as well as on Vine, and Youtube!

Contents

- Quickstart
  - Introduction
  - Downloading
  - Getting Started
  - More examples
  - Using Bokeh Plot Server
  - Using Bokeh with IPython Notebooks
  - Sample Data
  - What's next?

texas

```
from collections import OrderedDict

from bokeh.sampledata import us_counties, unemployment
from bokeh.plotting import *
from bokeh.models import HoverTool

county_xs=[
    us_counties.data[code]['lons'] for code in us_counties.data
    if us_counties.data[code]['state'] == 'tx'
]
county_ys=[
    us_counties.data[code]['lats'] for code in us_counties.data
    if us_counties.data[code]['state'] == 'tx'
]

colors = ["#F1EEF6", "#D4B9DA", "#C994C7", "#DF65B0", "#DD1C77", "#980043"]

county_colors = []
for county_id in us_counties.data:
    if us_counties.data[county_id]['state'] != 'tx':
        continue
    try:
        rate = unemployment.data[county_id]
        idx = min(int(rate/2), 5)
        county_colors.append(colors[idx])
    except KeyError:
        county_colors.append("black")

output_file("texas.html", title="texas.py example")

TOOLS="pan,wheel_zoom,box_zoom,reset,hover,save"

p = figure(title="Texas Unemployment 2009", tools=TOOLS)
```

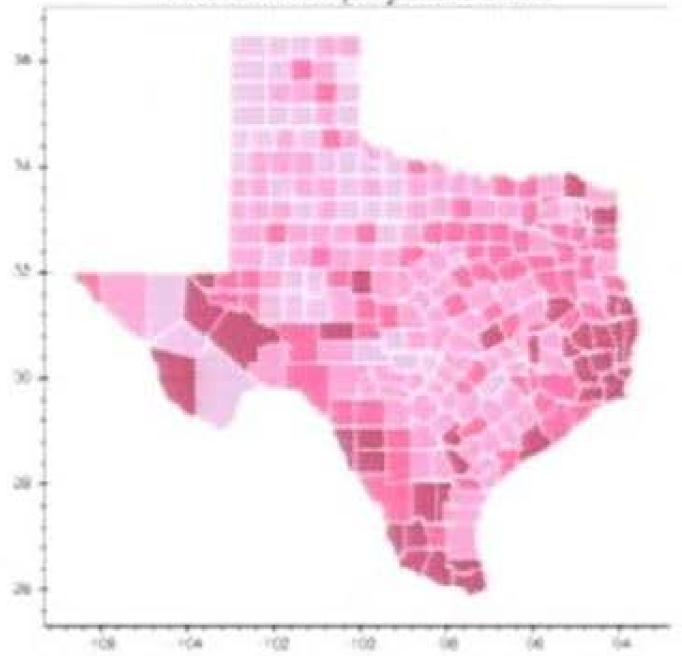
Chrome File Edit View History Bookmarks Window People Help Sun 1:36 PM

Ecosystem — texas — Boke Bokeh Plot S Bokeh Plot S Bokeh Plot S Andy

bokeh.pydata.org/en/latest/docs/gallery/texas.html

texas

< choropleth | back to Gallery | scatter >




```
from collections import OrderedDict

from bokeh.sampledata import us_counties, unemployment
from bokeh.plotting import *
from bokeh.models import HoverTool

county_xs=[
    us_counties.data[code]['lons'] for code in us_counties.data
    if us_counties.data[code]['state'] == 'tx'
```

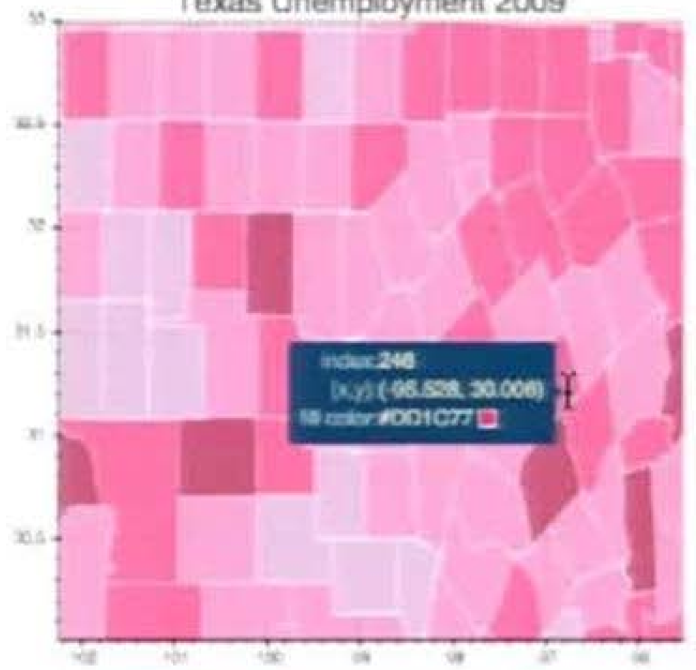
Chrome File Edit View History Bookmarks Window People Help Sun 1:36 PM

Ecosystem — | texas — Boke | Bokeh Plot | Bokeh Plot | Bokeh Plot | Andy

bokeh.pydata.org/en/latest/docs/gallery/texas.html

texas

« choropleth | back to Gallery | scatter »




```
from collections import OrderedDict

from bokeh.sampledata import us_counties, unemployment
from bokeh.plotting import *
from bokeh.models import HoverTool

county_xs=[
    us_counties.data[code]['lons'] for code in us_counties.data
    if us_counties.data[code]['state'] == 'tx'
```

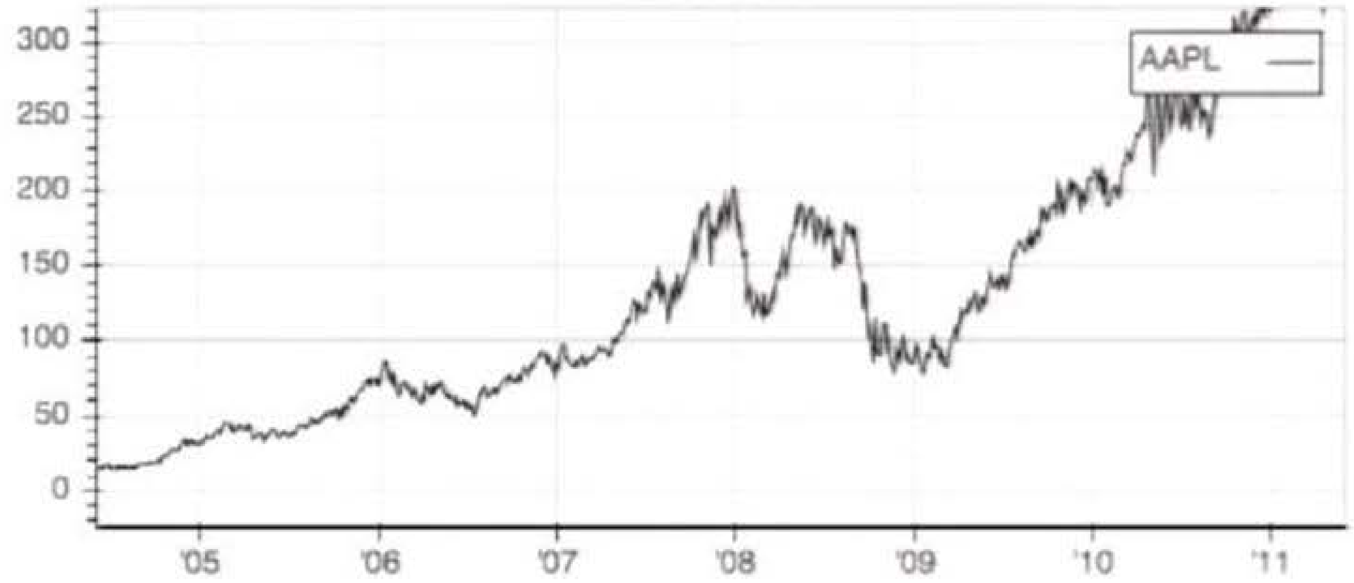
Chrome File Edit View History Bookmarks Window People Help Sun 1:36 PM

Ecosystem — | Gallery — Bo | Bokeh Plot S | Bokeh Plot S | Bokeh Plot S | Andy

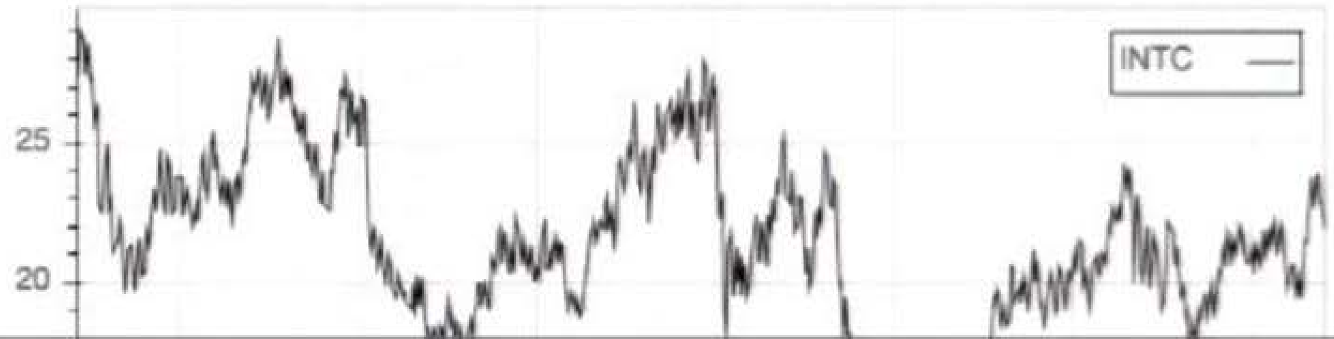
162.243.42.178:9000/v5.0/bokeh/timeseries/

link to this

AAPL



INTC



Chrome File Edit View History Bookmarks Window People Help Sun 1:37 PM

Ecosystem — | Gallery — Bo | Bokeh Plot S | Bokeh Plot S | Bokeh Plot S | Andy

162.243.42.178:9000/v5.0/bokeh/ocean/

xy slice

50

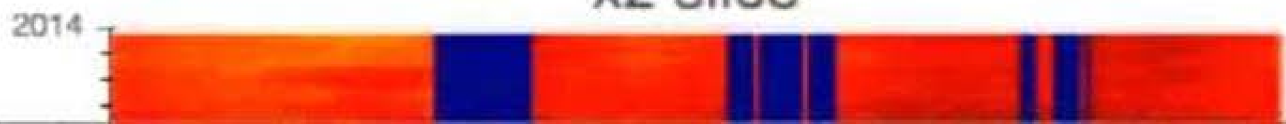
0

-50

-150 -100 -50 0 50 100 150

xz slice

2014



Chrome File Edit View History Bookmarks Window People Help Sun 1:37 PM

Ecosystem — | Gallery — Bo | Bokeh Plot Se | Bokeh Plot Se | Bokeh Plot Se | Andy

127.0.0.1:5006/bokeh/crossfilter/#

Bokeh Plot Server —

hp (Continuous)

count	392
mean	104.47
std	38.49
min	46.00
max	230.00

weight (Continuous)

count	392
mean	2977.58
std	849.40
min	1613.00
max	5140.00

accel (Continuous)

count	392
mean	15.54
std	2.76
min	8.00
max	24.80

yr (Continuous)

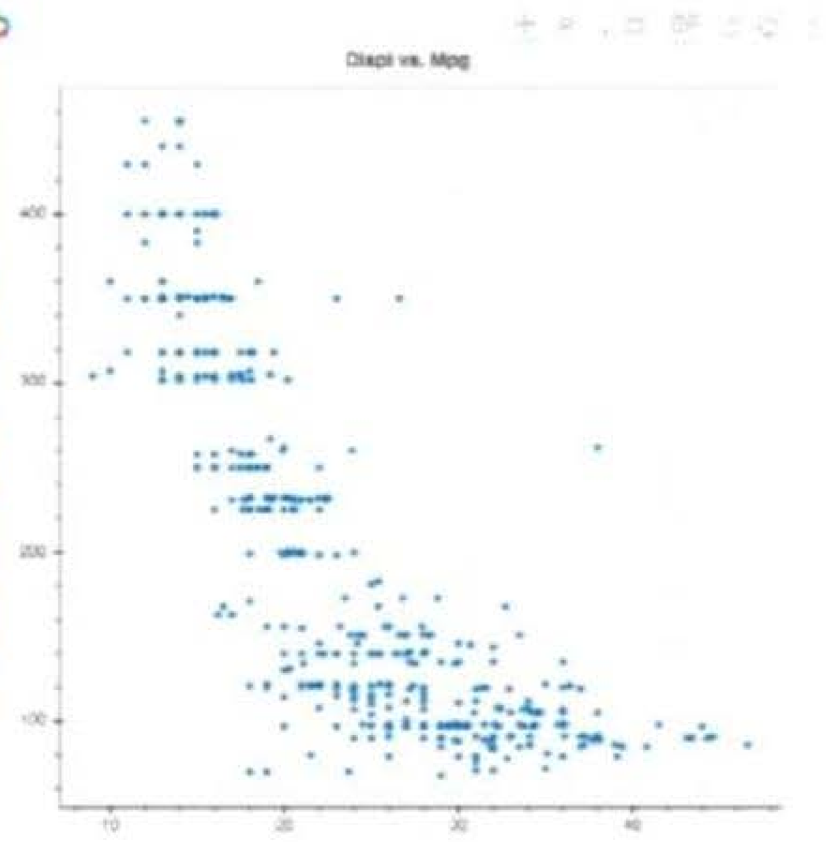
Filter

Facet X

Facet Y

Facet Tab

PlotType	x	y	agg
scatter	mpg	displ	sum



Introduction to Blaze

In this tutorial we'll learn how to use Blaze to discover, migrate, and query data living in other databases. Generally this tutorial will have the following format

1. `odo` - Move data to database
2. `blaze` - Query data in database

Install

This tutorial uses many different libraries that are all available with the Anaconda Distribution. Once you have Anaconda install, please run these commands from a terminal:

```
$ conda install -y blaze
$ conda install -y bokeh
$ conda install -y odo
$ conda install -y into # required because of recent renaming of in
to that collides inside blaze
```

The last command is there because of an issue with `odo` which was recently renamed (formerly `into`).

Goal: Accessible, Interactive, Analytic Queries

NumPy and Pandas provide accessible, interactive, analytic queries; this is valuable.

File Edit View Insert Cell Kernel Help | Python 2

Goal: Accessible, Interactive, Analytic Queries

NumPy and Pandas provide accessible, interactive, analytic queries; this is valuable.

In [1]:
```
import pandas as pd
df = pd.read_csv('iris.csv')
df.head()
```

Out[1]:

	Unnamed: 0	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
0	1	5.1	3.5	1.4	0.2	setosa
1	2	4.9	3.0	1.4	0.2	setosa
2	3	4.7	3.2	1.3	0.2	setosa
3	4	4.6	3.1	1.5	0.2	setosa
4	5	5.0	3.6	1.4	0.2	setosa

In [2]:
```
df.groupby(df.Species).PetalLength.mean()  # Average petal length per spec
```

Out[2]:
```
Species
setosa        1.462
versicolor    4.260
virginica     5.552
Name: PetalLength, dtype: float64
```

But as data grows and systems become more complex, moving data and querying data become

Mouseposé Edit Toggle Options Window Help 68% Sun 1:41 PM

Ecosystem — Blaze 0.7.3 | Gallery — Bokeh 0.8.1 do | Andy

bokeh.pydata.org/en/latest/docs/gallery.html

Gallery

Server Examples

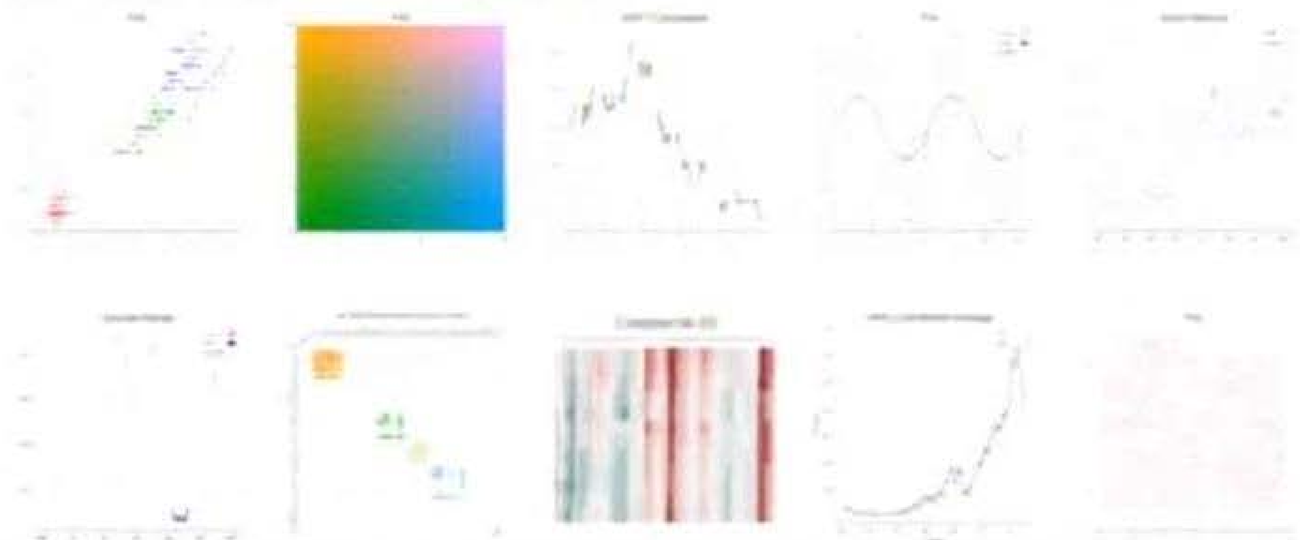
The examples linked below all show off usage of the Bokeh server. The Bokeh server provides a place where interesting things can happen— data can be updated to in turn update the plot, and UI and selection events can be processed to trigger more visual updates. It is also possible to downsample large data sets or stream continuously updating data from the Bokeh server.

- *Sliders Example.*
- *Crossfilter Example.*
- *Stocks Example.*
- *Animated Line Example.*
- *Animated Glyph Example.*

Static Examples

All of the static examples below are located in the examples subdirectory of your Bokeh checkout. By "static", we simply mean that no use is made of the Bokeh server. Static plots can still have many interactive tools and features, including linked panning and brushing, and hover inspectors.

Click on an image below to see its code and interact with the live plot.



Chrome File Edit View History Bookmarks Window People Help Sun 1:42 PM

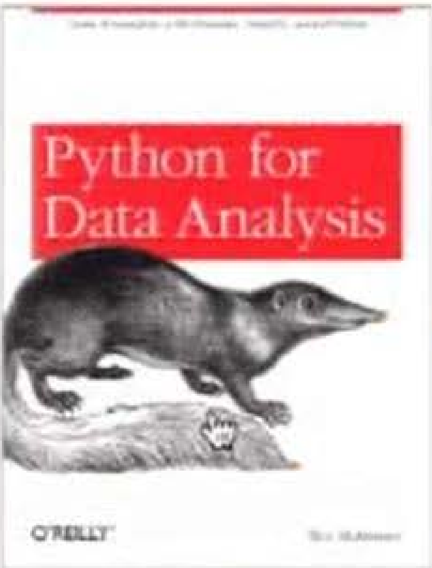
Ecosystem — Blaze 0.7.3 | Gallery — Bokeh 0.8.1 doc | Python for Data Analysis

shop.oreilly.com/product/0636920023784.do

Video Training | Browse Subjects | New | Upcoming | Early Release | Bestselling | Ebooks

1-800-889-8969 / 707

Start any video training course for FREE. More than 500 to choose from.



Larger Cover

Python for Data Analysis

Data Wrangling with Pandas, NumPy, and IPython

By Wes McKinney

Publisher: O'Reilly Media
Final Release Date: October 2012
Pages: 466

4.2

Read 24 Reviews | Write a Review

Python for Data Analysis is concerned with the nuts and bolts of manipulating, processing, cleaning, and crunching data in Python. It is also a practical, modern introduction to scientific computing in Python, tailored for data-intensive applications. This is a book about the parts of the Python language and libraries you'll...

Full description

Buy 2 Get 1 FREE

Buying O

Immediate A

Ebook: $33.99
Formats: DAIS
PDF

Print & Ebook:

Print: $39.99

Safari Books

Table of Contents | Product Details | About the Author | Colophon

Essential Li

Chapter 1 : Preliminaries

What Is This Book About?

Download E

Download E

shop.oreilly.com/product/0636920023784.do?sortby=bestSel

File Edit View Insert Cell Kernel Help | Python 2

Goal: Accessible, Interactive, Analytic Queries

NumPy and Pandas provide accessible, interactive, analytic queries; this is valuable.

In [1]:
```
import pandas as pd
df = pd.read_csv('iris.csv')
df.head()
```

Out[1]:

	Unnamed: 0	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
0	1	5.1	3.5	1.4	0.2	setosa
1	2	4.9	3.0	1.4	0.2	setosa
2	3	4.7	3.2	1.3	0.2	setosa
3	4	4.6	3.1	1.5	0.2	setosa
4	5	5.0	3.6	1.4	0.2	setosa

In [2]:
```
df.groupby(df.Species).PetalLength.mean()  # Average petal length per speci
```

Out[2]:
```
Species
setosa        1.462
versicolor    4.260
virginica     5.552
Name: PetalLength, dtype: float64
```

But as data grows and systems become more complex, moving data and querying data become

File Edit View Insert Cell Kernel Help | Python 2

Goal: Accessible, Interactive, Analytic Queries

NumPy and Pandas provide accessible, interactive, analytic queries; this is valuable.

In [*]:
```
import pandas as pd
df = pd.read_csv('iris.csv')
df.head()
```

In [2]:
```
df.groupby(df.Species).PetalLength.mean()  # Average petal length per speci
```

Out[2]:
```
Species
setosa        1.462
versicolor    4.260
virginica     5.552
Name: PetalLength, dtype: float64
```

But as data grows and systems [illegible] more complex, [illegible]ving data and querying data become more difficult. Python alread[illegible] exc[illegible] to[illegible] hat fits in memory, but we want to hook up to data that is inconvenient [illegible]

From now on, we're going to assume one of the following:

1. You have an inconvenient amount of data
2. That data should live someplace other than your computer

File Edit View Insert Cell Kernel Help | Python 2

Goal: Accessible, Interactive, Analytic Queries

NumPy and Pandas provide accessible, interactive, analytic queries; this is valuable.

In [1]:
```
import pandas as pd
df = pd.read_csv('iris.csv')
df.head()
```

Out[1]:

	Unnamed: 0	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
0	1	5.1	3.5	1.4	0.2	setosa
1	2	4.9	3.0	1.4	0.2	setosa
2	3	4.7	3.2	1.3	0.2	setosa
3	4	4.6	3.1	1.5	0.2	setosa
4	5	5.0	3.6	1.4	0.2	setosa

In [2]:
```
df.groupby(df.Species).PetalLength.mean()  # Average petal length per speci
```

Out[2]:
```
Species
setosa        1.462
versicolor    4.260
virginica     5.552
Name: PetalLength, dtype: float64
```

But as data grows and systems become more complex, moving data and querying data become

But as data grows and systems become more complex, moving data and querying data become more difficult. Python already has excellent tools for data that fits in memory, but we want to hook up to data that is inconvenient.

From now on, we're going to assume one of the following:

1. You have an inconvenient amount of data
2. That data should live someplace other than your computer

Databases and Python

When in-memory arrays/dataframes cease to be an option, we turn to databases. These live outside of the Python process and so might be less convenient. The open source Python ecosystem includes libraries to interact with these databases and with foreign data in general.

Examples:

- SQL - `sqlalchemy`
  - Hive/Cassandra - `pyhive`
  - Impala - `impyla`
  - RedShift - `redshift-sqlalchemy`
  - ...
- MongoDB - `pymongo`

Databases and Python

When in-memory arrays/dataframes cease to be an option, we turn to databases. These live outside of the Python process and so might be less convenient. The open source Python ecosystem includes libraries to interact with these databases and with foreign data in general.

Examples:

- SQL - `sqlalchemy`
    - Hive/Cassandra - `pyhive`
    - Impala - `impyla`
    - RedShift - `redshift-sqlalchemy`
    - ...
- MongoDB - `pymongo`
- HBase - `happybase`
- Spark - `pyspark`
- SSH - `paramiko`
- HDFS - `pywebhdfs`
- Amazon S3 - `boto`

Today we're going to use some of these indirectly with odo (was `into`) and Blaze. We'll try to point out these libraries as we automate them so that, if you'd like, you can use them independently.

````markdown
### Examples


Into moves data into a target from a source

```python
>>> odo(source, target)
```


The target and source can be either a Python object or a string URI.  The
following are all valid calls to `into`

```python
>>> odo('iris.csv', pd.DataFrame)  # Load CSV file into new DataFrame
>>> odo(my_df, 'iris.json')        # Write DataFrame into JSON file
>>> odo('iris.csv', 'iris.json')   # Migrate data from CSV to JSON
```


<hr/>
````

Excerise

Use `into` to load the `iris.csv` file into a Python `list`, a `np.ndarray`, and a `pd.DataFrame`

In [3]:
```python
from odo import odo
import numpy as np
import pandas as pd
```

Examples

Odo moves data into a target from a source

```
>>> odo(source, target)
```

The target and source can be either a Python object or a string URI. The following are all valid calls to `into`

```
>>> odo('iris.csv', pd.DataFrame)  # Load CSV file into new DataFra
me
>>> odo(my_df, 'iris.json')        # Write DataFrame into JSON file
>>> odo('iris.csv', 'iris.json')   # Migrate data from CSV to JSON
```

Excerise

Use `into` to load the `iris.csv` file into a Python `list`, a `np.ndarray`, and a `pd.DataFrame`

In [3]:
```
from odo import odo
import numpy as np
import pandas as pd
```

File Edit View Insert Cell Kernel Help | Python 2

Excerise

Use `into` to load the `iris.csv` file into a Python `list`, a `np.ndarray`, and a `pd.DataFrame`

In [*]:
```
from odo import odo
import numpy as np
import pandas as pd
```

URI Strings

Odo refers to foreign data either with a Python object like a `sqlalchemy.Table` object for a SQL table, or with a string URI, like `postgresql://hostname::tablename`.

URI's often take on the following form

```
protocol://path-to-resource::path-within-resource
```

Where `path-to-resource` might point to a file, a database hostname, etc. while `path-within-resource` might refer to a datapath or table name. Note the two main separators

- `://` separates the protocol on the left (`sqlite`, `mongodb`, `ssh`, `hdfs`, `hive`, ...)
- `::` separates the path within the database on the right (e.g. tablename)

[odo docs on uri strings]()

File Edit View Insert Cell Kernel Help Python 2

to into

```
>>> odo('iris.csv', pd.DataFrame)  # Load CSV file into new DataFra
me
>>> odo(my_df, 'iris.json')        # Write DataFrame into JSON file
>>> odo('iris.csv', 'iris.json')   # Migrate data from CSV to JSON
```

Excerise

Use `into` to load the `iris.csv` file into a Python `list`, a `np.ndarray`, and a `pd.DataFrame`

In [3]:
```
from odo import odo
import numpy as np
import pandas as pd
```

BokehJS successfully loaded.

In [ ]:

URI Strings

Odo refers to foreign data either with a Python object like a `sqlalchemy.Table` object for a SQL table, or with a string URI, like `postgresql://hostname::tablename`.

File Edit View Insert Cell Kernel Help Python 2

0	1	5.1	3.5	1.4	0.2	setosa
1	2	4.9	3.0	1.4	0.2	setosa
2	3	4.7	3.2	1.3	0.2	setosa
3	4	4.6	3.1	1.5	0.2	setosa
4	5	5.0	3.6	1.4	0.2	setosa
5	6	5.4	3.9	1.7	0.4	setosa
6	7	4.6	3.4	1.4	0.3	setosa
7	8	[illegible]	3.4	1.5	0.2	setosa
8	9	4.4	2.9	1.4	0.2	setosa
9	10	4.9	3.1	1.5	0.1	setosa
10	11	5.4	3.7	1.5	0.2	setosa
11	12	4.8	3.4	1.6	0.2	setosa
12	13	4.8	3.0	1.4	0.1	setosa
13	14	4.3	3.0	1.1	0.1	setosa
14	15	5.8	4.0	1.2	0.2	setosa
15	16	5.7	4.4	1.5	0.4	setosa
16	17	5.4	3.9	1.3	0.4	setosa
17	18	5.1	3.5	1.4	0.3	setosa
18	19	5.7	3.8	1.7	0.3	setosa

click to scroll output; double click to hide

File Edit View Insert Cell Kernel Help Python 2

URI Strings

Odo refers to foreign data either with a Python object like a `sqlalchemy.Table` object for a SQL table, or with a string URI, like `postgresql://hostname::tablename`.

URI's often take on the following form

```
protocol://path-to-resource::path-within-resource
```

Where `path-to-resource` might point to a file, a database hostname, etc. while `path-within-resource` might refer to a datapath or table name. Note the two main separators

- `://` separates the protocol on the left (`sqlite`, `mongodb`, `ssh`, `hdfs`, `hive`, ...)
- `::` separates the path within the database on the right (e.g. tablename)

odo docs on uri strings

Examples

Here are some example URIs

```
myfile.json
myfiles.*.csv'
postgresql://hostname::tablename
mongodb://hostname/db::collection
```

File Edit View Insert Cell Kernel Help Python 2

[odo docs on uri strings]

Examples

Here are some example URIs

```
myfile.json
myfiles.*.csv'
postgresql://hostname::tablename
mongodb://hostname/db::collection
ssh://user@host:/path/to/myfile.csv
hdfs://user@host:/path/to/*.csv
```

<hr />

Exercise

Migrate your CSV file into a table named `iris` in a new SQLite database at `sqlite:///my.db`. Remember to use the `::` separator and to separate your database name from your table name.

[odo docs on SQL]

```
In [4]: odo("iris.csv", "sqlite:///my.db::iris")

Out[4]: Table('iris', MetaData(bind=Engine(sqlite:///my.db)), Column('Unnamed: 0',
 BigInteger(), table=<iris>, nullable=False), Column('SepalLength', Float(
```

File Edit View Insert Cell Kernel Help Python 2

Exercise

Migrate your CSV file into a table named `iris` in a new SQLite database at `sqlite:///my.db`. Remember to use the `::` separator and to separate your database name from your table name.

[odo docs on SQL]

```
In [5]: odo("iris.csv", "sqlite:///my.db::iris")
```

```
Out[5]: Table('iris', MetaData(bind=Engine(sqlite:///my.db)), Column('Unnamed: 0',
 BIGINT(), table=<iris>, nullable=False), Column('SepalLength', FLOAT(), t
able=<iris>), Column('SepalWidth', FLOAT(), table=<iris>), Column('PetalLe
ngth', FLOAT(), table=<iris>), Column('PetalWidth', FLOAT(), table=<iris>)
, Column('Species', TEXT(), table=<iris>), schema=None)
```

What kind of object did you get receive as output? Call `type` on your result.

```
In [5]: type(_)
```

```
Out[5]: sqlalchemy.sql.schema.Table
```

How it works

Odo is a network of fast pairwise conversions between pairs of formats. We when we migrate

File Edit View Insert Cell Kernel Help Python 2

```
In [5]: odo("iris.csv", "sqlite:///my.db::iris")

Out[5]: Table('iris', MetaData(bind=Engine(sqlite:///my.db)), Column('Unnamed: 0',
         BIGINT(), table=<iris>, nullable=False), Column('SepalLength', FLOAT(), t
        able=<iris>), Column('SepalWidth', FLOAT(), table=<iris>), Column('PetalLe
        ngth', FLOAT(), table=<iris>), Column('PetalWidth', FLOAT(), table=<iris>)
        , Column('Species', TEXT(), table=<iris>), schema=None)
```

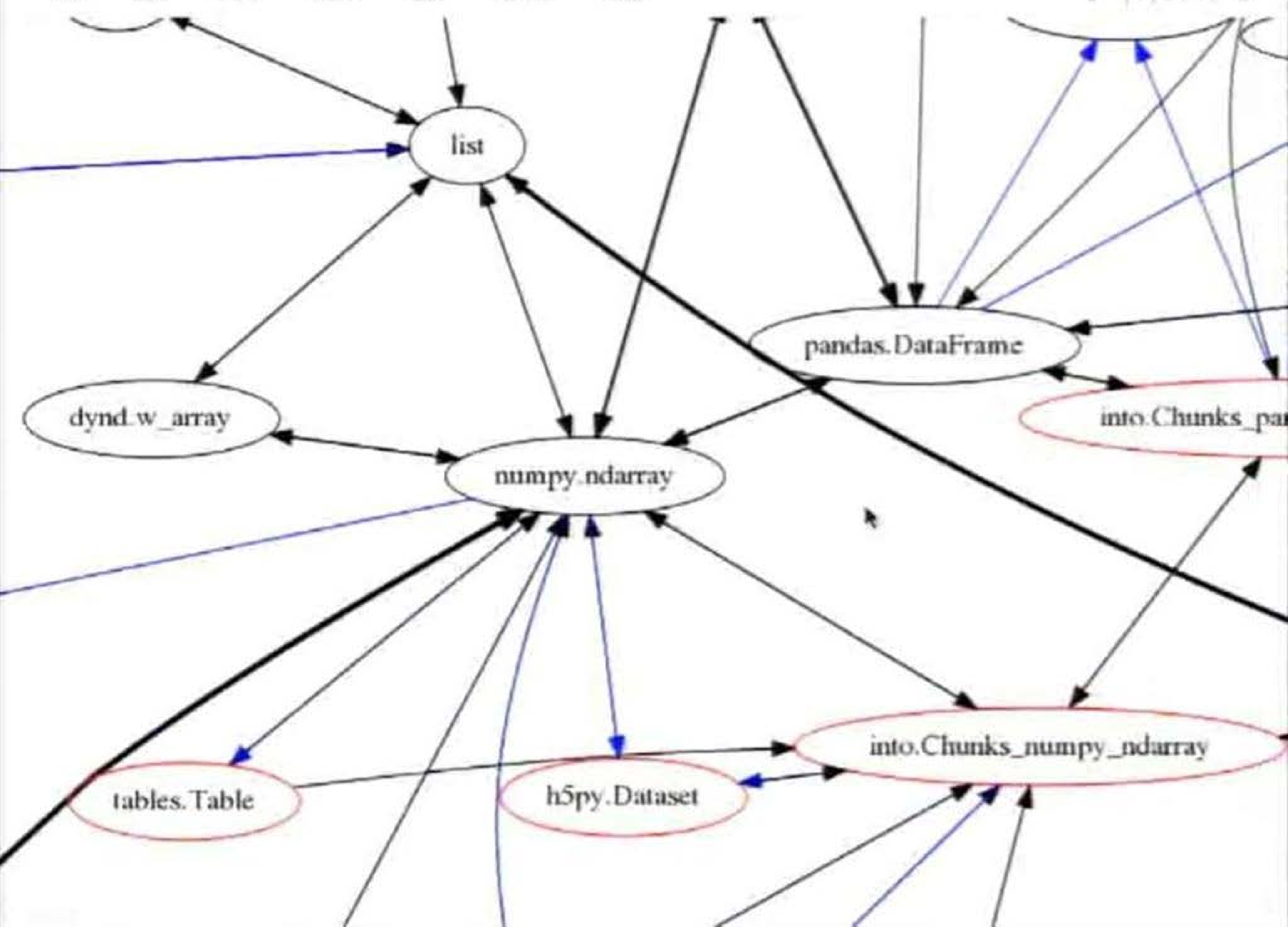
What kind of object did you get receive as output? Call `type` on your result.

```
In [6]: type(r)

Out[6]: sqlalchemy.sql.schema.Table
```

How it works

Odo is a network of fast pairwise conversions between pairs of formats. We when we migrate between two formats we traverse a path of pairwise conversions.

We visualize that network below:



File
Edit
View
Insert
Cell
Kernel
Help
Python 2
list
pandas.DataFrame
dynd.w_array
numpy.ndarray
tables.Table
h5py.Dataset
into.Chunks_numpy_ndarray

formats. Red nodes support larger-than-memory data.

A single call to into may traverse several intermediate formats calling on several conversion functions. For example, we when migrate a CSV file to a Mongo database we might take the following route:

- Load in to a `DataFrame` (`pandas.read_csv`)
- Convert to `np.recarray` (`DataFrame.to_records`)
- Then to a Python `Iterator` (`np.ndarray.tolist`)
- Finally to Mongo (`pymongo.Collection.insert`)

Alternatively we could write a special function that uses MongoDB's native CSV loader and shortcut this entire process with a direct edge `CSV -> Mongo`.

These functions are chosen because they are fast, often far faster than converting through a central serialization format.

This picture is actually from an older version of odo, when the graph was still small enough to visualize pleasantly. See odo docs for a more updated version.

Remote Data

We can interact with remote data in three locations

1. On Amazon's S3 (this will be quick)
2. On a remote machine via `ssh`
3. On the Hadoop File System (HDFS)

File Edit View Insert Cell Kernel Help | Python 2

Remote Data

We can interact with remote data in three locations

1. On Amazon's S3 (this will be quick)
2. On a remote machine via `ssh`
3. On the Hadoop File System (HDFS)

For most of this we'll wait until we've seen Blaze, briefly we'll use S3.

S3

For now, we quickly grab a file from Amazon's `S3`.

This example depends on boto to interact with S3.

```
conda install boto
```

odo docs on aws

In [6]:
```
odo('s3://nyqpug/tips.csv', pd.DataFrame)
```

```
ERROR:boto:Caught exception reading instance data
Traceback (most recent call last):
  File "/Users/aterrel/anaconda/lib/python2.7/site-packages/boto/utils.py"
, line 214, in retry_url
    r = opener.open(req)
  File "/Users/aterrel/anaconda/lib/python2.7/urllib2.py", line 431, in op
en
    response = self._open(req, data)
  File "/Users/aterrel/anaconda/lib/python2.7/urllib2.py", line 449, in _o
pen
```

File Edit View Insert Cell Kernel Help Python 2

```
Name: Species, dtype: object
```

Blaze example

In [9]:
```
import blaze as bz

d = bz.Data('iris.csv')
d.head(5)
```

Out[9]:

	Unnamed: 0	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
0	1	5.1	3.5	1.4	0.2	setosa
1	2	4.9	3.0	1.4	0.2	setosa
2	3	4.7	3.2	1.3	0.2	setosa
3	4	4.6	3.1	1.5	0.2	setosa
4	5	5.0	3.6	1.4	0.2	setosa

In [10]:
```
d.Species.distinct()
```

Out[10]:

	Species
0	setosa
1	versicolor
2	virginica

Foreign Data

Blaze does different things under-the-hood on different kinds of data

- CSV files: Pandas DataFrames (or iterators of DataFrames)
- SQL tables: SQLAlchemy.
- Mongo collections: PyMongo
- ...

SQL

We'll play with SQL a lot during this tutorial. Blaze translates your query to SQLAlchemy. SQLAlchemy then translates to the SQL dialect of your database, your database then executes that query intelligently.

- Blaze → SQLAlchemy → SQL → Database computation

This translation process lets analysts interact with a familiar interface while leveraging a potentially powerful database.

To keep things local we'll use SQLite, but this works with any database with a SQLAlchemy dialect. Examples in this section use the iris dataset. Exercises use the Lahman Baseball statistics database, year 2013.

If you have not downloaded this dataset you could do so here - https://github.com/jknecht/baseball-archive-sqlite/raw/master/lahman2013.sqlite.

File Edit View Insert Cell Kernel Help Python 2

```
Section0_intro.ipynb                  iris.csv
Section1-1_ipython.ipynb              iris.sqlite3
Section1-2_matplotlib.ipynb           lahman2013.sqlite
Section2_numpy_scipy_pandas.ipynb     my.db
Section3-1_blaze.ipynb                python.png
Section3-2_bokeh.ipynb
```

Examples

Lets dive into Blaze Syntax. For simple queries it looks and feels similar to Pandas

```
In [11]: db = bz.Data('sqlite:///my.db')
         db.iris

/Users/aterrel/anaconda/lib/python2.7/site-packages/IPython/core/formatter
s.py:239: FormatterWarning: Exception in text/html formatter: invalid lite
ral for long() with base 10: ''
  FormatterWarning,

Out[11]: <repr(<blaze.expr.expressions.Field at 0x1124036d8>) failed: ValueError: i
nvalid literal for long() with base 10: ''>
```

```
In [12]: db.iris.Species.distinct()
```

Out[12]:

	Species
0	Species
1	setosa
2	versicolor
3	virginica

File Edit View Insert Cell Kernel Help Python 2

```
Section0_intro.ipynb                     iris.csv
Section1-1_ipython.ipynb                 iris.sqlite3
Section1-2_matplotlib.ipynb              lahman2013.sqlite
Section2_numpy_scipy_pandas.ipynb my.db
Section3-1_blaze.ipynb                   python.png
Section3-2_bokeh.ipynb
```

Examples

Lets dive into Blaze Syntax. For simple queries it looks and feels similar to Pandas

```
In [11]: db = bz.Data('sqlite:///my.db')
         db.iris
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call last
)
/Users/aterrel/anaconda/lib/python2.7/site-packages/IPython/core/formatter
s.pyc in __call__(self, obj)
    693                         type_pprinters=self.type_printers,
    694                         deferred_pprinters=self.deferred_printers)
--> 695                 printer.pretty(obj)
    696                 printer.flush()
    697                 return stream.getvalue()

/Users/aterrel/anaconda/lib/python2.7/site-packages/IPython/lib/pretty.pyc
 in pretty(self, obj)
    399                             if callable(meth):
    400                                 return meth(obj, self, cycle)
--> 401                 return _default_pprint(obj, self, cycle)
    402             finally:
```

File Edit View Insert Cell Kernel Help Python 2

In [11]:
```
db = bz.Data('sqlite:///my.db')
#db.iris
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call last
)
/Users/aterrel/anaconda/lib/python2.7/site-packages/IPython/core/formatter
s.pyc in __call__(self, obj)
    693                             type_pprinters=self.type_printers,
    694                             deferred_pprinters=self.deferred_printers)
--> 695                 printer.pretty(obj)
    696                 printer.flush()
    697                 return stream.getvalue()

/Users/aterrel/anaconda/lib/python2.7/site-packages/IPython/lib/pretty.pyc
 in pretty(self, obj)
    399                                 if callable(meth):
    400                                     return meth(obj, self, cycle)
--> 401                 return _default_pprint(obj, self, cycle)
    402             finally:
    403                 self.end_group()
```

In [12]:
```
db.iris.Species.distinct()
```

Out[12]:

	Species
0	Species
1	setosa
2	versicolor

File Edit View Insert Cell Kernel Help | Python 2

In [12]:
```
db = bz.Data('sqlite:///my.db')
#db.iris
```

In [12]:
```
db.iris.Species.distinct()
```

Out[12]:

	Species
0	Species
1	setosa
2	versicolor
3	virginica

In [13]:
```
db.iris[db.iris.Species == 'versicolor'][['Species', 'SepalLength']]
```

Out[13]:

	Species	SepalLength
0	versicolor	7.0
1	versicolor	6.4
2	versicolor	6.9
3	versicolor	5.5
4	versicolor	6.5
5	versicolor	5.7
6	versicolor	6.3
7	versicolor	4.9

File Edit View Insert Cell Kernel Help Python 2

In [13]:
```
db = bz.Data('sqlite:///my.db')
#db.iris
db.iris.head()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last
)
/Users/aterrel/anaconda/lib/python2.7/site-packages/IPython/core/formatter
s.pyc in __call__(self, obj)
    693                             type_pprinters=self.type_printers,
    694                             deferred_pprinters=self.deferred_printers)
--> 695                 printer.pretty(obj)
    696                 printer.flush()
    697                 return stream.getvalue()

/Users/aterrel/anaconda/lib/python2.7/site-packages/IPython/lib/pretty.pyc
 in pretty(self, obj)
    399                                 if callable(meth):
    400                                     return meth(obj, self, cycle)
--> 401                 return _default_pprint(obj, self, cycle)
    402             finally:
    403                 self.end_group()
```

In [12]:
```
db.iris.Species.distinct()
```

Out[12]:

	Species
0	Species
1	setosa

File Edit View Insert Cell Kernel Help Python 2

3	virginica

In [16]:
```
db.iris[db.iris.Species == 'versicolor'][['Species', 'SepalLength']]
```

Out[16]:

	Species	SepalLength
0	versicolor	7.0
1	versicolor	6.4
2	versicolor	6.9
3	versicolor	5.5
4	versicolor	6.5
5	versicolor	5.7
6	versicolor	6.3
7	versicolor	4.9
8	versicolor	6.6
9	versicolor	5.2
10	versicolor	5.0

Work happens on the database

If we were using pandas we would read the table into pandas, then use pandas' fast in-memory algorithms for computation. Here we translate your query into SQL and then send that query to the

File Edit View Insert Cell Kernel Help | Python 2

Work happens on the database

If we were using pandas we would read the table into pandas, then use pandas' fast in-memory algorithms for computation. Here we translate your query into SQL and then send that query to the database to do the work.

- Pandas $\leftarrow_{\text{data}}$ SQL, then Pandas computes
- Blaze $\rightarrow_{\text{query}}$ SQL, then database computes

If we want to dive into the internal API we can inspect the query that Blaze transmits.

In [14]:
```
# Inspect SQL query
query = db.iris[db.iris.Species == 'versicolor'][['Species', 'SepalLength']]
print bz.compute(query)
```

```
SELECT iris."Species", iris."SepalLength"
FROM iris
WHERE iris."Species" = ?
```

In [15]:
```
query = bz.by(db.iris.Species, longest=db.iris.PetalLength.max(),
                               shortest=db.iris.PetalLength.min())
print bz.compute(query)
```

```
SELECT iris."Species", max(iris."PetalLength") AS longest, min(iris."Petal
Length") AS shortest
FROM iris GROUP BY iris."Species"
```

File Edit View Insert Cell Kernel Help Python 2

- Pandas $\leftarrow_{\text{data}}$ SQL, then Pandas computes
- Blaze $\rightarrow_{\text{query}}$ SQL, then database computes

If we want to dive into the internal API we can inspect the query that Blaze transmits.

In [17]:
```
# Inspect SQL query
query = db.iris[db.iris.Species == 'versicolor'][['Species', 'SepalLength']]
print bz.compute(query)
```

```
SELECT iris."Species", iris."SepalLength"
FROM iris
WHERE iris."Species" = ?
```

In [15]:
```
query = bz.by(db.iris.Species, longest=db.iris.PetalLength.max(),
                                shortest=db.iris.PetalLength.min())
print bz.compute(query)
```

```
SELECT iris."Species", max(iris."PetalLength") AS longest, min(iris."Petal
Length") AS shortest
FROM iris GROUP BY iris."Species"
```

Exercises

Now we load the Lahman baseball database and perform similar queries

In [16]:
```
# db = bz.Data('postgresql://postgres:postgres@ec2-54-159-160-163.compute-1
```

File Edit View Insert Cell Kernel Help | Python 2

In [17]:
```
# Inspect SQL query
query = db.iris[db.iris.Species == 'versicolor'][['Species', 'SepalLength']
print bz.compute(query)
```

```
SELECT iris."Species", iris."SepalLength"
FROM iris
WHERE iris."Species" = ?
```

In [18]:
```
query = bz.by(db.iris.Species, longest=db.iris.PetalLength.max(),
                               shortest=db.iris.PetalLength.min())
print bz.compute(query)
```

```
SELECT iris."Species", max(iris."PetalLength") AS longest, min(iris."Petal
Length") AS shortest
FROM iris GROUP BY iris."Species"
```

In [ ]:
```
odo(query,
```

Exercises

Now we load the Lahman baseball database and perform similar queries

In [16]:
```
# db = bz.Data('postgresql://postgres:postgres@ec2-54-159-160-163.compute-1
db = bz.Data('sqlite:///lahman2013.sqlite')
db.dshape
```

File Edit View Insert Cell Kernel Help Python 2

```
query = db.iris[db.iris.Species == 'versicolor'][['Species', 'SepalLength']]
print bz.compute(query)
```

```
SELECT iris."Species", iris."SepalLength"
FROM iris
WHERE iris."Species" = ?
```

In [18]:
```
query = bz.by(db.iris.Species, longest=db.iris.PetalLength.max(),
                               shortest=db.iris.PetalLength.min())
print bz.compute(query)
```

```
SELECT iris."Species", max(iris."PetalLength") AS longest, min(iris."Petal
Length") AS shortest
FROM iris GROUP BY iris."Species"
```

In [20]:
```
odo(query, list)
```

Out[20]:
```
[(u'Species', u'PetalLength', u'PetalLength'),
 (u'setosa', 1.9, 1.0),
 (u'versicolor', 5.1, 3.0),
 (u'virginica', 6.9, 4.5)]
```

Exercises

Now we load the Lahman baseball database and perform similar queries

In [16]:
```
# db = bz.Data('postgresql://postgres:postgres@ec2-54-159-160-163.compute-1
db = bz.Data('sqlite:///lahman2013.sqlite')
db.dshape
```

```
virginica     4.5
Name: PetalLength, dtype: float64
```

Store Results

By default Blaze only shows us the first ten lines of a result. This provides a more interactive feel and stops us from accidentally crushing our system. Sometimes we do want to compute all of the results and store them someplace.

Blaze expressions are valid sources for odo. So we can store our results in any format.

```
In [19]: iris = bz.Data('sqlite:///my.db::iris')
         query = bz.by(iris.Species, largest=iris.PetalLength.max(),   # A lazily ev
                                     smallest=iris.PetalLength.min())

         odo(query, list)  # A concrete result

Out[19]: [(u'Species', u'PetalLength', u'PetalLength'),
          (u'setosa', 1.9, 1.0),
          (u'versicolor', 5.1, 3.0),
          (u'virginica', 6.9, 4.5)]
```

Exercise: Storage

The solution to the first split-apply-combine problem is below. Store that result in a list, a CSV file,

Exercise: Storage

[illegible]

```
In [21]: [illegible]

In [21]: [illegible]

Out[21]: [(u'[illegible]', 3032230.9[illegible], 25000000.0, 3[illegible]),
 (u'[illegible]', 2[illegible], 2[illegible].0, 36[illegible]),
 (u'[illegible]', 29[illegible], 22[illegible].0, 365.[illegible]),
 (u'[illegible]', 1932538.[illegible], 22000000.0, 361.[illegible]),
 (u'[illegible]', 18[illegible], 22000000.0, 36[illegible]),
 (u'[illegible]', 19[illegible], 2[illegible].0, [illegible]),
 (u'[illegible]', 19[illegible], [illegible].0, 3[illegible]),
 (u'[illegible]', 1[illegible], [illegible], 3[illegible]),
 (u'[illegible]', 1[illegible], [illegible].0, 34[illegible]),
 (u'[illegible]', 1[illegible], [illegible].0, [illegible])]
```

Exercises

Now we load the Lahman baseball database and perform similar queries

```
In [16]: # db = bz.Data('postgresql://postgres:postgres@ec2-54-159-160-163.compute-1
         db = bz.Data('sqlite:///lahman2013.sqlite')
         db.dshape
```

```
Out[16]: dshape("""{
           AllstarFull: var * {
             playerID: ?string,
             yearID: ?int32,
             gameNum: ?int32,
             gameID: ?string,
             teamID: ?string,
             lgID: ?string,
             GP: ?int32,
             startingPos: ?int32
             },
           Appearances: var * {
             yearID: ?int32,
             teamID: ?string,
             lgID: ?string,
             playerID: ?string,
             G_all: ?int32,
             GS: ?int32,
             G_batting: ?int32,
```

```
In [ ]: # View the Salaries table
```

Chrome File Edit View History Bookmarks Window People Help Sun 2:03 PM

Home | Section3-2_bokeh | Section3-1_blaze

localhost:8889/notebooks/Section3-2_bokeh.ipynb

jupyter Section3-2_bokeh (autosaved)

File Edit View Insert Cell Kernel Help Python 2

Markdown Cell Toolbar: None

Introduction to Bokeh

In this tutorial we'll learn how to use Bokeh to build interactive visualizations viewable in a browser. Generally this tutorial will have the following format

1. `charting` - High level interface to go from data to plot
2. `plotting` - Intermediate interface allowing control to all parts of a plot
   - Vectorized attributes
   - Toolbars
   - Linked Brushing
   - Interactivity
3. Baseball example - Example of viewing statistics of baseball

File Edit View Insert Cell Kernel Help Python 2

Introduction to Bokeh

In this tutorial we'll learn how to use Bokeh to build interactive visualizations viewable in a browser. Generally this tutorial will have the following format

1. `charting` - High level interface to go from data to plot
2. `plotting` - Intermediate interface allowing control to all parts of a plot
    - Vectorized attributes
    - Toolbars
    - Linked Brushing
    - Interactivity
3. Baseball example - Example of viewing statistics of baseball

Install

This tutorial uses many different libraries that are all available with the Anaconda Distribution. Once you have Anaconda install, please run these commands from a terminal:

```
$ conda install -y blaze
$ conda install -y bokeh
$ conda install -y odo
$ conda install -y into
```

The last command is there because of an issue with odo which was recently renamed (formerly `into`).

File Edit View Insert Cell Kernel Help | Python 2

Introduction to Bokeh

Provide a first-class visualization library for web-aware applications, without requiring web-level programming.

We wrote JavaScript, so you don't have to.

Write a visualization python. Bokeh creates data descripors and a scenegraph consumed by BokehJS. This works in ipython notebook, creating static files and interacting with dynamic data sources.

Bokeh includes pre-built schemas in bokeh.charts, a low-level composition interface (similar to matplotlib), a server for large and/or dynamic datasources and widgets for providing client-side realtime interaction.

The non-JS framework also has prototypes in other languages (Scala, Julia...maybe R).

Note: There are examples notebooks in bokeh/examples/plotting/notebooks. Start an ipython notebook server there to get more examples.

Gallery -- tutorial -- Documentation -- Repo

In [1]:
```
import pandas as pd
import numpy as np
```

Output

File Edit View Insert Cell Kernel Help Python 2

realtime interaction.

The non-JS framework also has prototypes in other languages (Scala, Julia...maybe R).

Note: There are examples notebooks in bokeh/examples/plotting/notebooks. Start an ipython notebook server there to get more examples.

Gallery -- tutorial -- Documentation -- Repo

In [1]:
```
import pandas as pd
import numpy as np
```

Output

Bokeh can output to html, a notebook, or just fragments for embedding in a web application.

To start playing, we'll use the notebook. Later we will see the other types of output.

In [2]:
```
from bokeh.plotting import output_notebook
output_notebook() # Tell Bokeh to output in an ipython notebook (other opti
```

BokehJS successfully loaded.

bokeh.charts

Common Schemas for common tasks (and parameters).

Expects data to be formatted as either an OrderedDict or a pandas dataframe.

File Edit View Insert Cell Kernel Help | Python 2

```
                 ylabel="frequency",
                 legend="top_left",
                 width=400,
                 height=350,
                 notebook=True)
hist.show()
```

Reset

kwargs, dict_input

normal

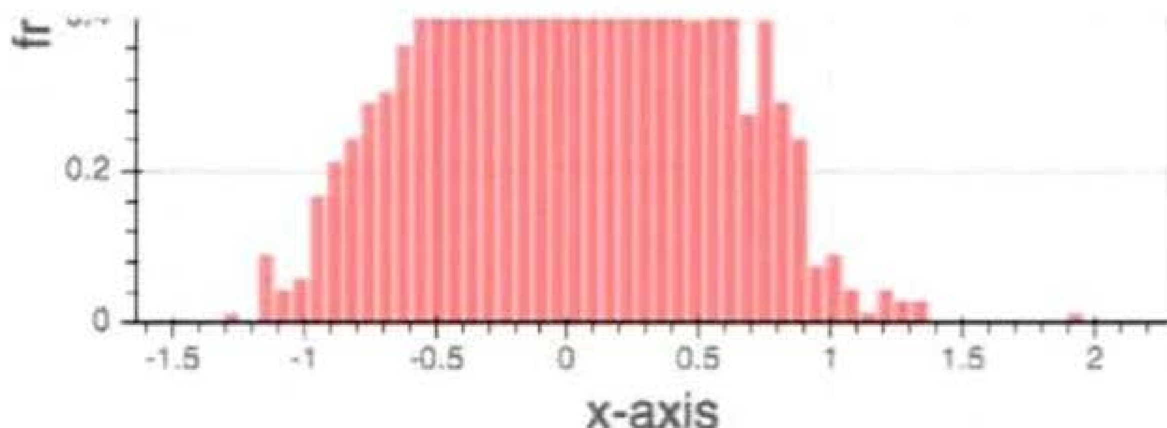
frequency

1
0.8
0.6
0.4
0.2
0

-1.5 -1 -0.5 0 0.5 1 1.5 2

x-axis

File Edit View Insert Cell Kernel Help Python 2

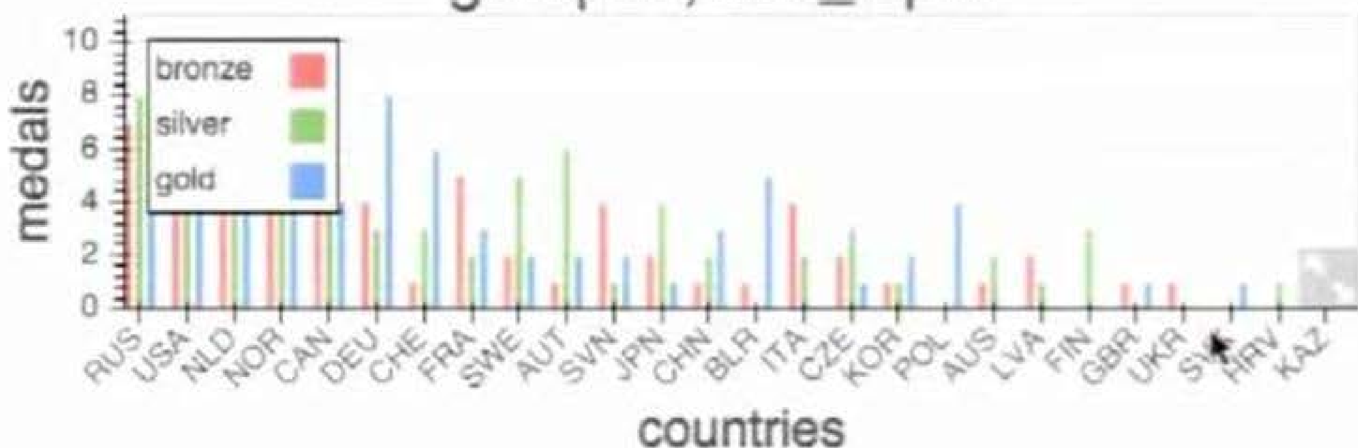



```
In [4]: from bokeh.sampledata.olympics2014 import data
        from bokeh.charts import Bar

        data = {d['abbr']: d['medals'] for d in data['data'] if d['medals']['total'

        countries = sorted(data.keys(), key=lambda x: data[x]['total'], reverse=Tru

        gold = np.array([data[abbr]['gold'] for abbr in countries], dtype=np.float)
        silver = np.array([data[abbr]['silver'] for abbr in countries], dtype=np.fl
        bronze = np.array([data[abbr]['bronze'] for abbr in countries], dtype=np.fl

        medals = OrderedDict(bronze=bronze, silver=silver, gold=gold)
        bar = Bar(medals, countries, title="grouped, dict_input", xlabel="countries
                  legend=True, width=800, height=300, notebook=True)
        bar.show()
```

Plotting py

File Edit View Insert Cell Kernel Help Python 2

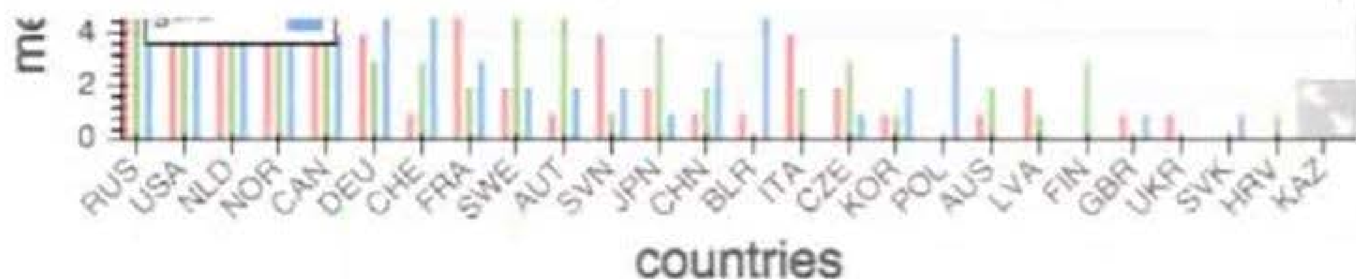
```
bronze = np.array([data[abbr]['bronze'] for abbr in countries], dtype=np.fl

medals = OrderedDict(bronze=bronze, silver=silver, gold=gold)
bar = Bar(medals, countries, title="grouped, dict_input", xlabel="countries
          legend=True, width=800, height=300, notebook=True)
bar.show()
```



Plotting.py

This is a mid-level interface, used by the charting library (and other parts of bokeh). It can also be used directly. The basic idea: there is an active plot, and you are modifying it.

File Edit View Insert Cell Kernel Help Python 2



Plotting.py

This is a mid-level interface, used by the charting library (and other parts of bokeh). It can also be used directly. The basic idea: there is an active plot, and you are modifying it.

In [5]:
```
import numpy as np
from bokeh.plotting import *
N = 102
lin_arr = np.linspace(0, 4*np.pi, N)
sin_arr = np.sin(lin_arr)
cos_arr = np.cos(lin_arr)
```

Scatter Plots

To begin with let's make a scatter plot.

In [6]:
```
p1 = figure()
```

File Edit View Insert Cell Kernel Help | Python 2

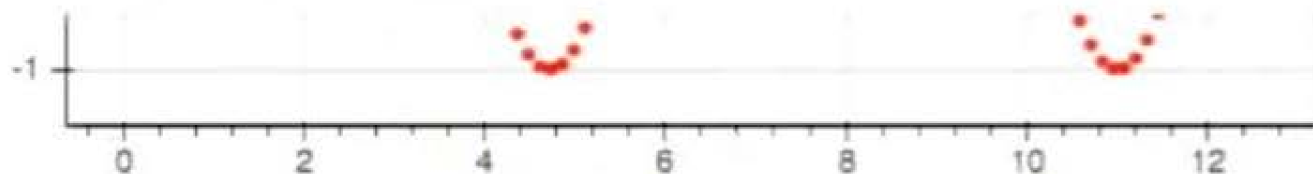
In [5]:
```
import numpy as np
from bokeh.plotting import *
N = 102
lin_arr = np.linspace(0, 4*np.pi, N)
sin_arr = np.sin(lin_arr)
cos_arr = np.cos(lin_arr)
```

Scatter Plots

To begin with let's make a scatter plot.

In [6]:
```
p1 = figure()
p1.scatter(lin_arr, sin_arr, color="#FF00FF")
p1.scatter(lin_arr, cos_arr, color="green")
show(p1)
```

File Edit View Insert Cell Kernel Help | Python 2



In [8]:
```
p3 = figure()
p3.scatter(x=lin_arr, y=cos_arr , marker="square", color="green")
show(p3)
```

Plot

1

0.5

0

File Edit View Insert Cell Kernel Help | Python 2

Everything is vectorized

While we have only been passing vectors for locations, we can do so for almost any parameter.

Let's use the `cos_arr` to size the circles

In [9]:
```
p4 = figure()
p4.scatter(x=lin_arr, y=sin_arr, size=cos_arr**2*10)
show(p4)
```

Let's play with colors now. Brewer is a popular set of palletes. Here we pick one and then build a vector of colors for the plot.

In [10]:
```
from bokeh.palettes import brewer
print "Brewer Palettes:", brewer.keys()
print "Brewer Grey Palettes:", brewer["Greys"].keys()
palette = brewer["Greys"][9] + list(reversed(brewer["Greys"][9]))
colors = palette * (len(lin_arr) / len(palette)) + palette[0:len(lin_arr) %
```

```
Brewer Palettes: ['Spectral', 'RdYlGn', 'OrRd', 'PuBu', 'BuPu', 'RdBu', 'O
ranges', 'BuGn', 'PiYG', 'YlOrBr', 'YlGn', 'RdPu', 'Greens', 'PRGn', 'YlGn
Bu', 'RdYlBu', 'BrBG', 'Purples', 'Reds', 'GnBu', 'Greys', 'RdGy', 'YlOrRd
', 'PuOr', 'PuRd', 'Blues', 'PuBuGn']
Brewer Grey Palettes: [3, 4, 5, 6, 7, 8, 9]
```

File Edit View Insert Cell Kernel Help Python 2

Let's play with colors now. Brewer is a popular set of palletes. Here we pick one and then build a vector of colors for the plot.

```
In [10]: from bokeh.palettes import brewer
         print "Brewer Palettes:", brewer.keys()
         print "Brewer Grey Palettes:", brewer["Greys"].keys()
         palette = brewer["Greys"][9] + list(reversed(brewer["Greys"][9]))
         colors = palette * (len(lin_arr) / len(palette)) + palette[0:len(lin_arr) %
```

```
Brewer Palettes: ['Spectral', 'RdYlGn', 'OrRd', 'PuBu', 'BuPu', 'RdBu', 'O
ranges', 'BuGn', 'PiYG', 'YlOrBr', 'YlGn', 'RdPu', 'Greens', 'PRGn', 'YlGn
Bu', 'RdYlBu', 'BrBG', 'Purples', 'Reds', 'GnBu', 'Greys', 'RdGy', 'YlOrRd
', 'PuOr', 'PuRd', 'Blues', 'PuBuGn']
Brewer Grey Palettes: [3, 4, 5, 6, 7, 8, 9]
```

```
In [11]: p5 = figure()
         p5.scatter(x=lin_arr, y=sin_arr, size=cos_arr**2*10 + 5, fill_color=colors)
         show(p5)
```

Tools

If you notice the bar at the top of the you see several places to interact with the plot.

These are tools and there a many different tools built into the Bokeh.

Let's take a look at HoverTools, but first we use Bokeh's data source which watches changes.

File Edit View Insert Cell Kernel Help Python 2

If you notice the bar at the top of the you see several places to interact with the plot.

These are tools and there a many different tools built into the Bokeh.

Let's take a look at HoverTools, but first we use Bokeh's data source which watches changes.

In [12]:
```
source = ColumnDataSource(
    data=dict(
        x=lin_arr,
        y=sin_arr,
        size=cos_arr**2*10 + 5,
        colors=colors
    )
)
```

In [13]:
```
from bokeh.models import HoverTool
from collections import OrderedDict

TOOLS="crosshair,pan,wheel_zoom,box_zoom,reset,hover,previewsave"
p6 = figure(title="Hoverful Scatter", tools=TOOLS)
p6.circle(x="x", y="y", size="size", source=source,
    fill_color="colors", fill_alpha=0.6, line_color=None)

hover = p6.select(dict(type=HoverTool))
hover.tooltips = OrderedDict([
    ("index", "$index"),
    ("(x,y)", "(@x, @y)"),
    ("size", "@size"),
    ("fill color", "$color[hex, swatch]:fill_color"),
])
show(p6)
```

File Edit View Insert Cell Kernel Help Python 2

```
    data=dict(
        x=lin_arr,
        y=sin_arr,
        size=cos_arr**2*10 + 5,
        colors=colors
    )
)
```

In [13]:
```
from bokeh.models import HoverTool
from collections import OrderedDict

TOOLS="crosshair,pan,wheel_zoom,box_zoom,reset,hover,previewsave"
p6 = figure(title="Hoverful Scatter", tools=TOOLS)
p6.circle(x="x", y="y", size="size", source=source,
    fill_color="colors", fill_alpha=0.6, line_color=None)

hover = p6.select(dict(type=HoverTool))
hover.tooltips = OrderedDict([
    ("index", "$index"),
    ("(x,y)", "(@x, @y)"),
    ("size", "@size"),
    ("fill color", "$color[hex, swatch]:fill_color"),
])
show(p6)
```

Linking two plots

One of the best aspects of Bokeh is linking plots. We can link the brushing. This will allow you to select and pan with the plots both reacting to each other

File Edit View Insert Cell Kernel Help | Python 2

Linking two plots

One of the best aspects of Bokeh is linking plots. We can link the brushing. This will allow you to select and pan with the plots both reacting to each other.

In [14]:

```
N = 300
x = np.linspace(0, 4*np.pi, N)
y1 = np.sin(x)
y2 = np.cos(x)

source = ColumnDataSource()
source.add(data=x, name='x')
source.add(data=y1, name='y1')
source.add(data=y2, name='y2')

TOOLS = "pan,wheel_zoom,box_zoom,reset,save,box_select,lasso_select"

s1 = figure(tools=TOOLS, plot_width=350, plot_height=350)
s1.scatter('x', 'y1', source=source)

# Linked brushing in Bokeh is expressed by sharing data sources between
# renderers. Note below that s2.scatter is called with the `source`
# keyword argument, and supplied with the same data source from s1.scatter
s2 = figure(tools=TOOLS, plot_width=350, plot_height=350, x_range=s1.x_ranc
s2.scatter('x', 'y2', source=source, )

p = gridplot([[s1,s2]])
show(p)
```

File Edit View Insert Cell Kernel Help Python 2

Basic interactivity

Bokeh lets you use a Python update function to update your plots.

In IPython notebook we can use the interactive widgets provided by the notebook. One can also use Flask or Bokeh-server to embed in outside the notebook.

```
In [15]: x = np.linspace(0, 2*np.pi, 2000)
         y = np.sin(x)

         source = ColumnDataSource(data=dict(x=x, y=y))

         p = figure(title="simple line example", plot_height=300, plot_width=600)
         p.line(x, y, color="#2222aa", line_width=3, source=source, name="foo")

Out[15]: <bokeh.plotting.Figure at 0x10b77b310>

In [16]: from IPython.html.widgets import interact
         @interact(f=["sin", "cos", "tan"], w=(0,100), A=(1,10), phi=(0, 10, 0.1))
         def update(f, w=1, A=1, phi=0):
             if   f == "sin": func = np.sin
             elif f == "cos": func = np.cos
             elif f == "tan": func = np.tan
             source.data['y'] = A * func(w * x + phi)
             source.push_notebook()
```

File Edit View Insert Cell Kernel Help Python 2

```
In [15]: x = np.linspace(0, 2*np.pi, 2000)
         y = np.sin(x)

         source = ColumnDataSource(data=dict(x=x, y=y))

         p = figure(title="simple line example", plot_height=300, plot_width=600)
         p.line(x, y, color="#2222aa", line_width=3, source=source, name="foo")

Out[15]: <bokeh.plotting.Figure at 0x10c100d50>
```

```
In [16]: from IPython.html.widgets import interact
         @interact(f=["sin", "cos", "tan"], w=(0,100), A=(1,10), phi=(0, 10, 0.1))
         def update(f, w=1, A=1, phi=0):
             if   f == "sin": func = np.sin
             elif f == "cos": func = np.cos
             elif f == "tan": func = np.tan
             source.data['y'] = A * func(w * x + phi)
             source.push_notebook()
```

```
In [17]: show(p)
```

Baseball App Example

In this example we use Blaze and Bokeh to explore the Lahman Baseball Statistics database.

File Edit View Insert Cell Kernel Help Python 2

Baseball App Example

In this example we use Blaze and Bokeh to explore the Lahman Baseball Statistics database.

In [22]:
```
import blaze as bz
import pandas as pd
import numpy as np
from odo import odo
from bokeh.plotting import *
output_notebook()
```

In [19]:
```
db = bz.Data('sqlite:///lahman2013.sqlite')
db.dshape
```

Out[19]:
```
dshape("""{
  AllstarFull: var * {
    playerID: ?string,
    yearID: ?int32,
    gameNum: ?int32,
    gameID: ?string,
    teamID: ?string,
    lgID: ?string,
    GP: ?int32,
    startingPos: ?int32
    },
  Appearances: var * {
    yearID: ?int32,
    teamID: ?string,
    lgID: ?string,
    playerID: ?string,
```

File Edit View Insert Cell Kernel Help Python 2

```
        G_all: ?int32,
        GS: ?int32,
        G_batting: ?int32,
        G_defense: ?int32,
        G_p: ?int32,
        G_c: ?int32,
        G_1b: ?int32,
        G_2b: ?int32,
        G_3b: ?int32,
        G_ss: ?int32,
        G_lf: ?int32,
        G_cf: ?int32,
        G_rf: ?int32,
        G_of: ?int32,
        G_dh: ?int32,
        G_ph: ?int32,
        G_pr: ?int32
        },
      AwardsManagers: var * {
```

```
In [20]: list(db.Salaries.teamID.distinct())
```

```
Out[20]: [u'ATL',
          u'BAL',
          u'BOS',
          u'CAL',
          u'CHA',
          u'CHN',
          u'CIN',
          u'CLE',
          u'DET',
          u'HOU',
          u'KCA',
```

19	PIT
20	SDN
21	SEA
22	SFN
23	SLN
24	TEX
25	TOR
26	COL
27	FLO
28	ANA
29	ARI
30	MIL
31	TBA
32	LAA
33	WAS
34	MIA

In [24]:
```
result = bz.by(db.Salaries.teamID, avg=db.Salaries.salary.mean(),
                                   max=db.Salaries.salary.max(),
                                   ratio=db.Salaries.salary.max() / db.Sala
              ).sort('ratio', ascending=False)
df = odo(result, pd.DataFrame)
```

File Edit View Insert Cell Kernel Help Python 2

32	LAA
33	WAS
34	MIA

In [24]:
```
result = bz.by(db.Salaries.teamID, avg=db.Salaries.salary.mean(),
                                   max=db.Salaries.salary.max(),
                                   ratio=db.Salaries.salary.max() / db.Sala
              ).sort('ratio', ascending=False)
df = odo(result, pd.DataFrame)
```

In [25]:
```
df = df.sort('avg')
source = ColumnDataSource(df)
p = figure(x_range=list(df["teamID"]))
p.scatter(x="teamID", y="avg", source=source)
show(p)
```

Hmm, can't read the y axis very well...

In [26]:
```
df = df.sort('avg')
source = ColumnDataSource(df)
p = figure(x_range=list(df["teamID"]))
p.scatter(x="teamID", y="avg", source=source)
p.xaxis.major_label_orientation = np.pi/3

show(p)
```

Let's view a max versus ratio

File Edit View Insert Cell Kernel Help | Python 2

32	LAA
33	WAS
34	MIA

In [21]:
```
result = bz.by(db.Salaries.teamID, avg=db.Salaries.salary.mean(),
                                   max=db.Salaries.salary.max(),
                                   ratio=db.Salaries.salary.max() / db.Sala
              ).sort('ratio', ascending=False)
df = odo(result, pd.DataFrame)
```

In [ ]:

In [25]:
```
df = df.sort('avg')
source = ColumnDataSource(df)
p = figure(x_range=list(df["teamID"]))
p.scatter(x="teamID", y="avg", source=source)
show(p)
```

In [ ]:

In [ ]:

Hmm, can't read the y axis very well...

In [26]:
```
df = df.sort('avg')
source = ColumnDataSource(df)
p = figure(x_range=list(df["teamID"]))
```

File Edit View Insert Cell Kernel Help

32	LAA
33	WAS
34	MIA

In [21]:
```
result = bz.by(db.Salaries.teamID, avg=db.Salaries.salary.mean(),
                                   max=db.Salaries.salary.max(),
                                   ratio=db.Salaries.salary.max() / db.Sala
               ).sort('ratio', ascending=False)
df = odo(result, pd.DataFrame)
```

In [22]:
```
df.head()
```

Out[22]:

	teamID	avg	max	ratio
0	PHI	2092230.932836	25000000	416.666667
1	LAN	2098380.932036	23854494	397.574906
2	NYN	2317349.977246	23145011	385.750183
3	DET	1980834.990208	23000000	383.333333
4	MIN	1525031.650386	23000000	383.333333

In [23]:
```
df = df.sort('avg')
source = ColumnDataSource(df)
p = figure(x_range=list(df['teamID']))
p.scatter(x='teamID', y='avg', source=source)
show(p)
```

File Edit View Insert Cell Kernel Help | Python 2

32	LAA
33	WAS
34	MIA

In [21]:
```
ɔ.Salaries.teamID, avg=db.Salaries.salary.mean(),
                   max=db.Salaries.salary.max(),
                   ratio=db.Salaries.salary.max() / db.Salaries.salary.min()
).sort('ratio', ascending=False)
 pd.DataFrame)
```

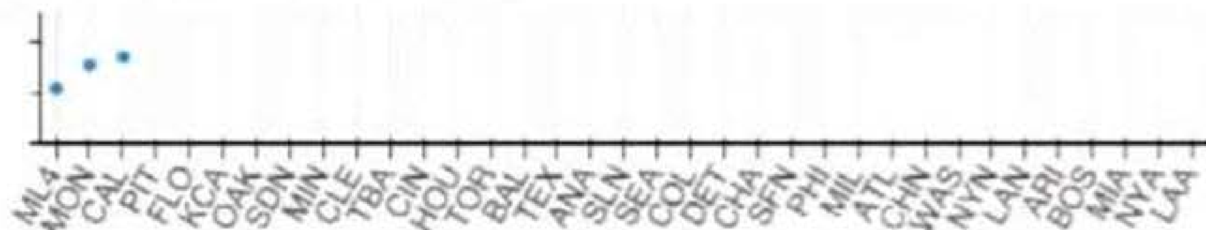
In [22]:
```
df.head()
```

Out[22]:

	teamID	avg	max	ratio
0	PHI	2092230.932636	25000000	416.666667
1	LAN	2346982.698026	23854494	397.574900
2	NYN	2317349.977246	23145011	385.750183
3	DET	1980834.990208	23000000	383.333333
4	MIN	1525031.650386	23000000	383.333333

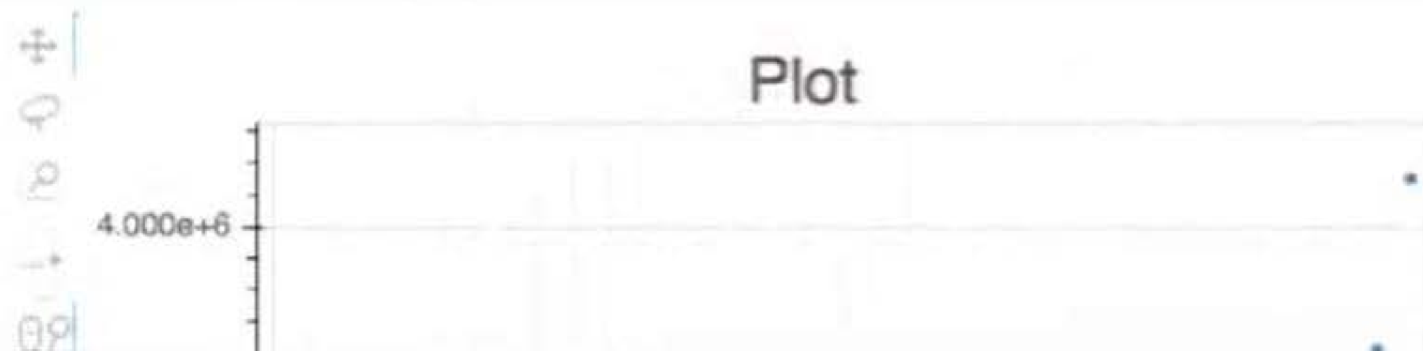
In [25]:
```
df = df.sort('avg')
source = ColumnDataSource(df)
p = figure(x_range=list(df["teamID"]))
p.scatter(x="teamID", y="avg", source=source)
show(p)
```

File Edit View Insert Cell Kernel Help | Python 2



Let's view a max versus ratio

In [25]:
```
TOOLS = "pan,wheel_zoom,box_zoom,reset,save,box_select,lasso_select"

df = df.sort('avg')
source = ColumnDataSource(df)
s1 = figure(x_range=list(df["teamID"]), tools=TOOLS)
s1.scatter(x="teamID", y="avg", source=source)
s1.xaxis.major_label_orientation = np.pi/3

s2 = figure(x_range=list(df["teamID"]), tools=TOOLS)
s2.scatter(x="teamID", y="ratio", source=source)
s2.xaxis.major_label_orientation = np.pi/3

p = gridplot([[s1],[s2]])
show(p)
```



File Edit View Insert Cell Kernel Help | Python 2

```
m = odo(r, pd.DataFrame)["all_stars"].max()
print "max number of all stars from a single team:", m

print "normalized list of all_stars:\n", bz.compute((r.all_stars / m).head(

# Now let's use this as the size of the circles in the scatter plot
df1 = into(pd.DataFrame, r)
df1['all_stars'] /= (df1['all_stars'].max() / 10)
df1['all_stars'] += 10
```

```
max number of all stars from a single team: 412
normalized list of all_stars:
0    1.000000
1    0.737864
2    0.691748
3    0.623786
4    0.570388
5    0.565534
6    0.512136
7    0.492718
8    0.458738
9    0.453883
Name: all_stars, dtype: float64
```

Now lets join the data to all_star sizes

In [29]:
```
r = bz.join(bz.Data(df1), bz.Data(df), 'teamID')
r.head()
```

Out[29]:

	teamID	all_stars	avg	max	ratio

File Edit View Insert Cell Kernel Help

Python 2

In [31]:
```
def compute_df(year=2012):
    result = db.Salaries[ db.Salaries.yearID==year ]
    result = bz.Data(into(pd.DataFrame, result))
    result = bz.by(result.teamID, max=result.salary.max()).sort('max', asce
    df = into(pd.DataFrame, result)
    asf_year = db.AllstarFull[ db.AllstarFull.yearID==year]
    result = bz.by(asf_year.teamID, all_stars=db.AllstarFull.playerID.count
                   ).sort('all_stars', ascending=False)
    r = bz.Data(into(pd.DataFrame, result))
    df1 = into(pd.DataFrame, r)
    df1['all_stars'] /= (df1['all_stars'].max() / 10)
    df1['all_stars'] += 10
    r = bz.join(bz.Data(df1), bz.Data(df), 'teamID')
    df_j = into(pd.DataFrame, r)
    df_j = df_j.sort("max")
    return df_j

source = into(ColumnDataSource, compute_df())

p = figure(x_range=list(source.data["teamID"]))
p.scatter(x="teamID", y="max", size="all_stars", source=source, fill_alpha=
p.xaxis.major_label_orientation = np.pi/3
```

In [32]:
```
from IPython.html.widgets import interact, IntSliderWidget

def update(year):
    df = compute_df(year)
    source.data['all_stars'] = df['all_stars']
    source.data['max'] = df['max']
    source.push_notebook()
#interact(update, year=(1980, 2013))
interact(update, year=IntSliderWidget(min=1985, max=2013, value=2013))
```

File Edit View Insert Cell Kernel Help | Python 2

```
p.xaxis.major_label_orientation = np.pi/3
```

In [27]:
```
from IPython.html.widgets import interact, IntSliderWidget

def update(year):
    df = compute_df(year)
    source.data['all_stars'] = df['all_stars']
    source.data['max'] = df['max']
    source.push_notebook()
#interact(update, year=(1980, 2013))
interact(update, year=IntSliderWidget(min=1985, max=2013, value=2013))
```

year 2013

Javascript error adding output!
TypeError: Cannot read property 'set' of undefined
See your browser Javascript console for more details.

Out[27]:
```
<function __main__.update>
```

In [28]:
```
show(p)
```