Financial Programming and Databases

WANG, Zigan (王自干)
Ph.D., Columbia University
Faculty of Business Economics, The University of Hong Kong

Course Outline

- Data management with Python (3 weeks)
  - Packages
    - Numpy/Scipy
    - Dataframe
    - Pandas
  - OOP
    - Class
  - Geolocation
  - Multiprocessing
  - Pickle, Json, HDF5 (.h5)
  - Web scraping (regular expression, selenium)

External Libraries

External libraries

A very complete list can be found at PyPi the Python Package Index:
https://pypi.python.org/pypi

To install, use pip, which comes with Python:

```
pip install package
```

or download, unzip, and run the installer directly from the directory:

```
python setup.py install
```

If you have Python 2 and Python 3 installed, use pip3 (though not with Anaconda) or make sure the right version is first in your PATH.

Python Libraries for Data Science

NumPy:

- introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects
- provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance
- many other python libraries are built on NumPy

Link: http://www.numpy.org/

Numpy

http://www.numpy.org/

Mathematics and statistics, especially multi-dimensional array manipulation for data processing.

Good introductory tutorials by Software Carpentry:

http://swcarpentry.github.io/python-novice-inflammation/

Numpy data

Perhaps the nicest thing about numpy is its handling of complicated 2D datasets. It has its own array types which overload the indexing operators. Note the difference in the below from the standard [1d][2d] notation:

```
import numpy
data = numpy.int_([
[1,2,3,4,5],
[10,20,30,40,50],
[100,200,300,400,500]
])

print(data[0,0])                    #       1
print(data[1:3,1:3])                #       [[20 30][200 300]]
```

On a standard list, data[1:3][1:3] wouldn't work, at best data[1:3][0][1:3] would give you [20][30]

Numpy operations

You can additionally do maths on the arrays, including matrix manipulation.

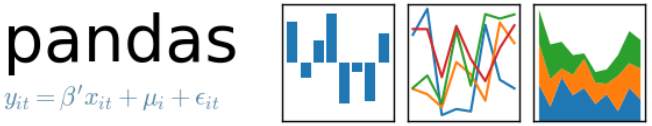
```
import numpy
data = numpy.int_([
[1,2,3,4,5],
[10,20,30,40,50],
[100,200,300,400,500]
])
print(data[1:3,1:3] - 10)                  #     [[10 20],[190 290]]
print(numpy.transpose(data[1:3,1:3])) #     [[20 200],[30 300]]
```

Python Libraries for Data Science

SciPy:

- collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more
- part of SciPy Stack
- built on NumPy

Link: https://www.scipy.org/scipylib/



Python Libraries for Data Science

Pandas:

- adds data structures and tools designed to work with table-like data (similar to Series and Data Frames in R)
- provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- allows handling missing data

Link: http://pandas.pydata.org/

Pandas

http://pandas.pydata.org/

Data analysis.

Based on Numpy, but adds more sophistication.

Pandas data

Pandas data focuses around DataFrames, 2D arrays with addition abilities to name and use rows and columns.

```
import pandas
df = pandas.DataFrame(
        data,                   # numpy array from before.
        index=['i','ii','iii'],
        columns=['A','B','C','D','E']
)
print (data['A'])
print(df.mean(0)['A'])
print(df.mean(1)['i'])
```

Prints:

```
i       1
ii      10
iii     100
Name: A, dtype: int32
37.0
3.0
```

Python Libraries for Data Science

SciKit-Learn:

- provides machine learning algorithms: classification, regression, clustering, model validation etc.

- built on NumPy, SciPy and matplotlib

Link: http://scikit-learn.org/

scikit-learn

http://scikit-learn.org/

Scientific analysis and machine learning.

Used for machine learning. Founded on Numpy data formats.

Python Libraries for Data Science

matplotlib:

- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- a set of functionalities similar to those of MATLAB
- line plots, scatter plots, barcharts, histograms, pie charts etc.
- relatively low-level; some effort needed to create advanced visualization

Link: https://matplotlib.org/

Python Libraries for Data Science

Seaborn:

- based on matplotlib
- provides high level interface for drawing attractive statistical graphics
- Similar (in style) to the popular ggplot2 library in R

Link: https://seaborn.pydata.org/

Beautiful Soup

https://www.crummy.com/software/BeautifulSoup/
Web analysis.

Need other packages to actually download pages like the library `requests`.
http://docs.python-requests.org/en/master/

BeautifulSoup navigates the Document Object Model:
http://www.w3schools.com/

Not a library, but a nice intro to web programming with Python.
https://wiki.python.org/moin/WebProgramming

Tweepy

http://www.tweepy.org/

Downloading Tweets for analysis.

You'll also need a developer key:

http://themepacific.com/how-to-generate-api-key-consumer-token-access-key-for-twitter-oauth/994/

Most social media sites have equivalent APIs (functions to access them) and modules to use those.

NLTK

http://www.nltk.org/

Natural Language Toolkit.

Parse text and analyse everything from Parts Of Speech to positivity or negativity of statements (sentiment analysis).

Celery

http://www.celeryproject.org/

Concurrent computing / parallelisation.

For splitting up programs and running them on multiple computers e.g. to remove memory limits.

See also:

https://docs.python.org/3/library/concurrency.html

Loading Python Libraries

```
In [ ]: #Import Python Libraries
        import numpy as np
        import scipy as sp
        import pandas as pd
        import matplotlib as mpl
        import seaborn as sns
```

Press `Shift+Enter` to execute the *jupyter* cell

Start Jupyter nootebook

```
# On the Shared Computing Cluster
[scc1 ~] jupyter notebook
```

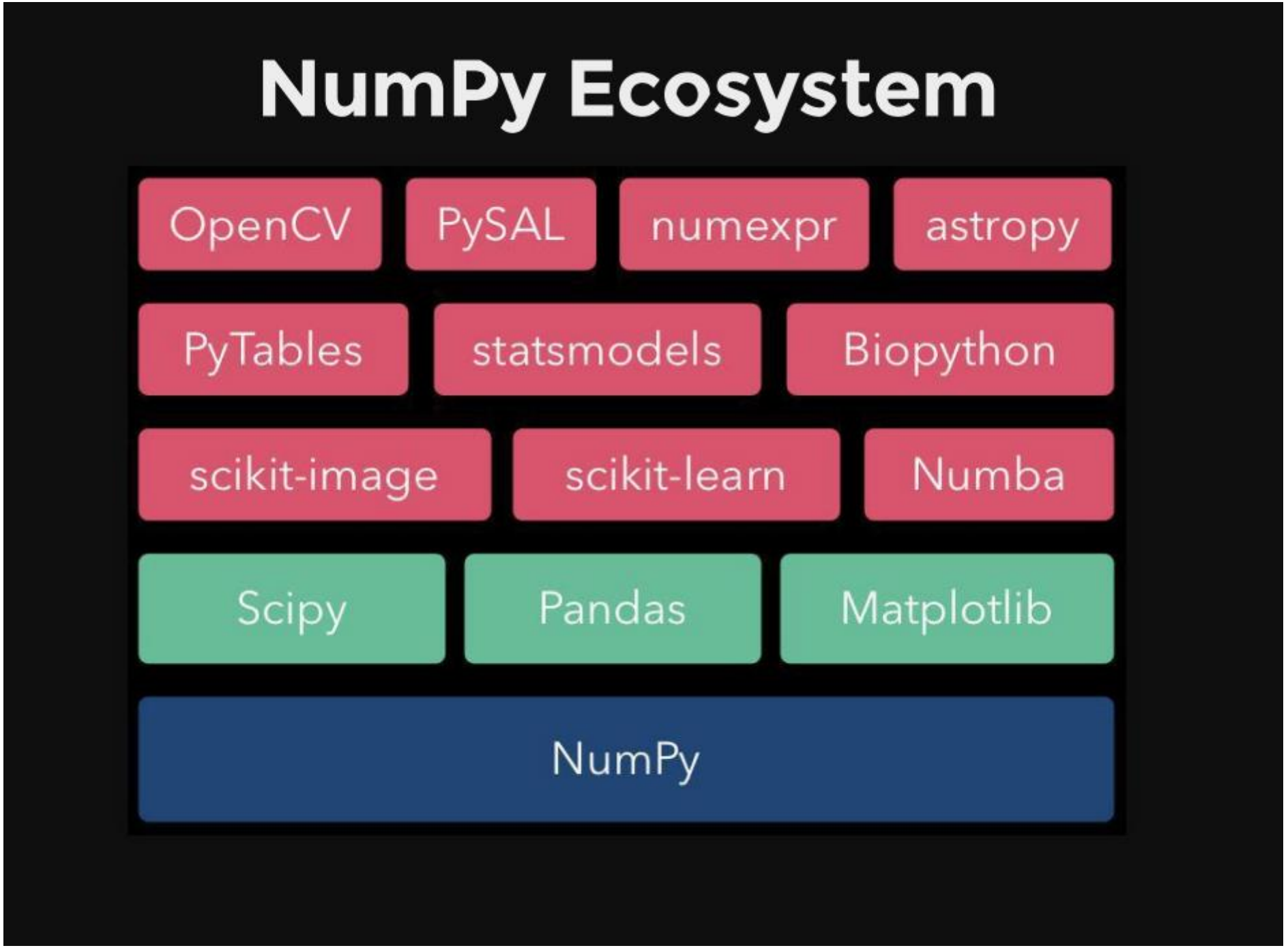
Course Outline

- Data management with Python (3 weeks)
  - Packages
    - Numpy/Scipy
    - Dataframe
    - Pandas
  - OOP
    - Class
  - Geolocation
  - Multiprocessing
  - Pickle, Json, HDF5 (.h5)
  - Web scraping (regular expression, selenium)

NumPy

- NumPy is a Python C extension library for array-oriented computing
  - Efficient
  - In-memory
  - Contiguous (or Strided)
  - Homogeneous (but types can be algebraic)

- NumPy is suited to many applications
  - Image processing
  - Signal processing
  - Linear algebra
  - A plethora of others

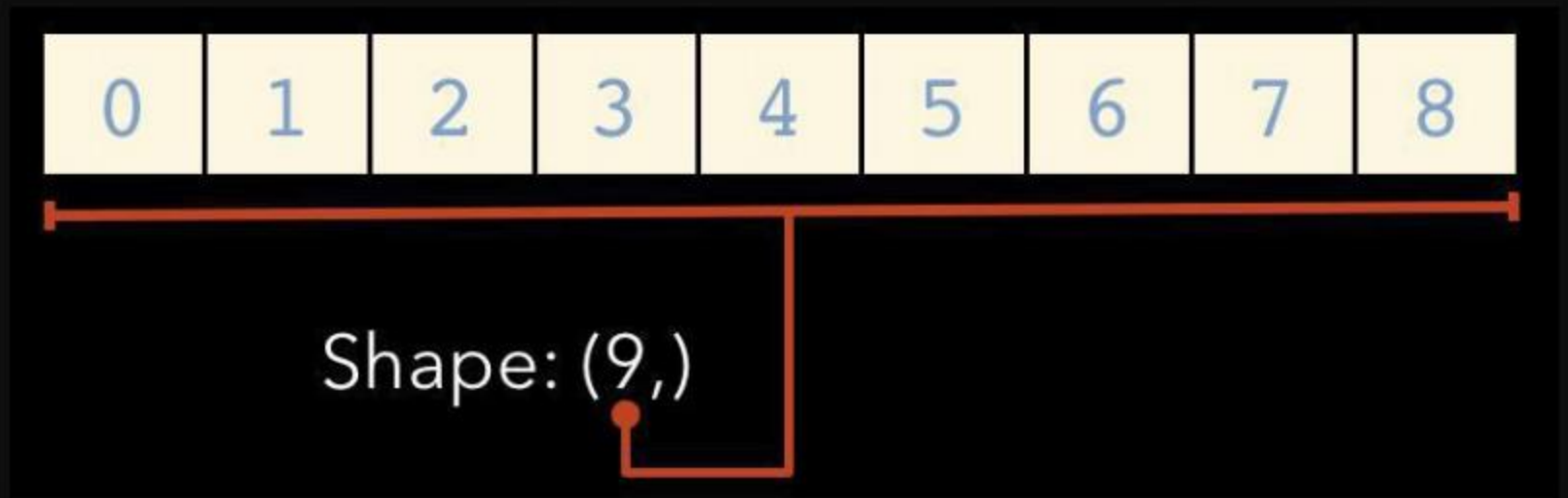

NumPy Ecosystem
OpenCV
PySAL
numexpr
astropy
PyTables
statsmodels
Biopython
scikit-image
scikit-learn
Numba
Scipy
Pandas
Matplotlib
NumPy

Quick Start

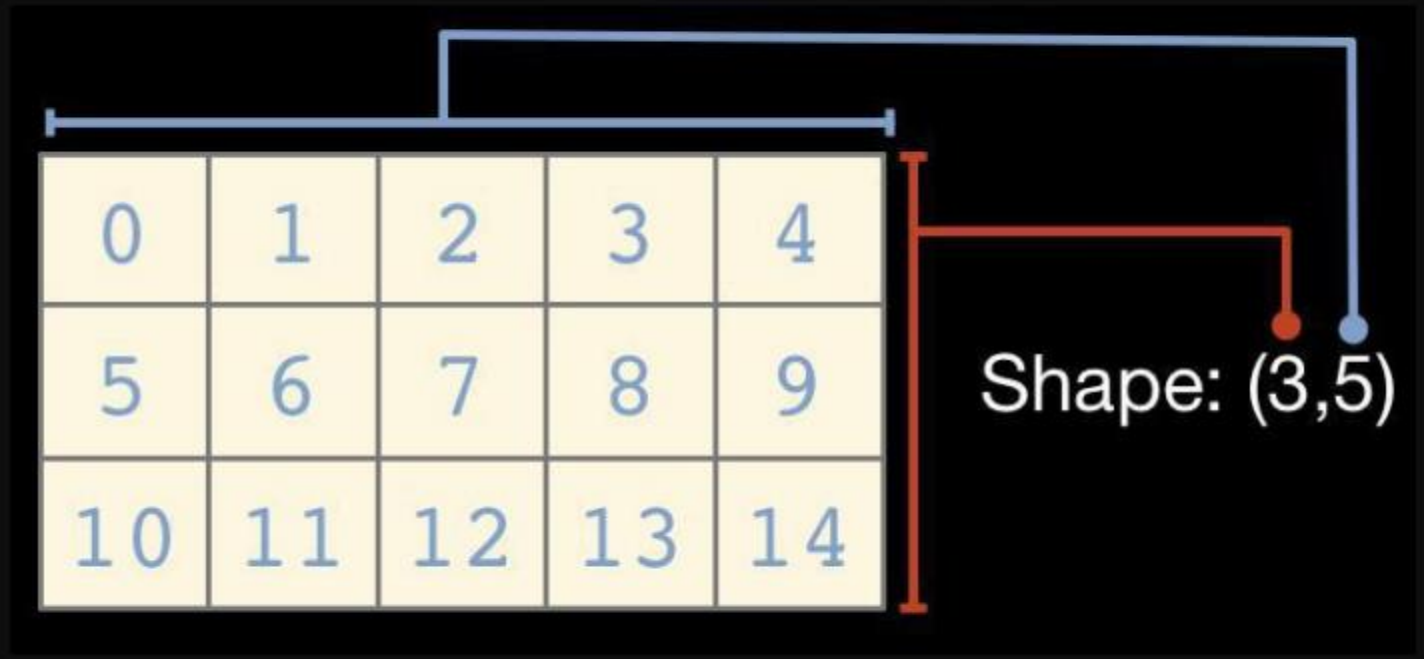
```
In [1]: import numpy as np

In [2]: a = np.array([1,2,3,4,5,6,7,8,9])

In [3]: a
Out[3]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])

In [4]: b = a.reshape((3,3))

In [5]: b
Out[5]:
array([[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])

In [6]: b * 10 + 4
Out[6]:
array([[14, 24, 34],
[44, 54, 64],
[74, 84, 94]])
```

Array Shape

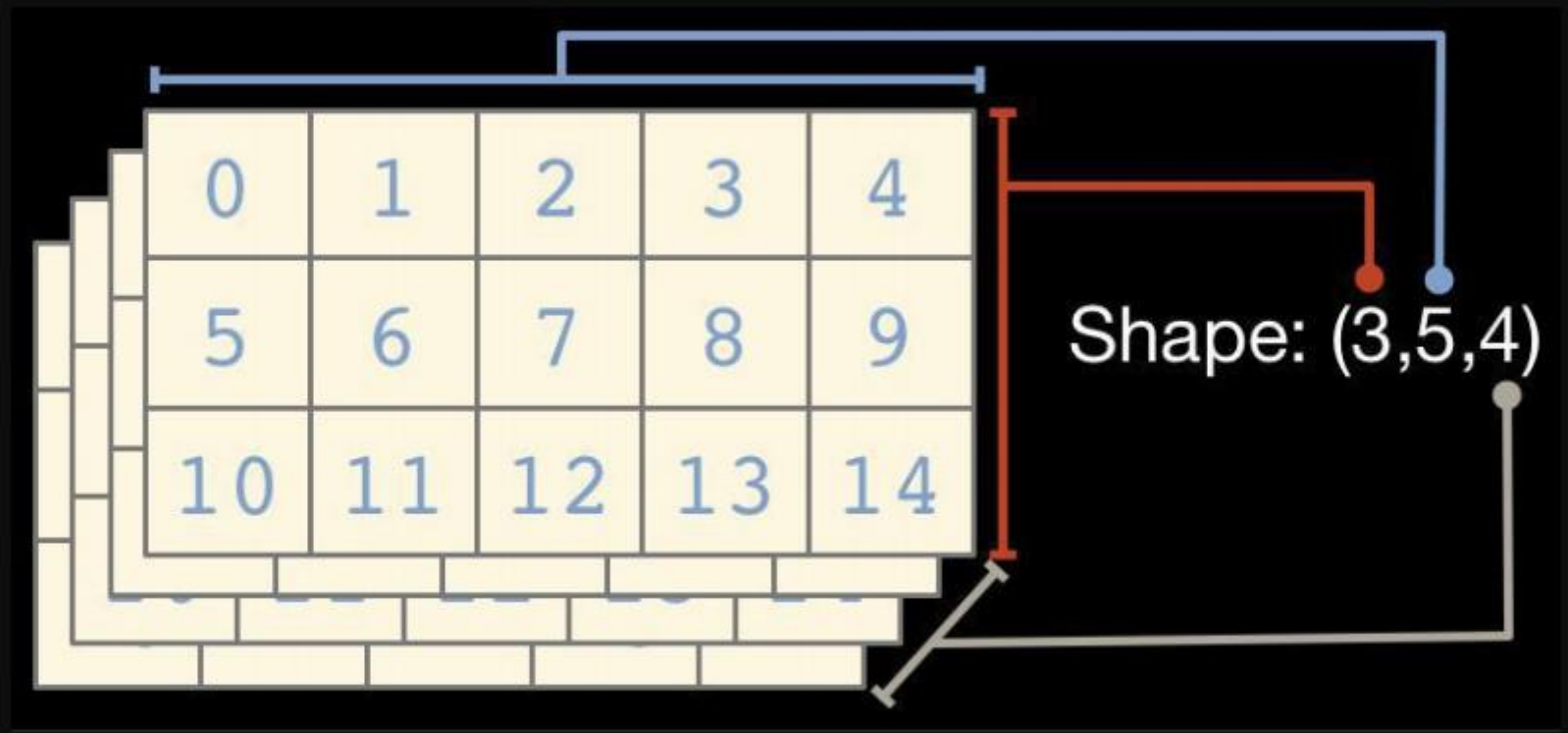
One dimensional arrays have a 1-tuple for their shape



...Two dimensional arrays have a 2-tuple



...And so on



Array Creation

Explicitly from a list of values

```
In [2]: np.array([1,2,3,4])
Out[2]: array([1, 2, 3, 4])
```

As a range of values

```
In [3]: np.arange(10)
Out[3]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

By specifying the number of elements

```
In [4]: np.linspace(0, 1, 5)
Out[4]: array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ])
```

Zero-initialized

```
In [4]: np.zeros((2,2))
Out[4]:
array([[ 0.,  0.],
[ 0.,  0.]])
```

One-initialized

```
In [5]: np.ones((1,5))
Out[5]: array([[ 1.,  1.,  1.,  1.,  1.]])
```

Uninitialized

```
In [4]: np.empty((1,3))
Out[4]: array([[  2.12716633e-314,   2.12716633e-314,   2.15203762e-314]])
```

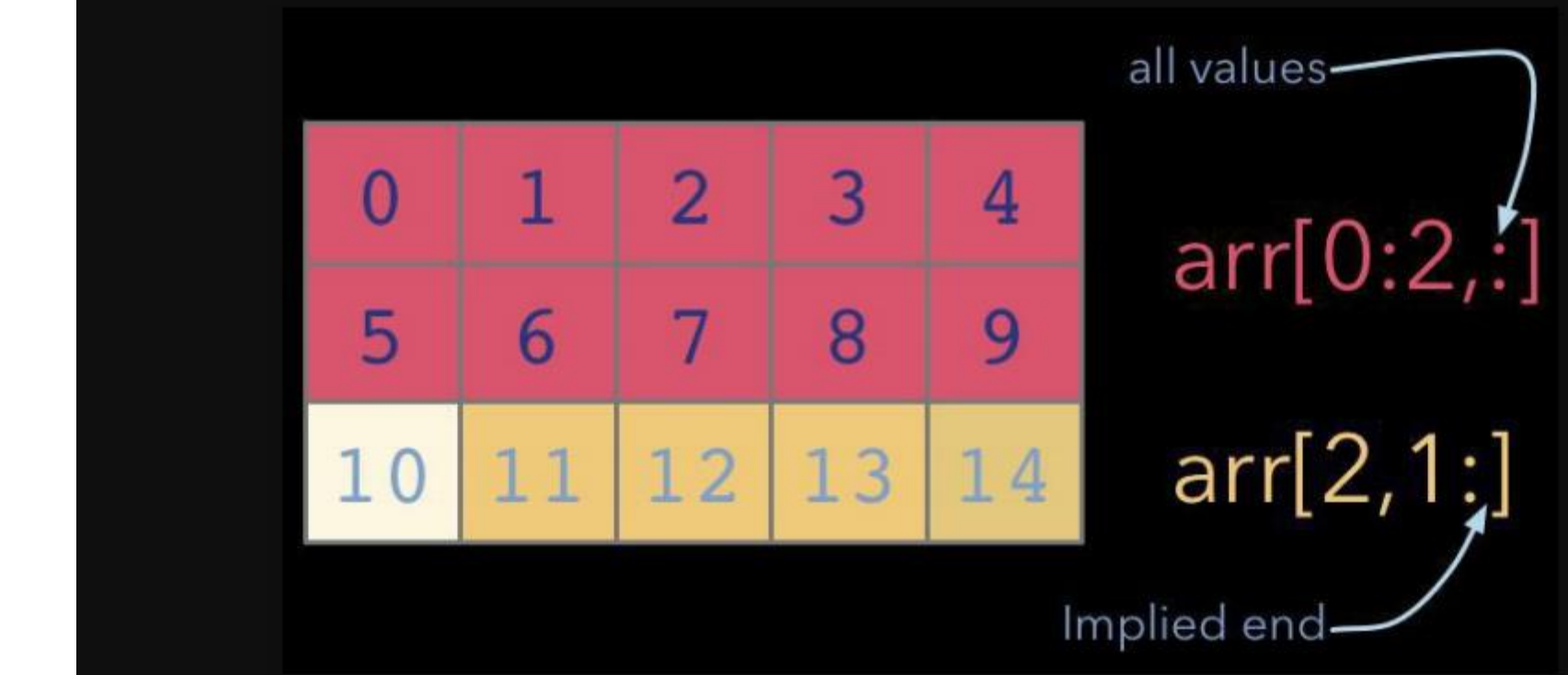
Constant diagonal value

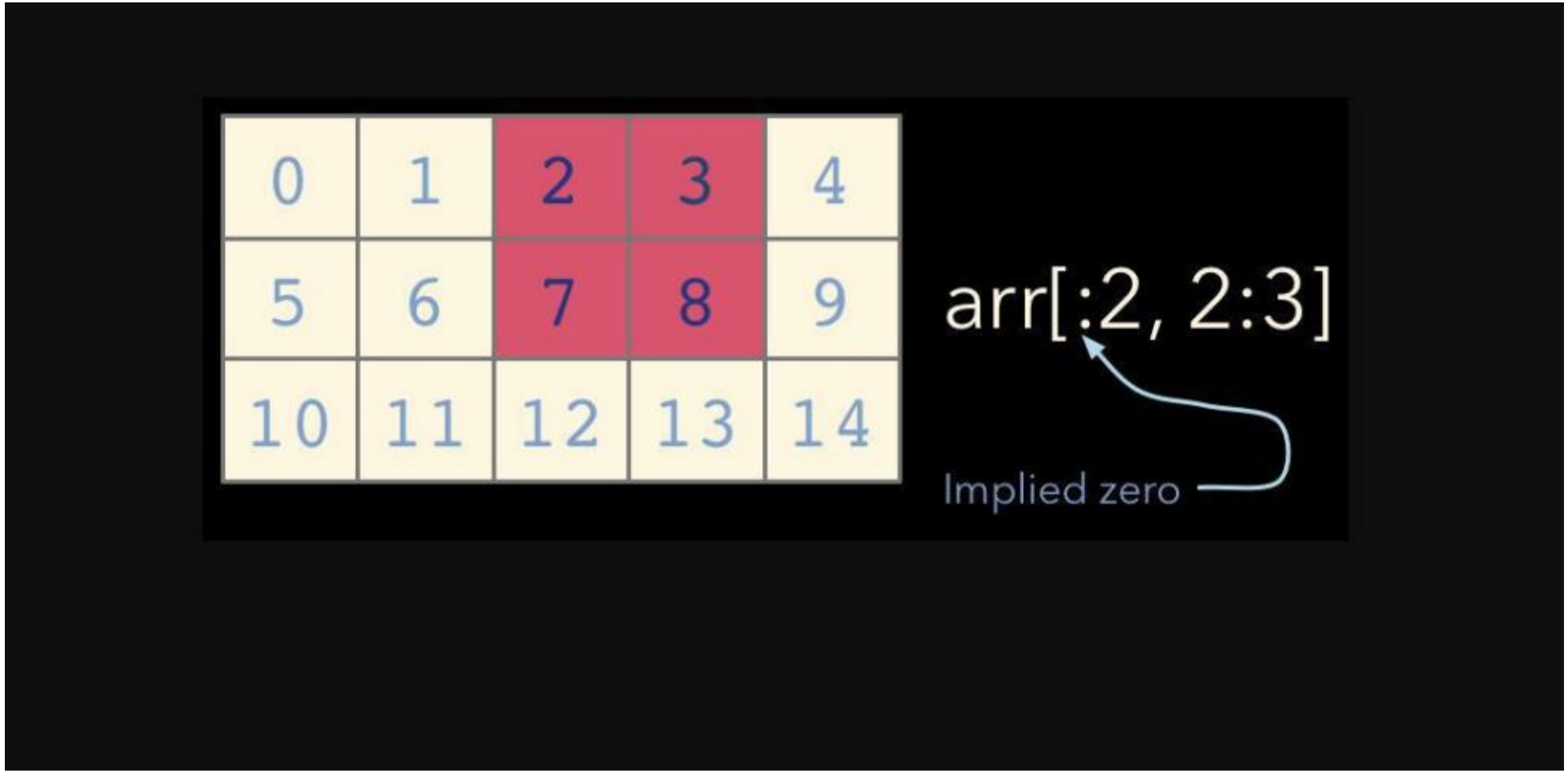
```
In [6]: np.eye(3)
Out[6]:
array([[ 1.,  0.,  0.],
[ 0.,  1.,  0.],
[ 0.,  0.,  1.]])
```

Multiple diagonal values

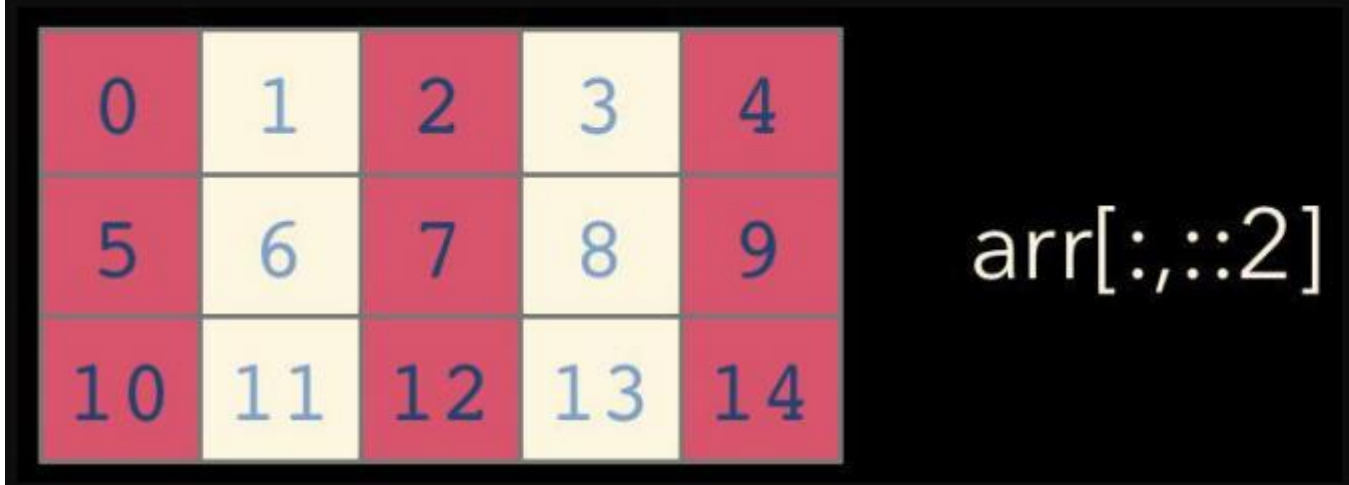
```
In [7]: np.diag([1,2,3,4])
Out[7]:
array([[1, 0, 0, 0],
[0, 2, 0, 0],
[0, 0, 3, 0],
[0, 0, 0, 4]])
```

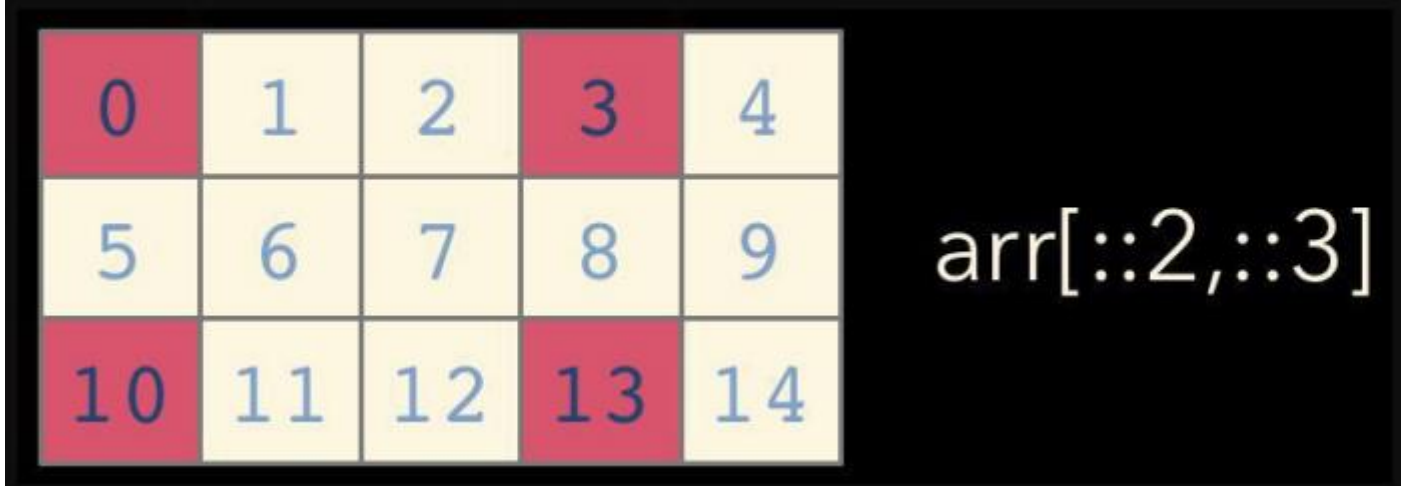
Indexing and Slicing




0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
arr[:2, 2:3]
Implied zero

NumPy array indices can also take an optional stride





```
a = np.arange(10).reshape((2,5))

a.ndim     # 2 dimension
a.shape    # (2, 5) shape of array
a.size     # 10 # of elements
a.T        # transpose
a.dtype    # data type
```

Arithmetic operators: **elementwise** application

```
a = np.arange(4)
# array([0, 1, 2, 3])

b = np.array([2, 3, 2, 4])

a * b  # array([ 0,  3,  4, 12])
b - a  # array([2, 2, 0, 1])

c = [2, 3, 4, 5]
a * c  # array([ 0,  3,  8, 15])
```

Also, we can use += and *=.

a += b: a = a+b

Introducing NumPy Arrays

SIMPLE ARRAY CREATION

```
>>> a = array([0,1,2,3])
>>> a
array([0, 1, 2, 3])
```

CHECKING THE TYPE

```
>>> type(a)
<type 'array'>
```

NUMERIC 'TYPE' OF ELEMENTS

```
>>> a.dtype
dtype('int32')
```

BYTES PER ELEMENT

```
>>> a.itemsize # per element
4
```

ARRAY SHAPE

```
# shape returns a tuple
# listing the length of the
# array along each dimension.
>>> a.shape
(4,)
>>> shape(a)
(4,)
```

ARRAY SIZE

```
# size reports the entire
# number of elements in an
# array.
>>> a.size
4
>>> size(a)
4
```

Introducing NumPy Arrays

BYTES OF MEMORY USED

```
# returns the number of bytes
# used by the data portion of
# the array.
 >>> a.nbytes
12
```

NUMBER OF DIMENSIONS

```
>>> a.ndim
1
```

ARRAY COPY

```
# create a copy of the array
>>> b = a.copy()
>>> b
array([0, 1, 2, 3])
```

CONVERSION TO LIST

```
# convert a numpy array to a
# python list.
>>> a.tolist()
[0, 1, 2, 3]

# For 1D arrays, list also
# works equivalently, but
# is slower.
>>> list(a)
[0, 1, 2, 3]
```

Setting Array Elements

ARRAY INDEXING

```
>>> a[0]
0
>>> a[0] = 10
>>> a
[10, 1, 2, 3]
```

FILL

```
# set all values in an array.
>>> a.fill(0)
>>> a
[0, 0, 0, 0]

# This also works, but may
# be slower.
>>> a[:] = 1
>>> a
[1, 1, 1, 1]
```

BEWARE OF TYPE COERSION

```
>>> a.dtype
dtype('int32')

# assigning a float to into
# an int32 array will
# truncate decimal part.
>>> a[0] = 10.6
>>> a
[10, 1, 2, 3]

# fill has the same behavior
>>> a.fill(-4.8)
>>> a
[-4, -4, -4, -4]
```

Multi-Dimensional Arrays

MULTI-DIMENSIONAL ARRAYS

```
>>> a = array([[ 0, 1, 2, 3],
               [10,11,12,13]])
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,13]])
```

(ROWS,COLUMNS)

```
>>> a.shape
(2, 4)
>>> shape(a)
(2, 4)
```

ELEMENT COUNT

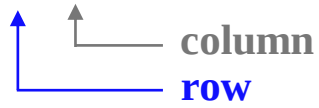
```
>>> a.size
8
>>> size(a)
8
```

NUMBER OF DIMENSIONS

```
>>> a.ndims
2
```

GET/SET ELEMENTS

```
>>> a[1,3]
13
```




```
>>> a[1,3] = -1
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,-1]])
```

ADDRESS FIRST ROW USING SINGLE INDEX

```
>>> a[1]
array([10, 11, 12, -1])
```

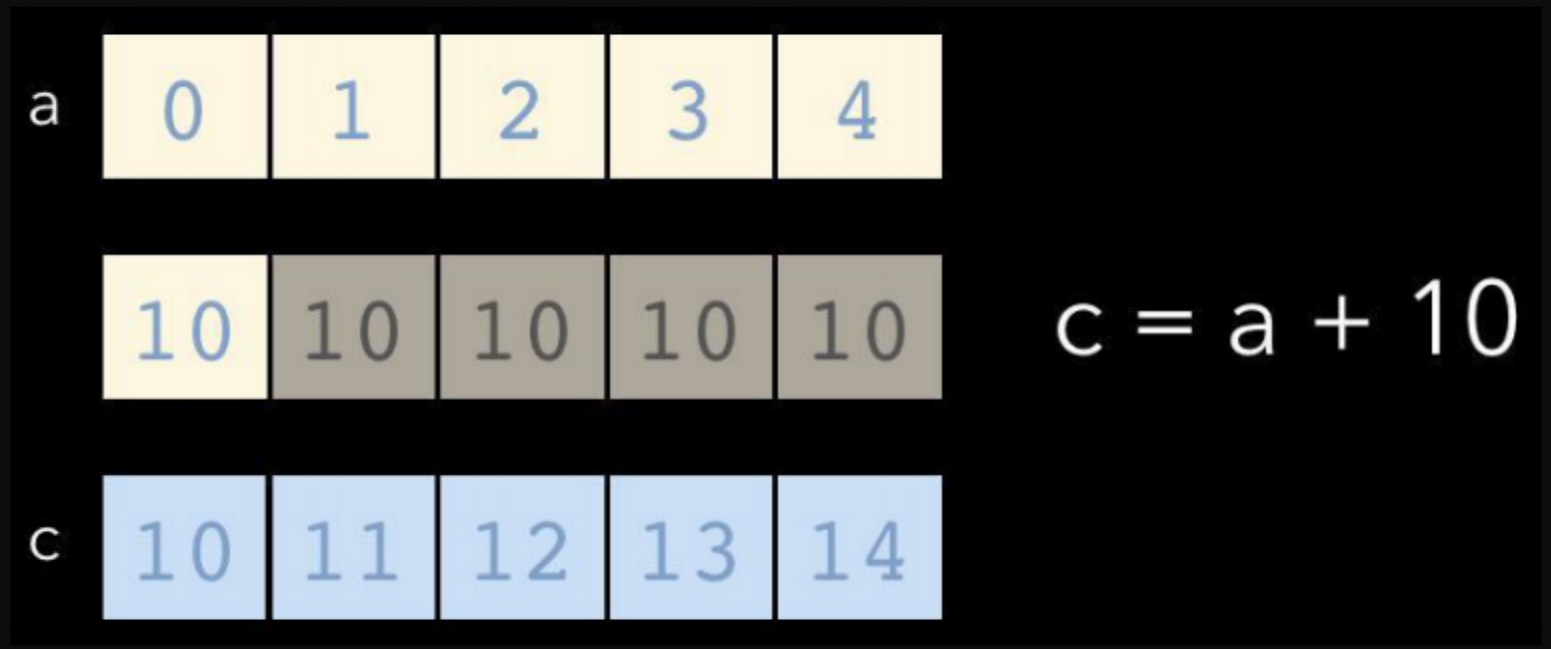
Array broadcasting with scalars

This also allows us to add a constant to a matrix or multiply a matrix by a constant

```
A = np.ones((3,3))

print 3 * A - 1
# [[ 2.  2.  2.]
#  [ 2.  2.  2.]
#  [ 2.  2.  2.]]
```

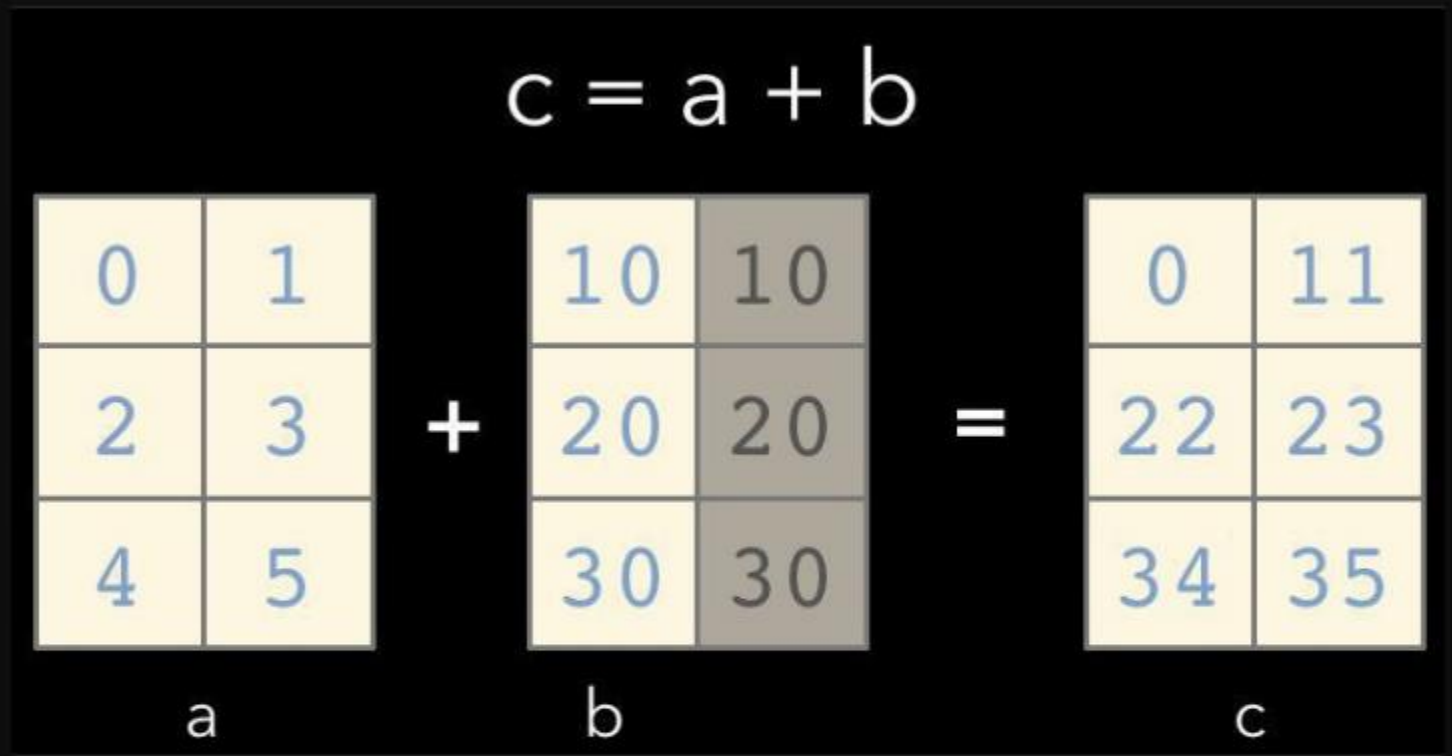
Broadcasting

A key feature of NumPy is broadcasting, where arrays with different, but compatible shapes can be used as arguments to ufuncs



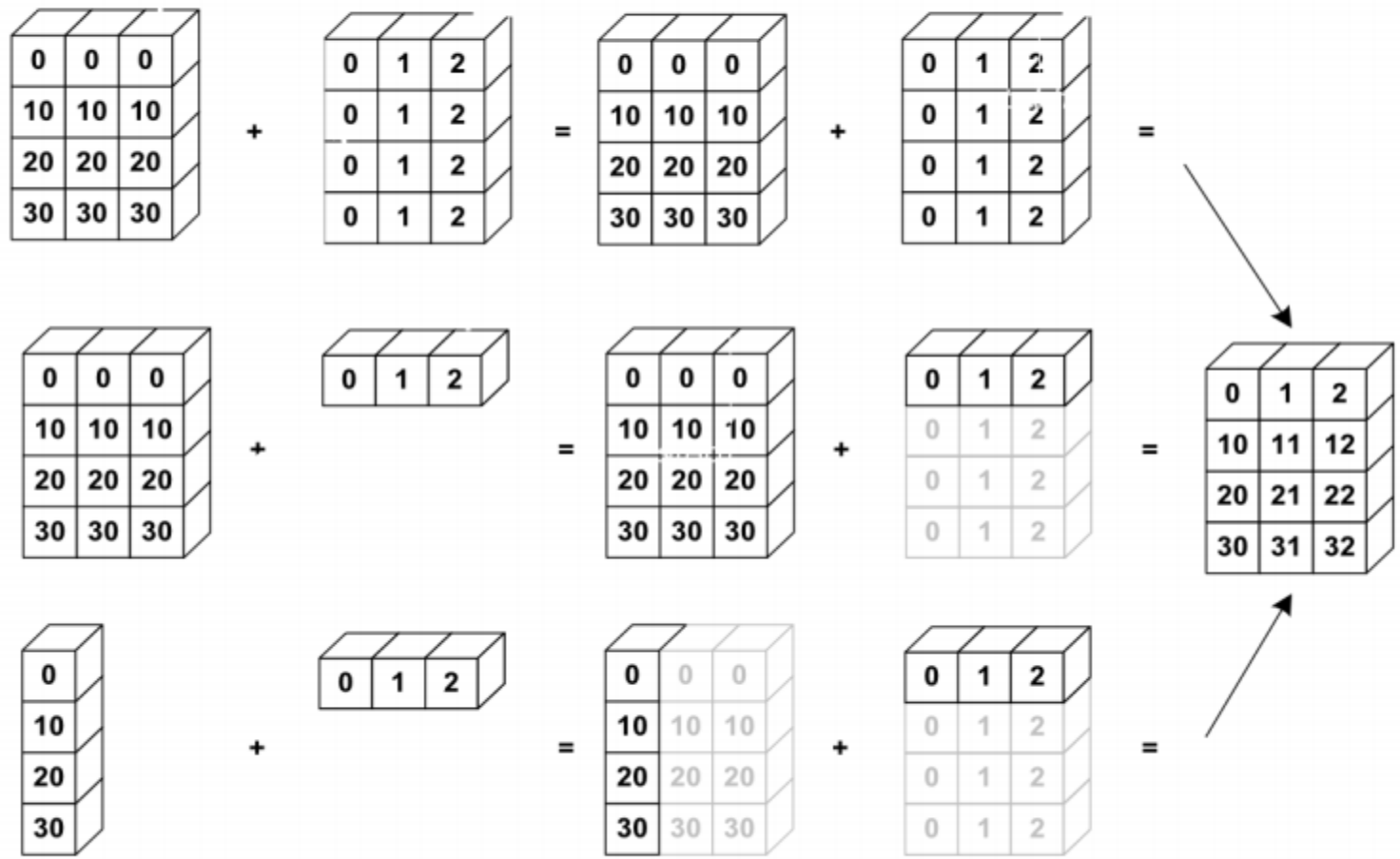
In this case an array scalar is broadcast to an array with shape `(5, )`

A slightly more involved broadcasting example in two dimensions



Here an array of shape `(3, 1)` is broadcast to an array with shape `(3, 2)`

Array broadcasting



Broadcasting Rules

In order for an operation to broadcast, the size of all the trailing dimensions for both arrays must either:

be **equal** OR be **one**

```
A      (1d array):             3
B      (2d array):         2 x 3
Result (2d array):         2 x 3

A      (2d array):         6 x 1
B      (3d array):     1 x 6 x 4
Result (3d array):     1 x 6 x 4

A      (4d array):  3 x 1 x 6 x 1
B      (3d array):      2 x 1 x 4
Result (4d array):  3 x 2 x 6 x 4
```

Vector operations

- inner product
- outer product
- dot product (matrix multiplication)

```
# note: numpy automatically converts lists
u = [1, 2, 3]
v = [1, 1, 1]

np.inner(u, v)
# 6
np.outer(u, v)
# array([[1, 1, 1],
#        [2, 2, 2],
#        [3, 3, 3]])
np.dot(u, v)
# 6
```

Matrix operations

First, define some matrices:

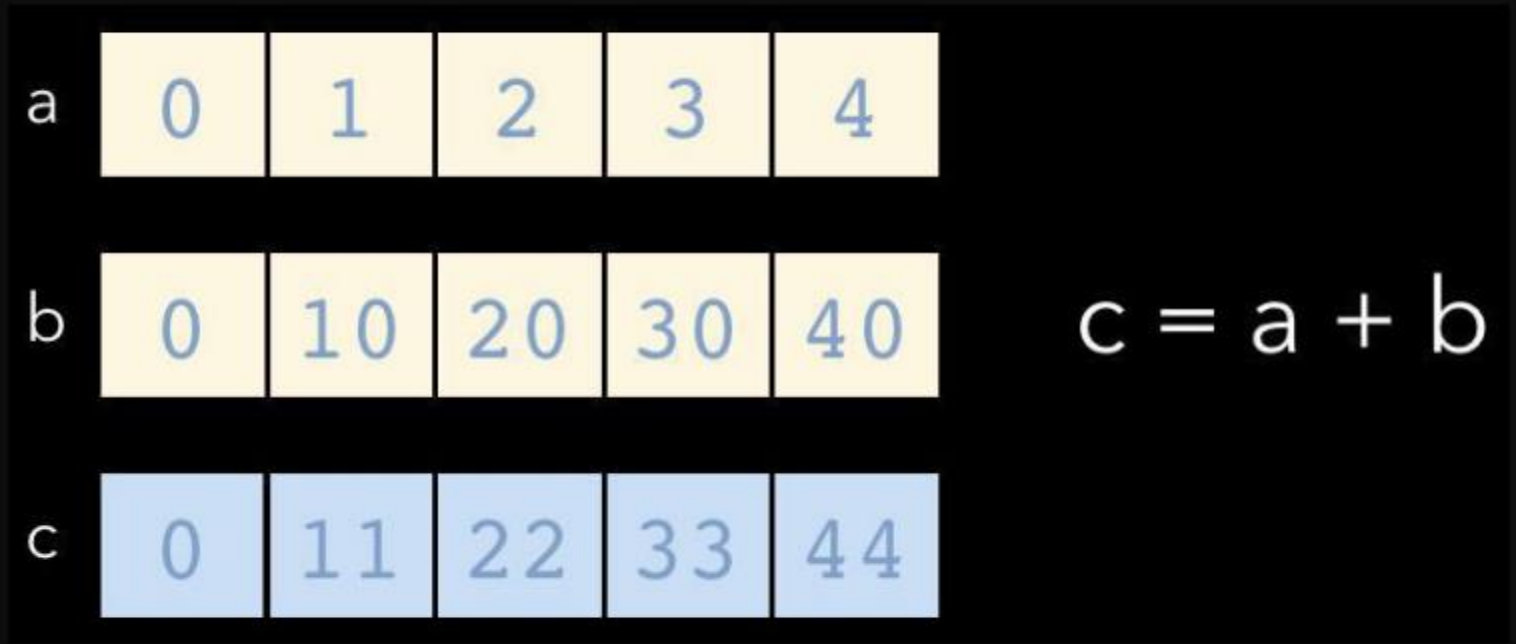
```
A = np.ones((3, 2))
# array([[ 1.,  1.],
#        [ 1.,  1.],
#        [ 1.,  1.]])
A.T
# array([[ 1.,  1.,  1.],
#        [ 1.,  1.,  1.]])

B = np.ones((2, 3))
# array([[ 1.,  1.,  1.],
#        [ 1.,  1.,  1.]])
```

Matrix operations

```
np.dot(A, B)
# array([[ 2.,  2.,  2.],
#        [ 2.,  2.,  2.],
#        [ 2.,  2.,  2.]])

np.dot(B, A)
# array([[ 3.,  3.],
#        [ 3.,  3.]])

np.dot(B.T, A.T)
# array([[ 2.,  2.,  2.],
#        [ 2.,  2.,  2.],
#        [ 2.,  2.,  2.]])

np.dot(A, B.T)
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# ValueError: shapes (3,2) and (3,2) not aligned:  ...
# ... 2 (dim 1) != 3 (dim 0)
```

Universal Functions (ufuncs)

NumPy ufuncs are functions that operate element-wise on one or more arrays



ufuncs dispatch to optimized C inner-loops based on array dtype

Mathematic Binary Operators

```
a + b →  add(a,b)
a - b →  subtract(a,b)
a % b →  remainder(a,b)
```

```
a * b  →  multiply(a,b)
a / b  →  divide(a,b)
a ** b →  power(a,b)
```

MULTIPLY BY A SCALAR

```
>>> a = array((1,2))
>>> a*3.
array([3., 6.])
```

ELEMENT BY ELEMENT ADDITION

```
>>> a = array([1,2])
>>> b = array([3,4])
>>> a + b
array([4, 6])
```

ADDITION USING AN OPERATOR FUNCTION

```
>>> add(a,b)
array([4, 6])
```

IN PLACE OPERATION

```
# Overwrite contents of a.
# Saves array creation
# overhead
>>> add(a,b,a) # a += b
array([4, 6])
>>> a
array([4, 6])
```

Comparison and Logical Operators

equal (==)	not_equal (!=)	greater (>)
greater_equal (>=)	less (<)	less_equal (<=)
logical_and	logical_or	logical_xor
logical_not		

2D EXAMPLE

```
>>> a = array(((1,2,3,4),(2,3,4,5)))
>>> b = array(((1,2,5,4),(1,3,4,5)))
>>> a == b
array([[True, True, False, True],
       [False, True, True, True]])
# functional equivalent
>>> equal(a,b)
array([[True, True, False, True],
       [False, True, True, True]])
```

Bitwise Operators

bitwise_and (&)	invert (~)	right_shift(a,shifts)
bitwise_or (|)	bitwise_xor	left_shift (a,shifts)

BITWISE EXAMPLES

```
>>> a = array((1,2,4,8))
>>> b = array((16,32,64,128))
>>> bitwise_or(a,b)
array([ 17,  34,  68, 136])

# bit inversion
>>> a = array((1,2,3,4), uint8)
>>> invert(a)
array([254, 253, 252, 251], dtype=uint8)

# left shift operation
>>> left_shift(a,3)
array([ 8, 16, 24, 32], dtype=uint8)
```

Trig and Other Functions

TRIGONOMETRIC

```
sin(x)      sinh(x)
cos(x)      cosh(x)
arccos(x)   arccosh(x)

arctan(x)   arctanh(x)
arcsin(x)   arcsinh(x)
arctan2(x,y)
```

OTHERS

```
exp(x)          log(x)
log10(x)        sqrt(x)
absolute(x)     conjugate(x)
negative(x)     ceil(x)
floor(x)        fabs(x)
hypot(x,y)      fmod(x,y)
maximum(x,y)    minimum(x,y)
```

`hypot(x,y)`

Element by element distance calculation using $\sqrt{x^2 + y^2}$

NumPy has many built-in ufuncs

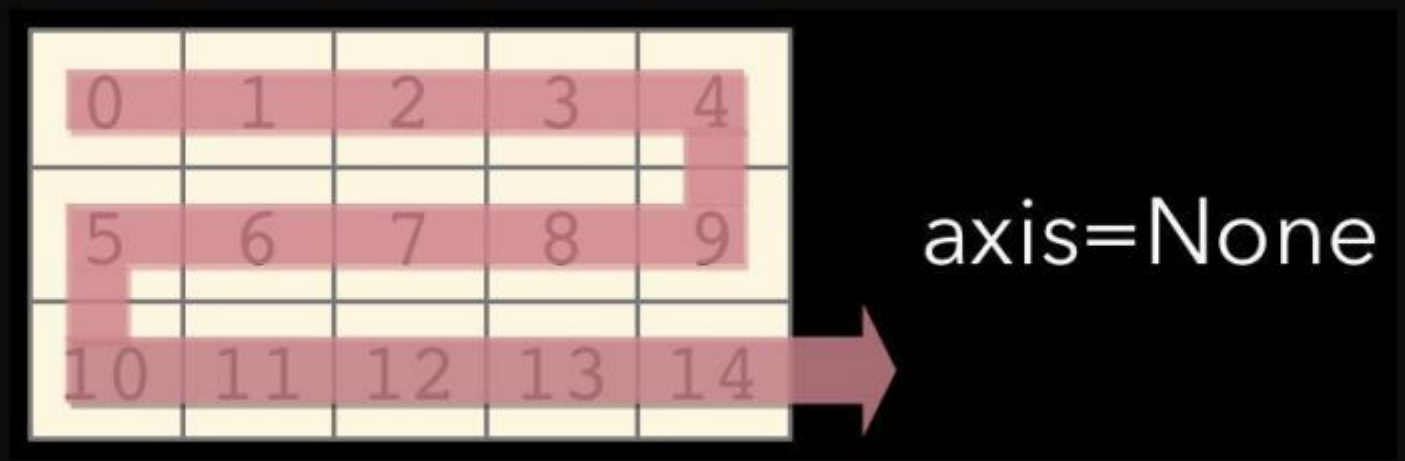
- comparison: `<`, `<=`, `==`, `!=`, `>=`, `>`
- arithmetic: `+`, `-`, `*`, `/`, `reciprocal`, `square`
- exponential: `exp`, `expm1`, `exp2`, `log`, `log10`, `log1p`, `log2`, `power`, `sqrt`
- trigonometric: `sin`, `cos`, `tan`, `acsin`, `arccos`, `atctan`
- hyperbolic: `sinh`, `cosh`, `tanh`, `acsinh`, `arccosh`, `atctanh`
- bitwise operations: `&`, `|`, `~`, `^`, `left_shift`, `right_shift`
- logical operations: `and`, `logical_xor`, `not`, `or`
- predicates: `isfinite`, `isinf`, `isnan`, `signbit`
- other: `abs`, `ceil`, `floor`, `mod`, `modf`, `round`, `sinc`, `sign`, `trunc`

Axis

Array method reductions take an optional `axis` parameter that specifies over which axes to reduce

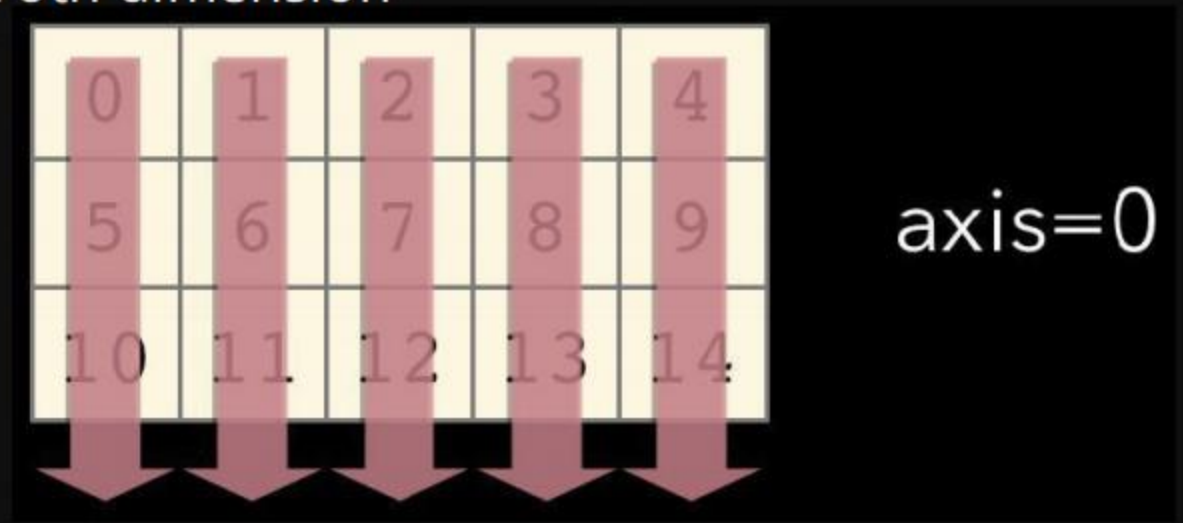
`axis=None` reduces into a single scalar

```
In [7]: a.sum
()
Out[7]: 105
```



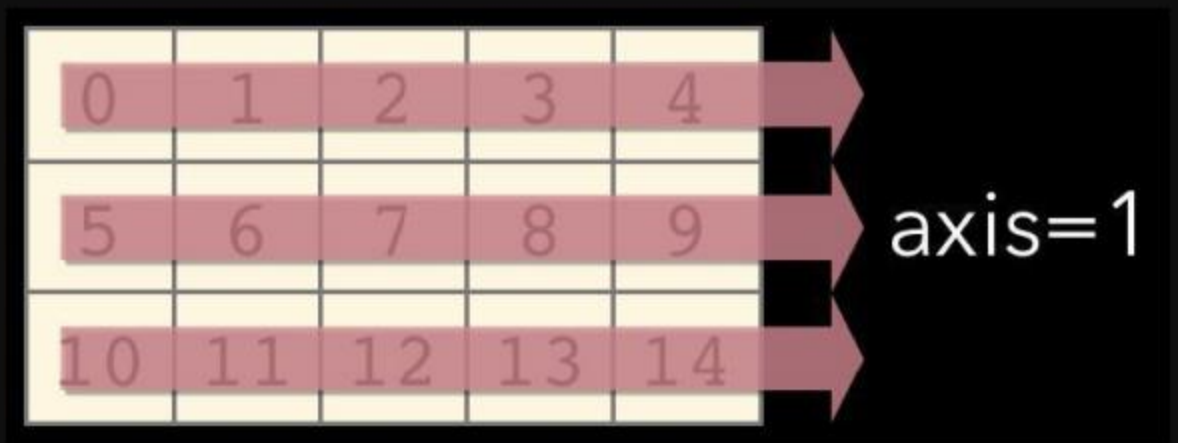
`axis=0` reduces into the zeroth dimension

```
In [8]: a.sum(axis=0)
Out[8]: array([15, 18, 21, 24,
27])
```



`axis=1` reduces into the first dimension

```
In [9]: a.sum(axis=1)
Out[9]: array([10,35,60])
```



Fancy Indexing

NumPy arrays may be used to index into other arrays

```
In [2]: a = np.arange(15).reshape((3,5))

In [3]: a
Out[3]:
array([[ 0,  1,  2,  3,  4],
[ 5,  6,  7,  8,  9],
[10, 11, 12, 13, 14]])
In [4]: i = np.array([[0,1], [1, 2]])

In [5]: j = np.array([[2, 1], [4, 4]])

In [6]: a[i,j]
Out[6]:
array([[ 2,  6],
[ 9, 14]])
```

Boolean arrays can also be used as indices into other arrays

```
In [2]: a = np.arange(15).reshape((3,5))

In [3]: a
Out[3]:
array([[ 0,  1,  2,  3,  4],
[ 5,  6,  7,  8,  9],
[10, 11, 12, 13, 14]])

In [4]: b = (a % 3 == 0)

In [5]: b
Out[5]:
array([[ True, False, False,  True, False],
[False,  True, False, False,  True],
[False, False,  True, False, False]], dtype=bool)

In [6]: a[b]
Out[6]: array([ 0,  3,  6,  9, 12])
```

NumPy Functions

- Data I/O
  - `fromfile, genfromtxt, load, loadtxt, save, savetxt`
- Mesh Creation
  - `mgrid, meshgrid, ogrid`
- Manipulation
  - `einsum, hstack, take, vstack`

- `numpy.fft` — Fast Fourier transforms
- `numpy.polynomial` — Efficient polynomials
- `numpy.linalg` — Linear algebra
  - `cholesky, det, eig, eigvals, inv, lstsq, norm, qr, svd`
- `numpy.math` — C standard library math functions
- `numpy.random` — Random number generation
  - `beta, gamma, geometric, hypergeometric, lognormal, normal, poisson, uniform, weibull`

loc: mean
scale: std. dev

```
np.random.random((2,3))
# array([[ 0.78084261,  0.64328818,  0.55380341],
#        [ 0.24611092,  0.37011213,  0.83313416]])

a = np.random.normal(loc=1.0, scale=2.0, size=(2,2))
# array([[ 2.87799514,  0.6284259 ],
#        [ 3.10683164,  2.05324587]])

np.savetxt("a_out.txt", a)
# save to file
b = np.loadtxt("a_out.txt")
# read from file
```

Operations along axes

```
a = np.random.random((2,3))
# array([[ 0.9190687 ,  0.36497813,  0.75644216],
#        [ 0.91938241,  0.08599547,  0.49544003]])
a.sum()
# 3.5413068994445549
a.sum(axis=0)  # column sum
# array([ 1.83845111,  0.4509736 ,  1.25188219])
a.cumsum()
# array([ 0.9190687 ,  1.28404683,  2.04048899,  2.9598714 ,
#         3.04586687,  3.5413069 ])
a.cumsum(axis=1)  # cumulative row sum
# array([[ 0.9190687 ,  1.28404683,  2.04048899],
#        [ 0.91938241,  1.00537788,  1.50081791]])
a.min()
# 0.085995469040367 7
a.max(axis=0)
# array([ 0.91938241,  0.36497813,  0.75644216])
```

Array Calculation Methods

SUM FUNCTION

```
>>> a = array([[1,2,3],
               [4,5,6]], float)

# Sum defaults to summing all
# *all* array values.
>>> sum(a)
21.

# supply the keyword axis to
# sum along the 0th axis.
>>> sum(a, axis=0)
array([5., 7., 9.])

# supply the keyword axis to
# sum along the last axis.
>>> sum(a, axis=-1)
array([6., 15.])
```

SUM ARRAY METHOD

```
# The a.sum() defaults to
# summing *all* array values
>>> a.sum()
21.

# Supply an axis argument to
# sum along a specific axis.
>>> a.sum(axis=0)
array([5., 7., 9.])
```

PRODUCT

```
# product along columns.
>>> a.prod(axis=0)
array([ 4., 10., 18.])

# functional form.
>>> prod(a, axis=0)
array([ 4., 10., 18.])
```

Min/Max

MIN

```
>>> a = array([2.,3.,0.,1.])
>>> a.min(axis=0)
0.
# use Numpy's amin() instead
# of Python's builtin min()
# for speed operations on
# multi-dimensional arrays.
>>> amin(a, axis=0)
0.
```

MAX

```
>>> a = array([2.,1.,0.,3.])
>>> a.max(axis=0)
3.

# functional form
>>> amax(a, axis=0)
3.
```

ARGMIN

```
# Find index of minimum value.
>>> a.argmin(axis=0)
2
# functional form
>>> argmin(a, axis=0)
2
```

ARGMAX

```
# Find index of maximum value.
>>> a.argmax(axis=0)
1
# functional form
>>> argmax(a, axis=0)
1
```

Statistics Array Methods

MEAN

```
>>> a = array([[1,2,3],
               [4,5,6]], float)

# mean value of each column
>>> a.mean(axis=0)
array([ 2.5,  3.5,  4.5])
>>> mean(a, axis=0)
array([ 2.5,  3.5,  4.5])
>>> average(a, axis=0)
array([ 2.5,  3.5,  4.5])

# average can also calculate
# a weighted average
>>> average(a, weights=[1,2],
...         axis=0)
array([ 3.,  4.,  5.])
```

STANDARD DEV./VARIANCE

```
# Standard Deviation
>>> a.std(axis=0)
array([ 1.5,  1.5,  1.5])

# Variance
>>> a.var(axis=0)
array([2.25, 2.25, 2.25])
>>> var(a, axis=0)
array([2.25, 2.25, 2.25])
```

Other Array Methods

CLIP

```
# Limit values to a range

>>> a = array([[1,2,3],
               [4,5,6]], float)

# Set values < 3 equal to 3.
# Set values > 5 equal to 5.
>>> a.clip(3,5)
>>> a
array([[ 3.,  3.,  3.],
       [ 4.,  5.,  5.]])
```

ROUND

```
# Round values in an array.
# Numpy rounds to even, so
# 1.5 and 2.5 both round to 2.
>>> a = array([1.35, 2.5, 1.5])
>>> a.round()
array([ 1.,  2.,  2.])

# Round to first decimal place.
>>> a.round(decimals=1)
array([ 1.4,  2.5,  1.5])
```

POINT TO POINT

```
# Calculate max - min for
# array along columns
>>> a.ptp(axis=0)
array([ 3.0,  3.0,  3.0])
# max - min for entire array.
>>> a.ptp(axis=None)
5.0
```

Summary of (most) array attributes/methods

BASIC ATTRIBUTES

```
a.dtype - Numerical type of array elements. float32, uint8, etc.
a.shape - Shape of the array. (m,n,o,...)
a.size - Number of elements in entire array.
a.itemsize - Number of bytes used by a single element in the array.
a.nbytes - Number of bytes used by entire array (data only).
a.ndim - Number of dimensions in the array.
```

SHAPE OPERATIONS

```
a.flat - An iterator to step through array as if it is 1D.
a.flatten() - Returns a 1D copy of a multi-dimensional array.
a.ravel() - Same as flatten(), but returns a 'view' if possible.
a.resize(new_size) - Change the size/shape of an array in-place.
a.swapaxes(axis1, axis2) - Swap the order of two axes in an array.
a.transpose(*axes) - Swap the order of any number of array axes.
a.T - Shorthand for a.transpose()
a.squeeze() - Remove any length=1 dimensions from an array.
```

Summary of (most) array attributes/methods

FILL AND COPY

```
a.copy() – Return a copy of the array.
a.fill(value) – Fill array with a scalar value.
```

CONVERSION / COERSION

```
a.tolist() – Convert array into nested lists of values.
a.tostring() – raw copy of array memory into a python string.
a.astype(dtype) – Return array coerced to given dtype.
a.byteswap(False) – Convert byte order (big <-> little endian).
```

COMPLEX NUMBERS

```
a.real – Return the real part of the array.
a.imag – Return the imaginary part of the array.
a.conjugate() – Return the complex conjugate of the array.
a.conj()– Return the complex conjugate of an array.(same as conjugate)
```

Summary of (most) array attributes/methods

SAVING

```
a.dump(file) - Store a binary array data out to the given file.
a.dumps() - returns the binary pickle of the array as a string.
a.tofile(fid, sep="", format="%s") Formatted ascii output to file.
```

SEARCH / SORT

```
a.nonzero() - Return indices for all non-zero elements in a.
a.sort(axis=-1) - Inplace sort of array elements along axis.
a.argsort(axis=-1) - Return indices for element sort order along axis.
a.searchsorted(b) - Return index where elements from b would go in a.
```

ELEMENT MATH OPERATIONS

```
a.clip(low, high) - Limit values in array to the specified range.
a.round(decimals=0) - Round to the specified number of digits.
a.cumsum(axis=None) - Cumulative sum of elements along axis.
a.cumprod(axis=None) - Cumulative product of elements along axis.
```

Summary of (most) array attributes/methods

REDUCTION METHODS

All the following methods "reduce" the size of the array by 1 dimension by carrying out an operation along the specified axis. If axis is None, the operation is carried out across the entire array.

```
a.sum(axis=None) - Sum up values along axis.
a.prod(axis=None) - Find the product of all values along axis.
a.min(axis=None)- Find the minimum value along axis.
a.max(axis=None) - Find the maximum value along axis.
a.argmin(axis=None) - Find the index of the minimum value along axis.
a.argmax(axis=None) - Find the index of the maximum value along axis.
a.ptp(axis=None) - Calculate a.max(axis) - a.min(axis)
a.mean(axis=None) - Find the mean (average) value along axis.
a.std(axis=None) - Find the standard deviation along axis.
a.var(axis=None) - Find the variance along axis.

a.any(axis=None) - True if any value along axis is non-zero.   (or)
a.all(axis=None) - True if all values along axis are non-zero. (and)
```

Array Operations

SIMPLE ARRAY MATH

```
>>> a = array([1,2,3,4])
>>> b = array([2,3,4,5])
>>> a + b
array([3, 5, 7, 9])
```

Numpy defines the following constants:

pi = 3.14159265359

e = 2.71828182846

MATH FUNCTIONS

```
# Create array from 0 to 10
>>> x = arange(11.)

# multiply entire array by
# scalar value
>>> a = (2*pi)/10.
>>> a
0.6283185307179586
>>> a*x
array([ 0.,0.628,…,6.283])

# inplace operations
>>> x *= a
>>> x
array([ 0.,0.628,…,6.283])

# apply functions to array.
>>> y = sin(x)
```

Recommendations

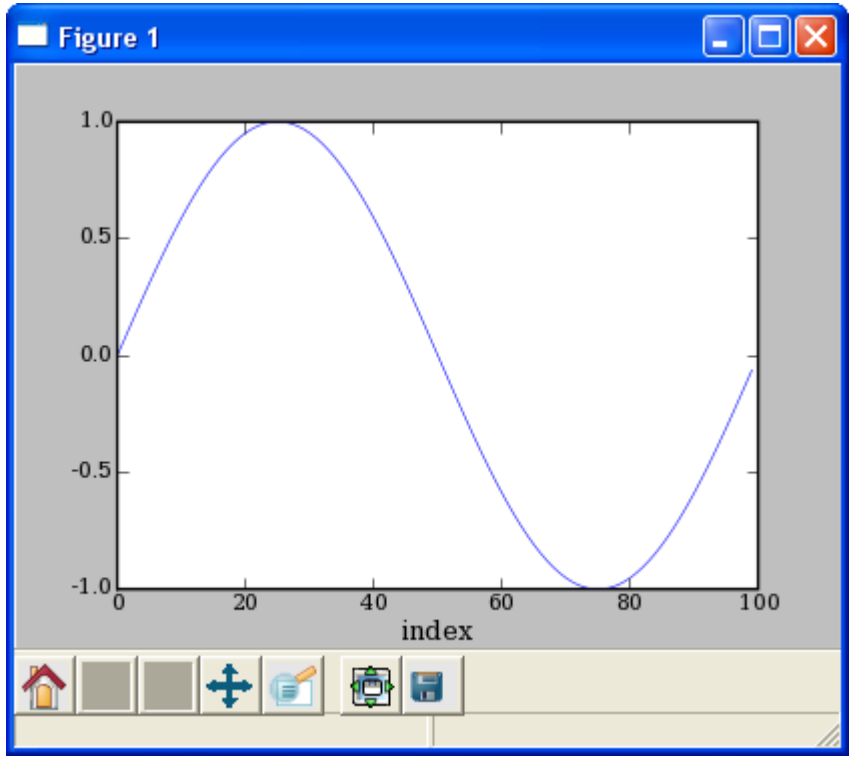
- Matplotlib for day-to-day data exploration.

  Matplotlib has a large community, tons of plot types, and is well integrated into ipython. It is the de-facto standard for ‘command line’ plotting from ipython.

Line Plots

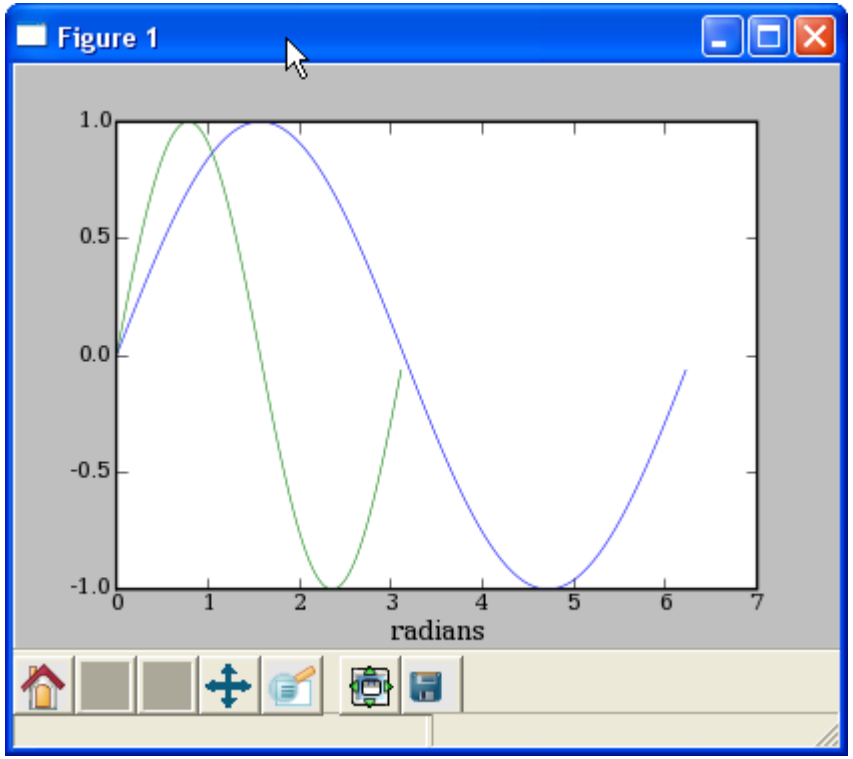
PLOT AGAINST INDICES

```
>>> x = arange(50)*2*pi/50.
>>> y = sin(x)
>>> mpl.pyplot.plot(y)
>>> mpl.pyplot.xlabel('index')
```



MULTIPLE DATA SETS

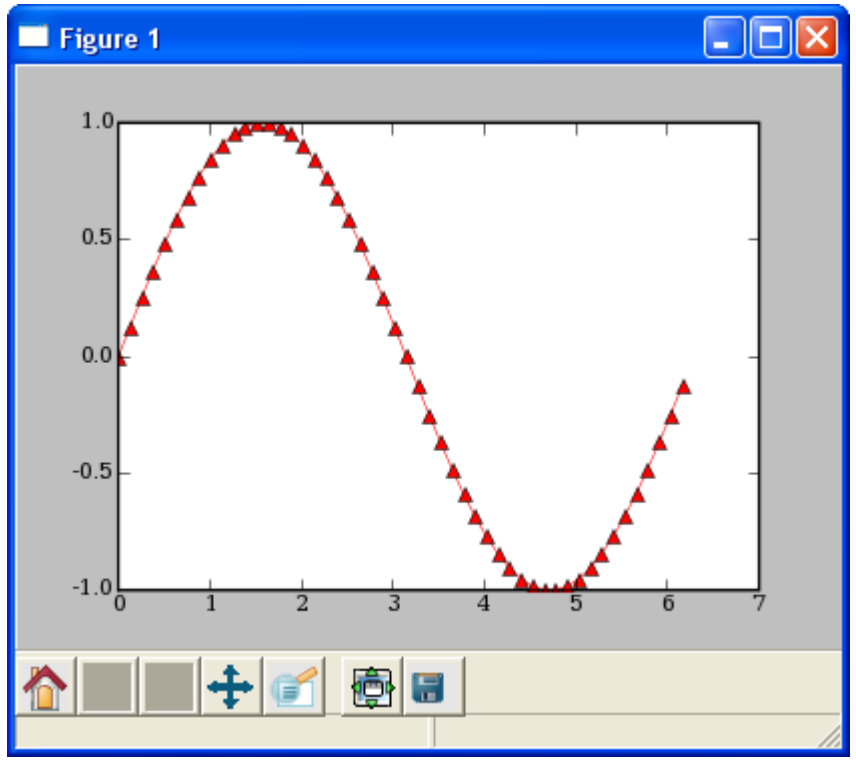
```
>>> plot(x,y,x2,y2)
>>> mpl.pyplot.xlabel('radians')
```



Line Plots

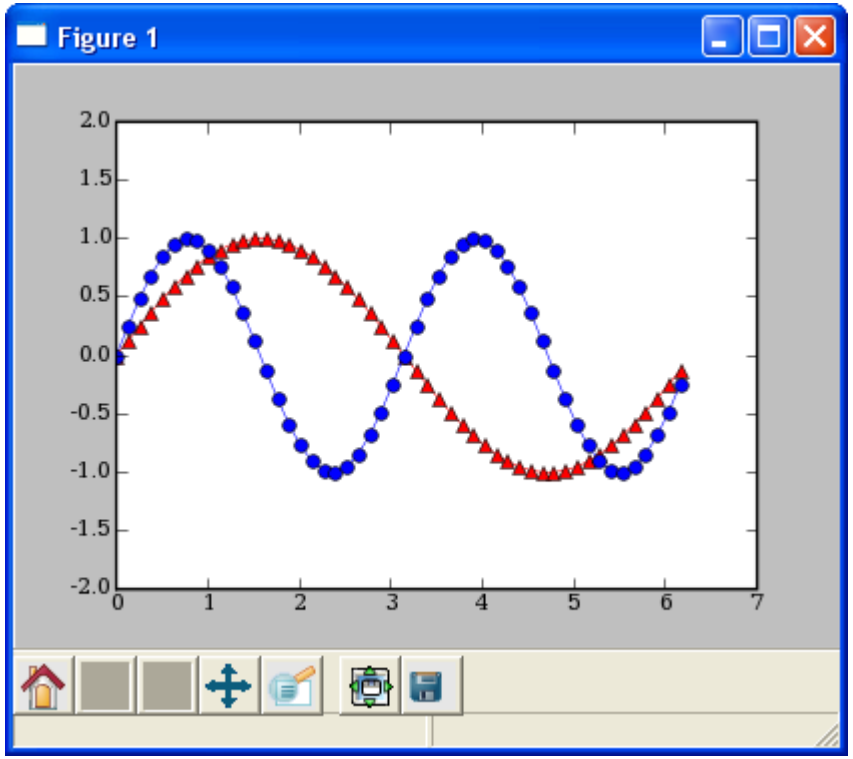
LINE FORMATTING

```
# red, dot-dash, triangles
>>> plot(x,sin(x),'r-^')
```



MULTIPLE PLOT GROUPS

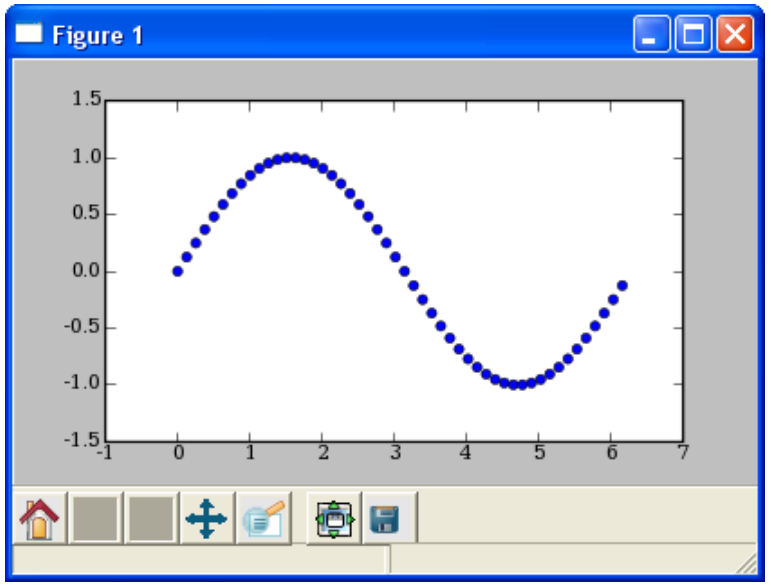
```
>>> plot(x,y1,'b-o', x,y2), r-^')
>>> axis([0,7,-2,2])
```



Scatter Plots

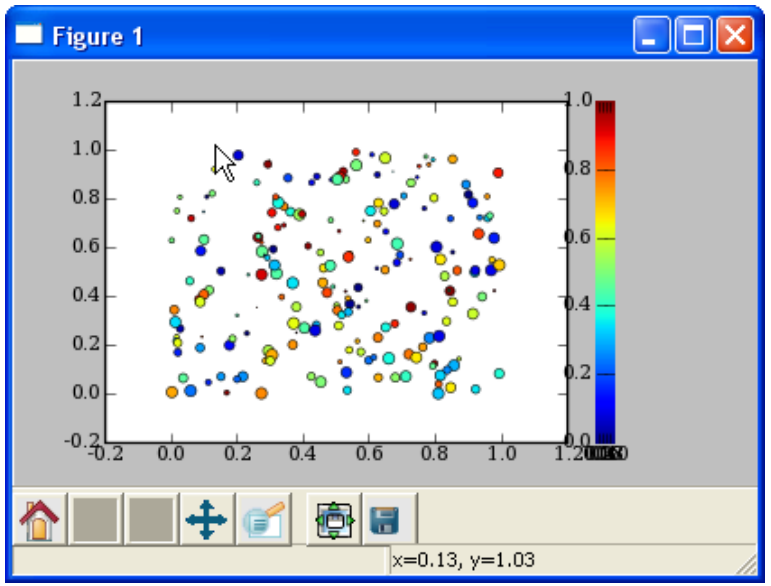
SIMPLE SCATTER PLOT

```
>>> x = arange(50)*2*pi/50.
>>> y = sin(x)
>>> pyplot.scatter(x,y)
```



COLORMAPPED SCATTER

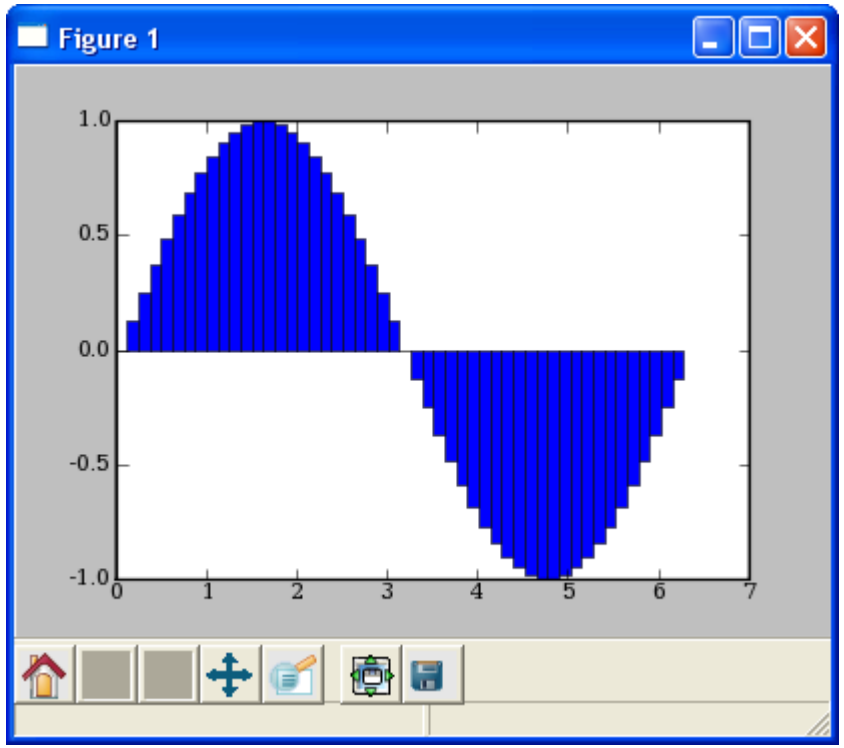
```
# marker size/color set with data
>>> x = rand(200)
>>> y = rand(200)
>>> size = rand(200)*30
>>> color = rand(200)
>>> mpl.pyplot.scatter(x, y, size,
color)
>>> colorbar()
```



Bar Plots

BAR PLOT

```
>>> pyplot.bar(x,sin(x),
...      width=x[1]-x[0])
```



SciPy

SciPy

- As the functions are less used these days, I leave this part for you to explore by yourself.

Course Outline

- Data management with Python (3 weeks)
  - Packages
    - Numpy/Scipy
    - Dataframe
    - Pandas
  - OOP
    - Class
  - Geolocation
  - Multiprocessing
  - Pickle, Json, HDF5 (.h5)
  - Web scraping (regular expression, selenium)

Reading data using pandas

```
In [ ]: #Read csv file
        df = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/Salaries.csv")
```

Note: The above command has many optional arguments to fine-tune the data import process.

There is a number of pandas commands to read other data formats:

```
pd.read_excel('myfile.xlsx',sheet_name='Sheet1', index_col=None, na_values=['NA'])
pd.read_stata('myfile.dta')
pd.read_sas('myfile.sas7bdat')
pd.read_hdf('myfile.h5','df')
```

Exploring data frames

In [3]:
```
#List first 5 records
df.head()
```

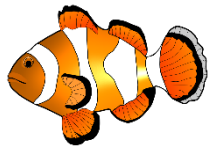
Out[3]:

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800

Hands-on exercises

- ✓ Try to read the first 10, 20, 50 records;
- ✓ Can you guess how to view the last few records;

Hint:



Data Frame data types

Pandas Type	Native Python Type	Description
object	string	The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).
int64	int	Numeric characters. 64 refers to the memory allocated to hold this character.
float64	float	Numeric characters with decimals. If a column contains numbers and NaNs(see below), pandas will default to float64, in case your missing value has a decimal.
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	Values meant to hold time data. Look into these for time series experiments.

Data Frame data types

```
In [4]: #Check a particular column type
        df['salary'].dtype
```

```
Out[4]: dtype('int64')
```

```
In [5]: #Check types for all the columns
        df.dtypes
```

```
Out[4]: rank          object
        discipline    object
        phd           int64
        service       int64
        sex           object
        salary        int64
        dtype: object
```

Data Frames attributes

Python objects have *attributes* and *methods*.

df.attribute	description
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

Hands-on exercises

- ✓ Find how many records this data frame has;
- ✓ How many elements are there?
- ✓ What are the column names?
- ✓ What types of columns we have in this data frame?

Data Frames methods

Unlike attributes, python methods have *parenthesis.*
All attributes and methods can be listed with a *dir()* function: `dir(df)`

df.method()	description
head([n]), tail([n])	first/last n rows
describe()	generate descriptive statistics (for numeric columns only)
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns
std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

Hands-on exercises

- ✓ Give the summary for the numeric columns in the dataset
- ✓ Calculate standard deviation for all numeric columns;
- ✓ What are the mean values of the first 50 records in the dataset? ***Hint:*** use head() method to subset the first 50 records and then calculate the mean

Selecting a column in a Data Frame

Method 1: Subset the data frame using column name:

```
df['sex']
```

Method 2: Use the column name as an attribute:

```
df.sex
```

Note: there is an attribute *rank* for pandas data frames, so to select a column with a name "rank" we should use method 1.

Hands-on exercises

- ✓ Calculate the basic statistics for the *salary* column;
- ✓ Find how many values in the *salary* column (use *count* method);
- ✓ Calculate the average salary;

groupby

Data Frames *groupby* method

Using "group by" method we can:

- Split the data into groups based on some criteria
- Calculate statistics (or apply a function) to each group
- Similar to dplyr() function in R

In [ ]:
```
#Group data using rank
df_rank = df.groupby(['rank'])
```

In [ ]:
```
#Calculate mean value for each numeric column per each group
df_rank.mean()
```

rank	phd	service	salary
AssocProf	15.076923	11.307692	91786.230769
AsstProf	5.052632	2.210526	81362.789474
Prof	27.065217	21.413043	123624.804348

Data Frames *groupby* method

Once groupby object is create we can calculate various statistics for each group:

```
In [ ]: #Calculate mean salary for each professor rank:
        df.groupby('rank')[['salary']].mean()
```

rank	salary
AssocProf	91786.230769
AsstProf	81362.789474
Prof	123624.804348

Note: If single brackets are used to specify the column (e.g. salary), then the output is Pandas Series object. When double brackets are used the output is a Data Frame

Data Frames *groupby* method

groupby performance notes:

- no grouping/splitting occurs until it's needed. Creating the *groupby* object only verifies that you have passed a valid mapping
- by default the group keys are sorted during the *groupby* operation. You may want to pass sort=False for potential speedup:

In [ ]:
```
#Calculate mean salary for each professor rank:
df.groupby(['rank'], sort=False)[['salary']].mean()
```

filtering

Data Frame: filtering

To subset the data we can apply Boolean indexing. This indexing is commonly known as a filter. For example if we want to subset the rows in which the salary value is greater than $120K:

In [ ]:
```
#Calculate mean salary for each professor rank:
df_sub = df[ df['salary'] > 120000 ]
```

Any Boolean operator can be used to subset the data:

> greater; >= greater or equal;

< less; <= less or equal;

== equal; != not equal;

In [ ]:
```
#Select only those rows that contain female professors:
df_f = df[ df['sex'] == 'Female' ]
```

slicing

Data Frames: Slicing

There are a number of ways to subset the Data Frame:

- one or more columns
- one or more rows
- a subset of rows and columns

Rows and columns can be selected by their position or label

Data Frames: Slicing

When selecting one column, it is possible to use single set of brackets, but the resulting object will be a Series (not a DataFrame):

In [ ]:
```
#Select column salary:
df['salary']
```

When we need to select more than one column and/or make the output to be a DataFrame, we should use double brackets:

In [ ]:
```
#Select column salary:
df[['rank','salary']]
```

Data Frames: Selecting rows

If we need to select a range of rows, we can specify the range using ":"

In [ ]:
```
#Select rows by their position:
df[10:20]
```

Notice that the first row has a position 0, and the last value in the range is omitted:
So for 0:10 range the first 10 rows are returned with the positions starting with 0 and ending with 9

Data Frames: method loc

If we need to select a range of rows, using their labels we can use method loc:

In [ ]:
```
#Select rows by their labels:
df_sub.loc[10:20,['rank','sex','salary']]
```

Out[ ]:

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
19	Prof	Male	150500

Data Frames: method iloc

If we need to select a range of rows and/or columns, using their positions we can use method iloc:

In [ ]:
```
#Select rows by their labels:
df_sub.iloc[10:20,[0, 3, 4, 5]]
```

Out[ ]:

	rank	service	sex	salary
26	Prof	19	Male	148750
27	Prof	43	Male	155865
29	Prof	20	Male	123683
31	Prof	21	Male	155750
35	Prof	23	Male	126933
36	Prof	45	Male	146856
39	Prof	18	Female	129000
40	Prof	36	Female	137000
44	Prof	19	Female	151768
45	Prof	25	Female	140096

Data Frames: method iloc (summary)

```
df.iloc[0]  # First row of a data frame
df.iloc[i]  #(i+1)th row
df.iloc[-1] # Last row
```

```
df.iloc[:, 0]  # First column
df.iloc[:, -1] # Last column
```

```
df.iloc[0:7]        #First 7 rows
df.iloc[:, 0:2]     #First 2 columns
df.iloc[1:3, 0:2]   #Second through third rows and first 2 columns
df.iloc[[0,5], [1,3]]  #1st and 6th rows and 2nd and 4th columns
```

sorting

Data Frames: Sorting

We can sort the data by a value in the column. By default the sorting will occur in ascending order and a new data frame is return.

In [ ]:
```
# Create a new data frame from the original sorted by the column Salary
df_sorted = df.sort_values( by ='service')
df_sorted.head()
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
55	AsstProf	A	2	0	Female	72500
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000

Data Frames: Sorting

We can sort the data using 2 or more columns:

In [ ]:
```
df_sorted = df.sort_values( by =['service', 'salary'], ascending = [True, False])
df_sorted.head(10)
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
52	Prof	A	12	0	Female	105000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
55	AsstProf	A	2	0	Female	72500
57	AsstProf	A	3	1	Female	72500
28	AsstProf	B	7	2	Male	91300
42	AsstProf	B	4	2	Female	80225
68	AsstProf	A	4	2	Female	77500

dropna

Missing Values

Missing values are marked as NaN

In [ ]:
```
# Read a dataset with missing values
flights = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/flights.csv")
```

In [ ]:
```
# Select the rows that have at least one missing value
flights[flights.isnull().any(axis=1)].head()
```

Out[ ]:

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	flight	origin	dest	air_time	distance	hour	minute
330	2013	1	1	1807.0	29.0	2251.0	NaN	UA	N31412	1228	EWR	SAN	NaN	2425	18.0	7.0
403	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EHAA	791	LGA	DFW	NaN	1389	NaN	NaN
404	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EVAA	1925	LGA	MIA	NaN	1096	NaN	NaN
855	2013	1	2	2145.0	16.0	NaN	NaN	UA	N12221	1299	EWR	RSW	NaN	1068	21.0	45.0
858	2013	1	2	NaN	NaN	NaN	NaN	AA	NaN	133	JFK	LAX	NaN	2475	NaN	NaN

Missing Values

There are a number of methods to deal with missing values in the data frame:

df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

pandas.DataFrame.dropna

DataFrame.**dropna**(*axis=0*, *how='any'*, *thresh=None*, *subset=None*, *inplace=False*) [source]

Return object with labels on given axis omitted where alternately any or all of the data are missing

Parameters:

axis : *{0 or 'index', 1 or 'columns'}, or tuple/list thereof*

Pass tuple or list to drop on multiple axes

how : *{'any', 'all'}*

- any : if any NA values are present, drop that label
- all : if all values are NA, drop that label

thresh : *int, default None*

int value : require that many non-NA values

subset : *array-like*

Labels along other axis to consider, e.g. if you are dropping rows these would be a list of columns to include

inplace : *boolean, default False*

If True, do operation inplace and return None.

Returns:

dropped : *DataFrame*

```python
import pandas as pd
import numpy as np
df = pd.DataFrame([[np.nan, 2, np.nan, 0], [3, 4, np.nan, 1],
                   [np.nan, np.nan, np.nan, 5]],
                  columns=list('ABCD'))
'''
     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
2  NaN  NaN NaN  5
'''
df.dropna(axis=1, how='all')
#drop the column only if all column values are NaN
'''
     A    B  D
0  NaN  2.0  0
1  3.0  4.0  1
2  NaN  NaN  5
'''
df.dropna(axis=1, how='any')
#drop the column if at lease one NaN cell, so only the last column remains
df.dropna(thresh=2)
#Keep only the rows with at least 2 non-na values
'''
     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
'''
```

pandas.Series.dropna

`Series.`**`dropna`**(*axis=0*, *inplace=False*, ***kwargs*)

Return Series without null values

Returns:	**valid** *: Series* **inplace** *: boolean, default False* Do operation in place.

```
from pandas import Series
import numpy as np

s1 = Series([1, 2, np.NaN])
s = s1.dropna(axis=0, inplace=False)
'''
0    1
1    2
'''

#Remove null values from series
```

Missing Values

- When summing the data, missing values will be treated as zero
- If all values are missing, the sum will be equal to NaN
- cumsum() and cumprod() methods ignore missing values but preserve them in the resulting arrays
- Missing values in GroupBy method are excluded (just like in R)
- Many descriptive statistics methods have *skipna* option to control if missing data should be excluded . This value is set to *True* by default (unlike R)

aggregation

Aggregation Functions in Pandas

Aggregation - computing a summary statistic about each group, i.e.

- compute group sums or means
- compute group sizes/counts

Common aggregation functions:

min, max
count, sum, prod
mean, median, mode, mad
std, var

Aggregation Functions in Pandas

agg() method are useful when multiple statistics are computed per column:

In [ ]:
```
flights[['dep_delay','arr_delay']].agg(['min','mean','max'])
```

Out[ ]:

	dep_delay	arr_delay
min	-16.000000	-62.000000
mean	9.384302	2.298675
max	351.000000	389.000000

pandas.core.groupby.GroupBy.aggregate

`GroupBy.`**`aggregate`**(*func*, **args*, ***kwargs*) [source]

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Dec 30 15:10:21 2017
@author: Liu Jiali
"""
import pandas as pd
df = pd.DataFrame({'A': [1,1,1,2,3,3,4,5],'B': [1,1,1,3,4,4,5,5], 'C': [1,1,1,1,1,2,3,4], 'D': [5,6,7,8,9,1,1,1]})
'''data looks like:
   A  B  C  D
0  1  1  1  5
1  1  1  1  6
2  1  1  1  7
3  2  3  1  8
4  3  4  1  9
5  3  4  2  1
6  4  5  3  1
7  5  5  4  1
'''

df.groupby(['A','B']).aggregate(['min','max'])
# output:
#       C       D
#     min max min max
# A B
# 1 1   1   1   5   7
# 2 3   1   1   8   8
# 3 4   1   2   1   9
# 4 5   3   3   1   1
# 5 5   4   4   1   1
```

Basic Descriptive Statistics

df.method()	description
describe	Basic statistics (count, mean, std, min, quantiles, max)
min, max	Minimum and maximum values
mean, median, mode	Arithmetic average, median and mode
var, std	Variance and standard deviation
sem	Standard error of mean
skew	Sample skewness
kurt	kurtosis

pd.dataframe.fillna

pandas.DataFrame.fillna

DataFrame.**fillna**(*value=None*, *method=None*, *axis=None*, *inplace=False*, *limit=None*, *downcast=None*, ***kwargs*)

[source]

Fill NA/NaN values using the specified method

Parameters:

value *: scalar, dict, Series, or DataFrame*

Value to use to fill holes (e.g. 0), alternately a dict/Series/DataFrame of values specifying which value to use for each index (for a Series) or column (for a DataFrame). (values not in the dict/Series/DataFrame will not be filled). This value cannot be a list.

method *: {'backfill', 'bfill', 'pad', 'ffill', None}, default None*

Method to use for filling holes in reindexed Series pad / ffill: propagate last valid observation forward to next valid backfill / bfill: use NEXT valid observation to fill gap

axis *: {0 or 'index', 1 or 'columns'}*

inplace *: boolean, default False*

If True, fill in place. Note: this will modify any other views on this object, (e.g. a no-copy slice for a column in a DataFrame).

limit *: int, default None*

If method is specified, this is the maximum number of consecutive NaN values to forward/backward fill. In other words, if there is a gap with more than this number of consecutive NaNs, it will only be partially filled. If method is not specified, this is the maximum number of entries along the entire axis where NaNs will be filled. Must be greater than 0 if not None.

downcast *: dict, default is None*

a dict of item->dtype of what to downcast if possible, or the string 'infer' which will try to downcast to an appropriate equal type (e.g. float64 to int64 if possible)

```python
import pandas as pd
import numpy as np
df = pd.DataFrame([[np.nan, 2, np.nan, 0],
                   [3, 4, np.nan, 1],
                   [np.nan, np.nan, np.nan, 5],
                   [np.nan, 3, np.nan, 4]],
                   columns=list('ABCD'))
df.fillna(0)
#Replace all NaN elements with 0s.
'''
    A   B   C   D
0   0.0 2.0 0.0 0
1   3.0 4.0 0.0 1
2   0.0 0.0 0.0 5
3   0.0 3.0 0.0 4
'''
df.fillna(method='ffill')
#propagate non-null values forward
'''
    A   B   C   D
0   NaN 2.0 NaN 0
1   3.0 4.0 NaN 1
2   3.0 4.0 NaN 5
3   3.0 3.0 NaN 4
'''
```

```
values = {'A': 0, 'B': 1, 'C': 2, 'D': 3}
df.fillna(value=values)
#Replace all NaN elements in column ‘A’, ‘B’, ‘C’, and ‘D’, with 0, 1, 2, and
#3 respectively.
'''
    A   B   C   D
0   0.0 2.0 2.0 0
1   3.0 4.0 2.0 1
2   0.0 1.0 2.0 5
3   0.0 3.0 2.0 4
'''
df.fillna(value=values, limit=1)
#Only replace the first NaN element.
'''
    A   B   C   D
0   0.0 2.0 2.0 0
1   3.0 4.0 NaN 1
2   NaN 1.0 NaN 5
3   NaN 3.0 NaN 4
'''
```

pd.dataframe.drop_duplicates

pandas.DataFrame.drop_duplicates

`DataFrame.drop_duplicates(subset=None, keep='first', inplace=False)` [source]

Return DataFrame with duplicate rows removed, optionally only considering certain columns

Parameters:

subset *: column label or sequence of labels, optional*

Only consider certain columns for identifying duplicates, by default use all of the columns

keep *: {'first', 'last', False}, default 'first'*

- `first` : Drop duplicates except for the first occurrence.
- `last` : Drop duplicates except for the last occurrence.
- False : Drop all duplicates.

inplace *: boolean, default False*

Whether to drop duplicates in place or to return a copy

Returns:

deduplicated *: DataFrame*

```
# -*- coding: utf-8 -*-
"""
Created on Set Dec 23 2017
@author: Luo Hankun (3035084914)
"""
import pandas as pd

df = pd.DataFrame({'A': [1,1,2,3,5,6,7,8],'B': [1,1,6,7,8,9,10,11]})

df.drop_duplicates()
'''
Output:
   A   B
0  1   1
2  2   6
3  3   7
4  5   8
5  6   9
6  7  10
7  8  11

Removed row 1 (duplicate)
'''
```

pd.dataframe.T

pandas.DataFrame.T

`DataFrame.T`

Transpose index and columns

```python
import pandas as pd
df = pd.DataFrame({'A': {0: 'a', 1: 'b', 2: 'c'},
                   'B': {0: 1, 1: 3, 2: 5},
                   'C': {0: 2, 1: 4, 2: 6}})
'''
   A  B  C
0  a  1  2
1  b  3  4
2  c  5  6
'''
df.T
#Transpose index and columns
'''
   0  1  2
A  a  b  c
B  1  3  5
C  2  4  6
'''
```

pd.unique

pandas.unique

`pandas.unique(values)` [source]

Hash table-based unique. Uniques are returned in order of appearance. This does NOT sort.

Significantly faster than numpy.unique. Includes NA values.

Parameters:	**values** *: 1d array-like*
Returns:	*unique values.* • If the input is an Index, the return is an Index • If the input is a Categorical dtype, the return is a Categorical • If the input is a Series/ndarray, the return will be an ndarray

See also: `pandas.Index.unique`, `pandas.Series.unique`

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 21 17:26:38 2017

@author: DemiWu
"""

import pandas as pd
df = pd.DataFrame({'A': [1, 3, 1, 4, 3], 'B': [3, 1, 4, 2, 2]})

#find unique values in column A
pd.unique(df.A)
'''the result is
array([1, 3, 4])
'''
```

pd.to_datetime

pandas.to_datetime

pandas.**to_datetime**(*arg*, *errors='raise'*, *dayfirst=False*, *yearfirst=False*, *utc=None*, *box=True*, *format=None*, *exact=True*, *unit=None*, *infer_datetime_format=False*, *origin='unix'*, *cache=False*) [source]

Convert argument to datetime.

Parameters:

arg : *integer, float, string, datetime, list, tuple, 1-d array, Series*
New in version 0.18.1: or DataFrame/dict-like

errors : *{'ignore', 'raise', 'coerce'}, default 'raise'*

- If 'raise', then invalid parsing will raise an exception
- If 'coerce', then invalid parsing will be set as NaT
- If 'ignore', then invalid parsing will return the input

dayfirst : *boolean, default False*
Specify a date parse order if *arg* is str or its list-likes. If True, parses dates with the day first, eg 10/11/12 is parsed as 2012-11-10. Warning: dayfirst=True is not strict, but will prefer to parse with day first (this is a known bug, based on dateutil behavior).

yearfirst : *boolean, default False*
Specify a date parse order if *arg* is str or its list-likes.

- If True parses dates with the year first, eg 10/11/12 is parsed as 2010-11-12.
- If both dayfirst and yearfirst are True, yearfirst is preceded (same as dateutil).

Warning: yearfirst=True is not strict, but will prefer to parse with year first (this is a known bug, based on dateutil beahavior).
New in version 0.16.1.

utc : *boolean, default None*
Return UTC DatetimeIndex if True (converting any tz-aware datetime.datetime objects as well).

box : *boolean, default True*

- If True returns a DatetimeIndex
- If False returns ndarray of values.

format : *string, default None*
strftime to parse time, eg "%d/%m/%Y", note that "%f" will parse all the way up to nanoseconds.

exact : *boolean, True by default*

- If True, require an exact format match.
- If False, allow the format to match anywhere in the target string.

unit : *string, default 'ns'*

unit of the arg (D,s,ms,us,ns) denote the unit, which is an integer or float number. This will be based off the origin. Example, with unit='ms' and origin='unix' (the default), this would calculate the number of milliseconds to the unix epoch start.

infer_datetime_format : *boolean, default False*

If True and no *format* is given, attempt to infer the format of the datetime strings, and if it can be inferred, switch to a faster method of parsing them. In some cases this can increase the parsing speed by ~5-10x.

origin : *scalar, default is 'unix'*

Define the reference date. The numeric values would be parsed as number of units (defined by *unit*) since this reference date.

- If 'unix' (or POSIX) time; origin is set to 1970-01-01.
- If 'julian', unit must be 'D', and origin is set to beginning of Julian Calendar. Julian day number 0 is assigned to the day starting at noon on January 1, 4713 BC.
- If Timestamp convertible, origin is set to Timestamp identified by origin.

New in version 0.20.0.

cache : *boolean, default False*

If True, use a cache of unique, converted dates to apply the datetime conversion. May produce sigificant speed-up when parsing duplicate date strings, especially ones with timezone offsets.

New in version 0.23.0.

Returns:

ret : *datetime if parsing succeeded.*

Return type depends on input:

- list-like: DatetimeIndex
- Series: Series of datetime64 dtype
- scalar: Timestamp

In case when it is not possible to return designated types (e.g. when any element of input is before Timestamp.min or after Timestamp.max) return will have datetime.datetime type (or corresponding array/Series).

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 22 16:53:43 2017

@author: DemiWu
"""

import pandas as pd
dt = ['2017-01-05 2:30:00 PM', 'Jan 5, 2017 14:30:00', '01/05/2016',
      '2017.01.05', '2017/01/05','20170105']

#convert dt to datetime format
pd.to_datetime(dt)
'''the result is
DatetimeIndex(['2017-01-05 14:30:00', '2017-01-05 14:30:00',
               '2016-01-05 00:00:00', '2017-01-05 00:00:00',
               '2017-01-05 00:00:00', '2017-01-05 00:00:00'],
              dtype='datetime64[ns]', freq=None)
'''
```

```
#about "yearfirst" argument
pd.to_datetime('10/11/12')
'''the result is
Timestamp('2012-10-11 00:00:00')
'''
pd.to_datetime('10/11/12', yearfirst=True)
'''the result is
Timestamp('2010-11-12 00:00:00')
'''
#note that when yearfirst is set to true, the first number in the input is recognized as year

#custome datetime format of the input
pd.to_datetime('2017$01$05')
'''ValueError: Unknown string format'''
pd.to_datetime('2017$01$05', format='%Y$%m$%d')
'''the result is
Timestamp('2017-01-05 00:00:00')
'''
pd.to_datetime('2017/01/05', infer_datetime_format=True)
'''the result is Timestamp('2017-01-05 00:00:00')'''

#handle invalid dates
pd.to_datetime(['2017-01-05', 'Jan 6, 2017', 'abc'])
'''ValueError: Unknown string format'''
pd.to_datetime(['2017-01-05', 'Jan 6, 2017', 'abc'], errors='ignore')
'''the result is
array(['2017-01-05', 'Jan 6, 2017', 'abc'], dtype=object)
'''
#note that the input is returned as original
pd.to_datetime(['2017-01-05', 'Jan 6, 2017', 'abc'], errors='coerce')
'''the result is
DatetimeIndex(['2017-01-05', '2017-01-06', 'NaT'], dtype='datetime64[ns]', freq=None)
'''
#note that 'abc' is returned as 'NaT'
```

```
#Unix time: the time has passed since the origin
pd.to_datetime(1501324478, unit='s')
'''the result is Timestamp('2017-07-29 10:34:38')'''
pd.to_datetime(1501324478000, unit='ms')
'''the resutl is Timestamp('2017-07-29 10:34:38')'''
#change the origin of unix time
pd.to_datetime(2458072.5, unit='D', origin='julian')
'''the result is Timestamp('2017-11-15 00:00:00')'''

#use utc time
pd.to_datetime(dt, utc=True)
'''the result is
DatetimeIndex(['2017-01-05 14:30:00+00:00', '2017-01-05 14:30:00+00:00',
               '2016-01-05 00:00:00+00:00', '2017-01-05 00:00:00+00:00',
               '2017-01-05 00:00:00+00:00', '2017-01-05 00:00:00+00:00'],
              dtype='datetime64[ns, UTC]', freq=None)
'''

#change types of return
pd.to_datetime(dt, box=False)
'''the result is
array(['2017-01-05T14:30:00.000000000', '2017-01-05T14:30:00.000000000',
       '2016-01-05T00:00:00.000000000', '2017-01-05T00:00:00.000000000',
       '2017-01-05T00:00:00.000000000', '2017-01-05T00:00:00.000000000'], dtype='datetime64[ns]')
'''
#note that the return is array instead of datetimeindex
```

pd.dataframe.merge

pandas.DataFrame.merge

`DataFrame.`**merge**(*right*, *how='inner'*, *on=None*, *left_on=None*, *right_on=None*, *left_index=False*, *right_index=False*, *sort=False*, *suffixes=('_x', '_y')*, *copy=True*, *indicator=False*, *validate=None*) [source]

Merge DataFrame objects by performing a database-style join operation by columns or indexes.

If joining columns on columns, the DataFrame indexes *will be ignored*. Otherwise if joining indexes on indexes or indexes on a column or columns, the index will be passed on.

right : *DataFrame*

how : *{'left', 'right', 'outer', 'inner'}, default 'inner'*

- left: use only keys from left frame, similar to a SQL left outer join; preserve key order
- right: use only keys from right frame, similar to a SQL right outer join; preserve key order
- outer: use union of keys from both frames, similar to a SQL full outer join; sort keys lexicographically
- inner: use intersection of keys from both frames, similar to a SQL inner join; preserve the order of the left keys

on : *label or list*

Column or index level names to join on. These must be found in both DataFrames. If *on* is None and not merging on indexes then this defaults to the intersection of the columns in both DataFrames.

left_on : *label or list, or array-like*

Column or index level names to join on in the left DataFrame. Can also be an array or list of arrays of the length of the left DataFrame. These arrays are treated as if they are columns.

right_on : *label or list, or array-like*

Column or index level names to join on in the right DataFrame. Can also be an array or list of arrays of the length of the right DataFrame. These arrays are treated as if they are columns.

Parameters:	**left_index** : *boolean, default False* Use the index from the left DataFrame as the join key(s). If it is a MultiIndex, the number of keys in the other DataFrame (either the index or a number of columns) must match the number of levels **right_index** : *boolean, default False* Use the index from the right DataFrame as the join key. Same caveats as left_index **sort** : *boolean, default False* Sort the join keys lexicographically in the result DataFrame. If False, the order of the join keys depends on the join type (how keyword) **suffixes** : *2-length sequence (tuple, list, …)* Suffix to apply to overlapping column names in the left and right side, respectively **copy** : *boolean, default True* If False, do not copy data unnecessarily **indicator** : *boolean or string, default False* If True, adds a column to output DataFrame called “_merge” with information on the source of each row. If string, column with information on source of each row will be added to output DataFrame, and column will be named value of string. Information column is Categorical-type and takes on a value of “left_only” for observations whose merge key only appears in ‘left’ DataFrame, “right_only” for observations whose merge key only appears in ‘right’ DataFrame, and “both” if the observation’s merge key is found in both. **validate** : *string, default None* If specified, checks if merge is of specified type. • “one_to_one” or “1:1”: check if merge keys are unique in both left and right datasets. • “one_to_many” or “1:m”: check if merge keys are unique in left dataset. • “many_to_one” or “m:1”: check if merge keys are unique in right dataset. • “many_to_many” or “m:m”: allowed, but does not result in checks. *New in version 0.21.0.*
Returns:	**merged** : *DataFrame* The output type will the be same as ‘left’, if it is a subclass of DataFrame.

```python
import pandas as pd

caller = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3', 'K4', 'K5'],
                       'A': ['A0', 'A1', 'A2', 'A3', 'A4', 'A5']})
'''
    A key
0  A0  K0
1  A1  K1
2  A2  K2
3  A3  K3
4  A4  K4
5  A5  K5
'''
other = pd.DataFrame({'key': ['K0', 'K1', 'K2'],
                      'B': ['B0', 'B1', 'B2']})
'''
    B key
0  B0  K0
1  B1  K1
2  B2  K2
'''
caller.merge(other, left_on='key', right_on='key', how='inner')
#left_on and right_on: Field names to join
#inner only keeps the intersection by 'key'
'''
    A key   B
0  A0  K0  B0
1  A1  K1  B1
2  A2  K2  B2
'''
```

```
'''
how : {‘left’, ‘right’, ‘outer’, ‘inner’}, default ‘inner’
left: use only keys from left frame, similar to a SQL left outer join; preserve
      key order
right: use only keys from right frame, similar to a SQL right outer join;
       preserve key order
outer: use union of keys from both frames, similar to a SQL full outer join;
       sort keys lexicographically
inner: use intersection of keys from both frames, similar to a SQL inner join;
       preserve the order of the left keys
'''
caller.merge(other, on='key', how='outer',sort=1)
#on can be used when both df have the same key column
#outer join keeps the union by 'key'
#sort=1:  Sort the join keys lexicographically in the result DataFrame. If
#False, the order of the join keys depends on the join type
'''
    A key    B
0  A0  K0   B0
1  A1  K1   B1
2  A2  K2   B2
3  A3  K3  NaN
4  A4  K4  NaN
5  A5  K5  NaN
'''
caller.merge(other, on='key', how='right',sort=1)
#use only keys from right frame, similar to a SQL right outer join; preserve
#key order
'''
    A key   B
0  A0  K0  B0
1  A1  K1  B1
2  A2  K2  B2
'''
```

pd.concat

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 25 15:06:38 2017

@author: jessicaleung
"""

import pandas as pd

s1 = pd.Series(['a', 'b'])
s2 = pd.Series(['c', 'd'])

pd.concat([s1, s2])
# Output:
# 0    a
# 1    b
# 0    c
# 1    d
# dtype: object

pd.concat([s1, s2], axis=1)
# Output:
#    0  1
# 0  a  c
# 1  b  d

''' Set axis=1 to concatenate along columns. '''

pd.concat([s1, s2], ignore_index=True)
# Output:
# 0    a
# 1    b
# 2    c
# 3    d
# dtype: object

''' Set ignore_index=True to assign new index values. It is by default False to
use the index values along the concatenation axis. '''
```

```
40
41 pd.concat([s1, s2], keys=['s1', 's2'])
42 # Output:
43 # s1  0    a
44 #     1    b
45 # s2  0    c
46 #     1    d
47 # dtype: object
48
49 ''' Set keys to construct hierarchical index as the outermost level. '''
50
51 pd.concat([s1, s2], keys=['s1', 's2'], names=['Series name', 'Row ID'])
52 # Output:
53 # Series name  Row ID
54 # s1           0         a
55 #              1         b
56 # s2           0         c
57 #              1         d
58 # dtype: object
59
60 ''' Add Names for the levels in the resulting hierarchical index. '''
```

```
61
62 s3 = pd.Series([1,2,3], index=['a','b','c'])
63 '''
64 s3 is like:
65 a    1
66 b    2
67 c    3
68 dtype: int64
69 '''
70
71 s4 = pd.Series([5,6,7], index=['a','c','d'])
72 '''
73 s4 is like:
74 a    5
75 c    6
76 d    7
77 dtype: int64
78 '''
79
80 pd.concat([s3,s4], axis=1)
81 # Output:
82 #      0    1
83 # a  1.0  5.0
84 # b  2.0  NaN
85 # c  3.0  6.0
86 # d  NaN  7.0
87
88 pd.concat([s3,s4], axis=1, join_axes=[['a','b','c']])
89 # Output:
90 #    0    1
91 # a  1  5.0
92 # b  2  NaN
93 # c  3  6.0
94
95 ''' Specify indexes to use for the other n - 1 axes instead of performing inner/outer set logic. '''
96
```

```python
df1 = pd.DataFrame([['a', 1], ['b', 2]], columns=['letter', 'number'])
'''
df1 is like:
  letter  number
0      a       1
1      b       2
'''

df2 = pd.DataFrame([['c', 3], ['d', 4]], columns=['letter', 'number'])
'''
df2 is like:
  letter  number
0      c       3
1      d       4
'''

pd.concat([df1, df2])
# Output:
#   letter  number
# 0      a       1
# 1      b       2
# 0      c       3
# 1      d       4

pd.concat([df1, df2], ignore_index=True)
# Output:
#   letter  number
# 0      a       1
# 1      b       2
# 2      c       3
# 3      d       4

new_df=pd.concat([df1, df2], axis=1, keys=['lv1','lv2'], names=['upper', 'lower'], levels=[['lv1','lv2','lv3']])
# Output:
# upper    lv1           lv2
# lower letter number letter number
# 0          a      1      c      3
# 1          b      2      d      4

new_df.columns
# Output:
# MultiIndex(levels=[['lv1', 'lv2', 'lv3'], ['letter', 'number']],
#            labels=[[0, 0, 1, 1], [0, 1, 0, 1]],
#            names=['upper', 'lower'])

''' adding levels to the parameter specfifies levels to construct MultiIndex. '''
```

```
143
144 df3 = pd.DataFrame([['c', 3, 'cat'], ['d', 4, 'dog']], columns=['letter', 'number', 'animal'])
145 '''
146 df3 is like:
147   letter  number animal
148 0      c       3    cat
149 1      d       4    dog
150 '''
151
152 pd.concat([df1, df2, df3])
153 # Output:
154 #   animal letter  number
155 # 0    NaN      a       1
156 # 1    NaN      b       2
157 # 0    NaN      c       3
158 # 1    NaN      d       4
159 # 0    cat      c       3
160 # 1    dog      d       4
161
162 pd.concat([df1, df2, df3], join='inner')
163 # Output:
164 #   letter  number
165 # 0      a       1
166 # 1      b       2
167 # 0      c       3
168 # 1      d       4
169 # 0      c       3
170 # 1      d       4
171
172 ''' Column with NaN entries are excluded when join='inner' '''
```

```
173
174 df5 = pd.DataFrame([1,5], index=['a','b'])
175 '''
176 df5 is like:
177    0
178 a  1
179 b  5
180 '''
181
182 df6 = pd.DataFrame([2,6], index=['a','c'])
183 '''
184 df6 is like:
185    0
186 a  2
187 c  6
188 '''
189
190 pd.concat([df5, df6])
191 # Output:
192 #    0
193 # a  1
194 # b  5
195 # a  2
196 # c  6
197
198 pd.concat([df5, df6], verify_integrity=True)
199 # Output:
200 # ValueError: Indexes have overlapping values: ['a']
201
202 ''' Check whether the new concatenated axis contains duplicates. '''
203
```

pd.merge

Merge, join, and concatenate

- https://pandas.pydata.org/pandas-docs/stable/merging.html

```
pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None,
         left_index=False, right_index=False, sort=True,
         suffixes=('_x', '_y'), copy=True, indicator=False,
         validate=None)
```

- `left`: A DataFrame object
- `right`: Another DataFrame object
- `on`: Columns (names) to join on. Must be found in both the left and right DataFrame objects. If not passed and `left_index` and `right_index` are `False`, the intersection of the columns in the DataFrames will be inferred to be the join keys
- `left_on`: Columns from the left DataFrame to use as keys. Can either be column names or arrays with length equal to the length of the DataFrame
- `right_on`: Columns from the right DataFrame to use as keys. Can either be column names or arrays with length equal to the length of the DataFrame
- `left_index`: If `True`, use the index (row labels) from the left DataFrame as its join key(s). In the case of a DataFrame with a MultiIndex (hierarchical), the number of levels must match the number of join keys from the right DataFrame
- `right_index`: Same usage as `left_index` for the right DataFrame
- `how`: One of `'left'`, `'right'`, `'outer'`, `'inner'`. Defaults to `inner`. See below for more detailed description of each method

- `sort`: Sort the result DataFrame by the join keys in lexicographical order. Defaults to `True`, setting to `False` will improve performance substantially in many cases

- `suffixes`: A tuple of string suffixes to apply to overlapping columns. Defaults to `('_x', '_y')`.

- `copy`: Always copy data (default `True`) from the passed DataFrame objects, even when reindexing is not necessary. Cannot be avoided in many cases but may improve performance / memory usage. The cases where copying can be avoided are somewhat pathological but this option is provided nonetheless.

- `indicator`: Add a column to the output DataFrame called `_merge` with information on the source of each row. `_merge` is Categorical-type and takes on a value of `left_only` for observations whose merge key only appears in `'left'` DataFrame, `right_only` for observations whose merge key only appears in `'right'` DataFrame, and `both` if the observation's merge key is found in both.

  New in version 0.17.0.

- `validate` : string, default None. If specified, checks if merge is of specified type.

  - "one_to_one" or "1:1": checks if merge keys are unique in both left and right datasets.
  - "one_to_many" or "1:m": checks if merge keys are unique in left dataset.
  - "many_to_one" or "m:1": checks if merge keys are unique in right dataset.
  - "many_to_many" or "m:m": allowed, but does not result in checks.

```
pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None,
         left_index=False, right_index=False, sort=True,
         suffixes=('_x', '_y'), copy=True, indicator=False,
         validate=None)
```

how : *{'left', 'right', 'outer', 'inner'}, default 'inner'*

- left: use only keys from left frame, similar to a SQL left outer join; preserve key order
- right: use only keys from right frame, similar to a SQL right outer join; preserve key order
- outer: use union of keys from both frames, similar to a SQL full outer join; sort keys lexicographically
- inner: use intersection of keys from both frames, similar to a SQL inner join; preserve the order of the left keys

```
In [38]: left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
   ....:                      'A': ['A0', 'A1', 'A2', 'A3'],
   ....:                      'B': ['B0', 'B1', 'B2', 'B3']})
   ....:

In [39]: right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
   ....:                       'C': ['C0', 'C1', 'C2', 'C3'],
   ....:                       'D': ['D0', 'D1', 'D2', 'D3']})
   ....:

In [40]: result = pd.merge(left, right, on='key')
```

left

	A	B	key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	A3	B3	K3

right

	C	D	key
0	C0	D0	K0
1	C1	D1	K1
2	C2	D2	K2
3	C3	D3	K3

Result

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A2	B2	K2	C2	D2
3	A3	B3	K3	C3	D3

```
In [46]: result = pd.merge(left, right, how='outer', on=['key1', 'key2'])
```

left

	A	B	key1	key2
0	A0	B0	K0	K0
1	A1	B1	K0	K1
2	A2	B2	K1	K0
3	A3	B3	K2	K1

right

	C	D	key1	key2
0	C0	D0	K0	K0
1	C1	D1	K1	K0
2	C2	D2	K1	K0
3	C3	D3	K2	K0

Result

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A1	B1	K0	K1	NaN	NaN
2	A2	B2	K1	K0	C1	D1
3	A2	B2	K1	K0	C2	D2
4	A3	B3	K2	K1	NaN	NaN
5	NaN	NaN	K2	K0	C3	D3

```
"""
Created on Thu Nov 23 20:28:36 2017

@author: wanglu
"""

#import package
import pandas as pd
import os

#define two DataFrame to be merged
left = pd.DataFrame({'key' : ['a','b','c','d','e'],
                     'value_1' : [1,2,3,4,5],
                     'value_2' : ['I','II','III','IV','V']})

right = pd.DataFrame({'key' : ['a','b','c','d'],
                     'value_3' : ['A','B','C','D'],
                     'value_4' : ['one','two','three','four']})

#merge DataFrame 'right' to 'left' using variables in column 'key' in the left DataFrame as join keys
# if how = 'outer', use union of keys from both frames
result = pd.merge(left, right, how='outer', on='key')

#save this DataFrame to a csv file
data_path = '/Users/wanglu/Desktop/Pandas Functions.General Functions/Pandas Functions.General Functions.merge'
result.to_csv(data_path + os.sep + 'General_Function_merge_result.csv')
```

	key	value_1	value_2	value_3	value_4
0	a	1	I	A	one
1	b	2	II	B	two
2	c	3	III	C	three
3	d	4	IV	D	four
4	e	5	V		

pandas.merge

`pandas.merge`(*left*, *right*, *how='inner'*, *on=None*, *left_on=None*, *right_on=None*, *left_index=False*, *right_index=False*, *sort=False*, *suffixes=('_x', '_y')*, *copy=True*, *indicator=False*, *validate=None*) [source]

Merge DataFrame objects by performing a database-style join operation by columns or indexes.

If joining columns on columns, the DataFrame indexes *will be ignored*. Otherwise if joining indexes on indexes or indexes on a column or columns, the index will be passed on.

left *: DataFrame*

right *: DataFrame*

how *: {'left', 'right', 'outer', 'inner'}, default 'inner'*

- left: use only keys from left frame, similar to a SQL left outer join; preserve key order
- right: use only keys from right frame, similar to a SQL right outer join; preserve key order
- outer: use union of keys from both frames, similar to a SQL full outer join; sort keys lexicographically
- inner: use intersection of keys from both frames, similar to a SQL inner join; preserve the order of the left keys

on *: label or list*

Field names to join on. Must be found in both DataFrames. If on is None and not merging on indexes, then it merges on the intersection of the columns by default.

left_on *: label or list, or array-like*

Field names to join on in left DataFrame. Can be a vector or list of vectors of the length of the DataFrame to use a particular vector as the join key instead of columns

right_on *: label or list, or array-like*

Field names to join on in right DataFrame or vector/list of vectors per left_on docs

Parameters:

left_index *: boolean, default False*

Use the index from the left DataFrame as the join key(s). If it is a MultiIndex, the number of keys in the other DataFrame (either the index or a number of columns) must match the number of levels

right_index *: boolean, default False*

Use the index from the right DataFrame as the join key. Same caveats as left_index

sort *: boolean, default False*

Sort the join keys lexicographically in the result DataFrame. If False, the order of the join keys depends on the join type (how keyword)

suffixes *: 2-length sequence (tuple, list, ...)*

Suffix to apply to overlapping column names in the left and right side, respectively

copy *: boolean, default True*

If False, do not copy data unnecessarily

indicator *: boolean or string, default False*

If True, adds a column to output DataFrame called “_merge” with information on the source of each row. If string, column with information on source of each row will be added to output DataFrame, and column will be named value of string. Information column is Categorical-type and takes on a value of “left_only” for observations whose merge key only appears in ‘left’ DataFrame, “right_only” for observations whose merge key only appears in ‘right’ DataFrame, and “both” if the observation’s merge key is found in both.

New in version 0.17.0.

validate *: string, default None*

If specified, checks if merge is of specified type.

- “one_to_one” or “1:1”: check if merge keys are unique in both left and right datasets.
- “one_to_many” or “1:m”: check if merge keys are unique in left dataset.
- “many_to_one” or “m:1”: check if merge keys are unique in right dataset.
- “many_to_many” or “m:m”: allowed, but does not result in checks.

New in version 0.21.0.

Returns:

merged : *DataFrame*

The output type will the be same as ‘left’, if it is a subclass of DataFrame.

See also: `merge_ordered`, `merge_asof`

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Dec 20 16:44:28 2017

@author: DemiWu
"""

import pandas as pd
df1 = pd.DataFrame({"city": ["new york","chicago","orlando", "baltimore"],
    "temperature": [21,14,35, 38]})
'''df1 looks like
        city  temperature
0   new york           21
1    chicago           14
2    orlando           35
3  baltimore           38
'''

df2 = pd.DataFrame({"city": ["chicago","new york","san diego"],
    "humidity": [65,68,71]})
'''data looks like
         city  humidity
0    chicago        65
1   new york        68
2  san diego        71
'''
```

```
#merge the two dataframes using only the cities that are in both dataframes
df3=pd.merge(df1,df2,on="city",how="inner")
'''the result is
       city  temperature  humidity
0  new york           21        68
1   chicago           14        65
'''

#merge the two dataframes using the cities that are in either dataframes
df3=pd.merge(df1,df2,on="city",how="outer")
'''the result is
        city  temperature  humidity
0   new york         21.0      68.0
1    chicago         14.0      65.0
2    orlando         35.0       NaN
3  baltimore         38.0       NaN
4  san diego          NaN      71.0
'''

#merge the two dataframes using the cities in df1
df3=pd.merge(df1,df2,on="city",how="left")
'''the result is
        city  temperature  humidity
0   new york           21      68.0
1    chicago           14      65.0
2    orlando           35       NaN
3  baltimore           38       NaN
'''

#merge the two dataframes using the cities in df2
df3=pd.merge(df1,df2,on="city",how="right")
'''the result is
        city  temperature  humidity
0   new york         21.0        68
1    chicago         14.0        65
2  san diego          NaN        71
'''
```

```
#add a column showing the sources of each row
df3=pd.merge(df1,df2,on="city",how="outer",indicator=True)
'''the result is
        city  temperature  humidity      _merge
0   new york         21.0      68.0        both
1    chicago         14.0      65.0        both
2    orlando         35.0       NaN   left_only
3  baltimore         38.0       NaN   left_only
4  san diego          NaN      71.0  right_only
'''

#use the "left_on" and "right_on" argument to replace "on"
df3=pd.merge(df1,df2,left_on="city", right_on="city", how="outer",indicator=True)
'''the result is
        city  temperature  humidity      _merge
0   new york         21.0      68.0        both
1    chicago         14.0      65.0        both
2    orlando         35.0       NaN   left_only
3  baltimore         38.0       NaN   left_only
4  san diego          NaN      71.0  right_only
'''
#the same as the previous result using "on" argument

```

```
#merge overlapping columns
df1 = pd.DataFrame({
    "city": ["new york","chicago","orlando", "baltimore"],
    "temperature": [21,14,35,38],
    "humidity": [65,68,71, 75]})
'''the data looks like
         city  humidity  temperature
0   new york        65           21
1    chicago        68           14
2    orlando        71           35
3  baltimore        75           38
'''
df2 = pd.DataFrame({
    "city": ["chicago","new york","san diego"],
    "temperature": [21,14,35],
    "humidity": [65,68,71]})
'''the data looks like
         city  humidity  temperature
0    chicago        65           21
1   new york        68           14
2  san diego        71           35
'''
df3= pd.merge(df1,df2,on="city",how="outer", suffixes=('_first','_second'))
'''the result is
         city  humidity_first  temperature_first  humidity_second  \
0   new york            65.0               21.0             68.0
1    chicago            68.0               14.0             65.0
2    orlando            71.0               35.0              NaN
3  baltimore            75.0               38.0              NaN
4  san diego             NaN                NaN             71.0

   temperature_second
0                14.0
1                21.0
2                 NaN
3                 NaN
4                35.0
'''
```

```
#use the index as the join key
df1 = pd.DataFrame({
    "city": [1,2,3],
    "temperature": [21,14,35],})
df1.set_index('city',inplace=True)
'''the data looks like
      temperature
city
1              21
2              14
3              35
'''
df2 = pd.DataFrame({
    "city": [2,3,4],
    "humidity": [65,68,75],})
df2.set_index('city',inplace=True)
'''the data looks like
      humidity
city
2           65
3           68
4           75
'''
df3= pd.merge(df1,df2,left_index=True, right_index=True, how="outer")
'''the result is
      temperature  humidity
city
1            21.0       NaN
2            14.0      65.0
3            35.0      68.0
4             NaN      75.0
'''
```

```
161
162 #about the sort argument
163 df1 = pd.DataFrame({
164     "city": ["new york","chicago","orlando"],
165     "temperature": [21,14,35],})
166 df1.set_index('city',inplace=True)
167 df2 = pd.DataFrame({
168     "city": ["chicago","new york","orlando"],
169     "humidity": [65,68,75],})
170 df2.set_index('city',inplace=True)
171 df3= pd.merge(df1,df2,left_index=True, right_index=True, how="outer", sort=True)
172 '''the result is
173           temperature  humidity
174 city
175 chicago            14        65
176 new york           21        68
177 orlando            35        75
178 '''
179 #the join keys are sorted lexicographically
180
```

pd.merge_asof

pandas.merge_asof

pandas.**merge_asof**(*left*, *right*, *on=None*, *left_on=None*, *right_on=None*, *left_index=False*, *right_index=False*, *by=None*, *left_by=None*, *right_by=None*, *suffixes=('_x', '_y')*, *tolerance=None*, *allow_exact_matches=True*, *direction='backward'*) [source]

Perform an asof merge. This is similar to a left-join except that we match on nearest key rather than equal keys.

Both DataFrames must be sorted by the key.

For each row in the left DataFrame:

- A “backward” search selects the last row in the right DataFrame whose ‘on’ key is less than or equal to the left’s key.
- A “forward” search selects the first row in the right DataFrame whose ‘on’ key is greater than or equal to the left’s key.
- A “nearest” search selects the row in the right DataFrame whose ‘on’ key is closest in absolute distance to the left’s key.

The default is “backward” and is compatible in versions below 0.20.0. The direction parameter was added in version 0.20.0 and introduces “forward” and “nearest”.

Optionally match on equivalent keys with ‘by’ before searching with ‘on’.

New in version 0.19.0.

Parameters:

left *: DataFrame*

right *: DataFrame*

on *: label*

Field name to join on. Must be found in both DataFrames. The data MUST be ordered. Furthermore this must be a numeric column, such as datetimelike, integer, or float. On or left_on/right_on must be given.

left_on *: label*

Field name to join on in left DataFrame.

right_on *: label*

Field name to join on in right DataFrame.

left_index *: boolean*

Use the index of the left DataFrame as the join key.

New in version 0.19.2.

right_index *: boolean*

Use the index of the right DataFrame as the join key.

New in version 0.19.2.

by *: column name or list of column names*

Match on these columns before performing merge operation.

left_by *: column name*

Field names to match on in the left DataFrame.

New in version 0.19.2.

right_by *: column name*

Field names to match on in the right DataFrame.

New in version 0.19.2.

suffixes *: 2-length sequence (tuple, list, ...)*

Suffix to apply to overlapping column names in the left and right side, respectively.

tolerance *: integer or Timedelta, optional, default None*

Select asof tolerance within this range; must be compatible with the merge index.

allow_exact_matches *: boolean, default True*

- If True, allow matching with the same ‘on’ value (i.e. less-than-or-equal-to / greater-than-or-equal-to)
- If False, don’t match the same ‘on’ value (i.e., stricly less-than / strictly greater-than)

direction *: ‘backward’ (default), ‘forward’, or ‘nearest’*

Whether to search for prior, subsequent, or closest matches.

New in version 0.20.0.

Returns: **merged** *: DataFrame*

See also: `merge`, `merge_ordered`

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Dec 20 19:00:09 2017

@author: DemiWu
"""

import pandas as pd
trades = pd.DataFrame({'time': pd.to_datetime(['20160525 13:30:00.023', '20160525 13:30:00.038', '20160525 13:30:00.048',
                                               '20160525 13:30:00.048', '20160525 13:30:00.048']),
                       'ticker': ['MSFT', 'MSFT', 'GOOG', 'GOOG', 'AAPL'],
                       'price': [51.95, 51.95, 720.77, 720.92, 98.00],
                       'quantity': [75, 155, 100, 100, 100]},
                        columns=['time', 'ticker', 'price', 'quantity'])
'''trades looks like
                    time ticker   price  quantity
0 2016-05-25 13:30:00.023   MSFT   51.95        75
1 2016-05-25 13:30:00.038   MSFT   51.95       155
2 2016-05-25 13:30:00.048   GOOG  720.77       100
3 2016-05-25 13:30:00.048   GOOG  720.92       100
4 2016-05-25 13:30:00.048   AAPL   98.00       100
'''
quotes = pd.DataFrame({'time': pd.to_datetime(['20160525 13:30:00.023', '20160525 13:30:00.023', '20160525 13:30:00.030',
                                               '20160525 13:30:00.041', '20160525 13:30:00.048', '20160525 13:30:00.049',
                                               '20160525 13:30:00.072', '20160525 13:30:00.075']),
                       'ticker': ['GOOG', 'MSFT', 'MSFT', 'MSFT', 'GOOG', 'AAPL', 'GOOG', 'MSFT'],
                       'bid': [720.50, 51.95, 51.97, 51.99, 720.50, 97.99, 720.50, 52.01],
                       'ask': [720.93, 51.96, 51.98, 52.00, 720.93, 98.01, 720.88, 52.03]},
                        columns=['time', 'ticker', 'bid', 'ask'])
'''quotes looks like
                    time ticker     bid     ask
0 2016-05-25 13:30:00.023   GOOG  720.50  720.93
1 2016-05-25 13:30:00.023   MSFT   51.95   51.96
2 2016-05-25 13:30:00.030   MSFT   51.97   51.98
3 2016-05-25 13:30:00.041   MSFT   51.99   52.00
4 2016-05-25 13:30:00.048   GOOG  720.50  720.93
5 2016-05-25 13:30:00.049   AAPL   97.99   98.01
6 2016-05-25 13:30:00.072   GOOG  720.50  720.88
7 2016-05-25 13:30:00.075   MSFT   52.01   52.03
'''
```

```
42
43 pd.merge_asof(trades, quotes, on='time', by='ticker')
44 '''the result is
45                      time ticker   price  quantity     bid     ask
46 0 2016-05-25 13:30:00.023   MSFT   51.95        75   51.95   51.96
47 1 2016-05-25 13:30:00.038   MSFT   51.95       155   51.97   51.98
48 2 2016-05-25 13:30:00.048   GOOG  720.77       100  720.50  720.93
49 3 2016-05-25 13:30:00.048   GOOG  720.92       100  720.50  720.93
50 4 2016-05-25 13:30:00.048   AAPL   98.00       100     NaN     NaN
51 '''
52 #for the row with the time 13:30:00.038, the merged dataframe uses the last row in "quotes" which matches with\
53 #the ticker "MSFT" and whose time is less than 13:30:00.038 (the one with the time 13:30:00.030)
54
55 pd.merge_asof(trades, quotes, on='time', by='ticker', tolerance=pd.Timedelta('2ms'))
56 '''the result is
57                      time ticker   price  quantity     bid     ask
58 0 2016-05-25 13:30:00.023   MSFT   51.95        75   51.95   51.96
59 1 2016-05-25 13:30:00.038   MSFT   51.95       155     NaN     NaN
60 2 2016-05-25 13:30:00.048   GOOG  720.77       100  720.50  720.93
61 3 2016-05-25 13:30:00.048   GOOG  720.92       100  720.50  720.93
62 4 2016-05-25 13:30:00.048   AAPL   98.00       100     NaN     NaN
63 '''
64 #because 13:30:00.030 is away from 13:30:00.038 by more than 2ms, the merged does not uses the corresponding data\
65 #from "quotes"
66
```

```
pd.merge_asof(trades, quotes, on='time', by='ticker', tolerance=pd.Timedelta('10ms'), allow_exact_matches=False)
'''the result is
                     time ticker   price  quantity    bid    ask
0 2016-05-25 13:30:00.023   MSFT   51.95        75    NaN    NaN
1 2016-05-25 13:30:00.038   MSFT   51.95       155  51.97  51.98
2 2016-05-25 13:30:00.048   GOOG  720.77       100    NaN    NaN
3 2016-05-25 13:30:00.048   GOOG  720.92       100    NaN    NaN
4 2016-05-25 13:30:00.048   AAPL   98.00       100    NaN    NaN
'''
#only the row with the time 13:30:00.038 has bid-ask data from "quotes" because it uses data from near rows rather than\
#exact match

pd.merge_asof(trades, quotes, on='time', by='ticker', direction='nearest')
'''the result is
                     time ticker   price  quantity     bid     ask
0 2016-05-25 13:30:00.023   MSFT   51.95        75   51.95   51.96
1 2016-05-25 13:30:00.038   MSFT   51.95       155   51.99   52.00
2 2016-05-25 13:30:00.048   GOOG  720.77       100  720.50  720.93
3 2016-05-25 13:30:00.048   GOOG  720.92       100  720.50  720.93
4 2016-05-25 13:30:00.048   AAPL   98.00       100   97.99   98.01
'''
#the row with the time 13:30:00.038 now uses the bid-ask data from "quotes" in the row with time 13:00:00.041, \
#which is the closest to 13:30:00.038

#the other arguments are the same as in pandas.merge function
```

Function Type	# Functions
General functions	26
Series	283
DataFrame	197
Panel	107
Index	69
Other indices	87
Window	35
Groupby	63
Resampling	29
Style	16
General utility functions	58
Total	970

Course Outline

- Data management with Python (3 weeks)
  - Packages
    - Numpy/Scipy
    - Dataframe
    - Pandas
  - OOP
    - Class
  - Geolocation
  - Multiprocessing
  - Pickle, Json, HDF5 (.h5)
  - Web scraping (regular expression, selenium)

Object

- An Object is a bit of self-contained Code and Data
- A key aspect of the Object approach is to break the problem into smaller understandable parts (divide and conquer)
- Objects have boundaries that allow us to ignore un-needed detail
- We have been using objects all along: String Objects, Integer Objects, Dictionary Objects, List Objects...

Definitions

- Class - a template - Dog
- Method or Message - A defined capability of a class - bark()
- Field or attribute- A bit of data in a class - length
- Object or Instance - A particular instance of a class - Snoopy

Terminology: Class

Defines the abstract characteristics of a thing (object), including:

1. the thing's characteristics (its attributes, fields or properties)
2. the thing's behaviors (the things it can do, or methods, operations or features).

One might say that a class is a blueprint or factory that describes the nature of something.

For example, the class Dog would consist of traits shared by all dogs, such as breed and fur color (characteristics), and the ability to bark and sit (behaviors).

Terminology: Class

A pattern (exemplar) of a class. The class of Dog defines all possible dogs by listing the characteristics and behaviors they can have; the object Snoopy is one particular dog, with particular versions of the characteristics. A Dog has fur; Snoopy has white fur.

Terminology: Instance

One can have an instance of a class or a particular object. The instance is the actual object created at runtime. The Snoopy object is an instance of the Dog class. The set of values of the attributes of a particular object is called its state. The object consists of state and the behavior that's defined in the object's class.

Object and Instance are often used interchangeably.

Terminology: Method

An object's abilities. In language, methods are verbs. Snoopy, being a Dog, has the ability to bark. So bark() is one of Snoopy's methods. He may have other methods as well, for example sit() or eat() or walk(). Within the program, using a method usually affects only one particular object; all Dogs can bark, but you need only one particular dog to do the barking

Method and Message are often used interchangeably.

A Sample Class

class is a reserved word.

This is the template for making PartyAnimal objects.

```
class PartyAnimal:
  x = 0

  def party(self) :
    self.x = self.x + 1
    print("So far",self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

Each PartyAnimal object has a bit of code.

Each PartyAnimal object has a bit of data.

Create a PartyAnimal object.

Tell the object to run the party() code.

PartyAnimal party an

run party() *within* the object an

Definitions

- Class - a template - Dog
- Method or Message - A defined capability of a class - bark()
- Object or Instance - A particular instance of a class - Snoopy

Playing with dir() and type()

A Nerdy Way to Find Capabilities

- The dir() command lists capabilities
  - Ignore the ones with underscores - these are used by Python itself
  - The rest are real operations that the object can perform
- It is like type() - it tells us something *about* a variable

```
>>> x = list()
>>> type(x)
<type 'list'>
>>> dir(x)
 '__add__', '__class__',
'__contains__', '__delattr__',
'__delitem__', '__delslice__',
'__doc__', '__eq__',
'__setitem__', '__setslice__',
'__str__', 'append', 'count',
'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort'
```

Try dir() with a String

```
>>> y = “Hello there”
>>> dir(y)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
'__eq__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__getslice__', '__gt__', '__hash__',
'__init__', '__le__', '__len__', '__lt__', '__repr__', '__rmod__',
'__rmul__', '__setattr__', '__str__', 'capitalize', 'center', 'count',
'decode', 'encode', 'endswith', 'expandtabs', 'find', 'index',
'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper',
'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',
'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

```
class PartyAnimal:
  x = 0

  def party(self) :
    self.x = self.x + 1
    print ("So far",self.x)

an = PartyAnimal()

print ("Type", type(an))
print ("Dir ", dir(an))
```

We can use dir() to find the "capabilities" of *our* newly created class.

```
$ python party2.py
Type <type 'instance'>
Dir  ['__doc__', '__module__',
'party', 'x']
```

Constructor

- The primary purpose of the constructor is to set up some instance variables to have the proper initial values when the object is created

```
class PartyAnimal:
  x = 0

  def __init__(self):
    print ("I am constructed“)

  def party(self) :
    self.x = self.x + 1
    print ("So far",self.x)

  def __del__(self):
    print ("I am destructed", self.x)

an = PartyAnimal()
an.party()
an.party()
an.party()
```

```
$ python party2.py
I am constructed
So far 1
So far 2
So far 3
I am destructed 3
```

The constructor and destructor are optional. The constructor is typically used to set up variables. The destructor is seldom used.

Constructor

- In object-oriented programming, a constructor in a class is a special block of statements called when an object is created

Many Instances

- We can create lots of objects - the class is the template for the object
- We can store each distinct object in its own variable
- We call this having multiple instances of the same class
- Each instance has its own copy of the instance variables

```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print (self.name,"constructed“)

   def party(self) :
     self.x = self.x + 1
     print (self.name,"party count",self.x)

s = PartyAnimal("Sally")
s.party()

j = PartyAnimal("Jim")
j.party()
s.party()
```

Constructors can have additional parameters. These can be used to setup instance variables for the particular instance of the class (i.e. for the particular obect).

Definitions

- Class - a template - Dog
- Method or Message - A defined capability of a class - bark()
- Object or Instance - A particular instance of a class - Lassie
- Constructor - A method which is called when the instance / object is created

Inheritance

https://docs.python.org/3/tutorial/classes.html#inheritance
http://www.ibiblio.org/g2swap/byteofpython/read/inheritance.html

Inheritance

- When we make a new class - we can reuse an existing class and inherit all the capabilities of an existing class and then add our own little bit to make our new class
- Another form of store and reuse
- Write once - reuse many times
- The new class (child) has all the capabilities of the old class (parent) - and then some more

Terminology: Inheritance

‘Subclasses’ are more specialized versions of a class, which inherit attributes and behaviors from their parent classes, and can introduce their own.

http://en.wikipedia.org/wiki/Object-oriented_programming

```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print (self.name,"constructed“)

   def party(self) :
     self.x = self.x + 1
     print (self.name,"party count",self.x)

class FootballFan(PartyAnimal):
   points = 0
   def touchdown(self):
      self.points = self.points + 7
      self.party()
      print (self.name,"points",self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```

FootballFan extends PartyAnimal. It has all the capabilities of PartyAnimal and more.

SUMMARY

Classes

```
class MyVector: """A simple vector class."""
    num_created = 0
    def __init__(self,x=0,y=0):
            self.__x = x
            self.__y = y
            MyVector.num_created += 1
    def get_size(self):
            return self.__x+self.__y
    @staticmethod
    def get_num_created
            return MyVector.num_created
```

```
#USAGE OF CLASS MyVector
print (MyVector.num_created)
v = MyVector()
w = MyVector(0.23,0.98)
print (w.get_size())
bool = isinstance(v, MyVector)
```

Output:

0

1.21

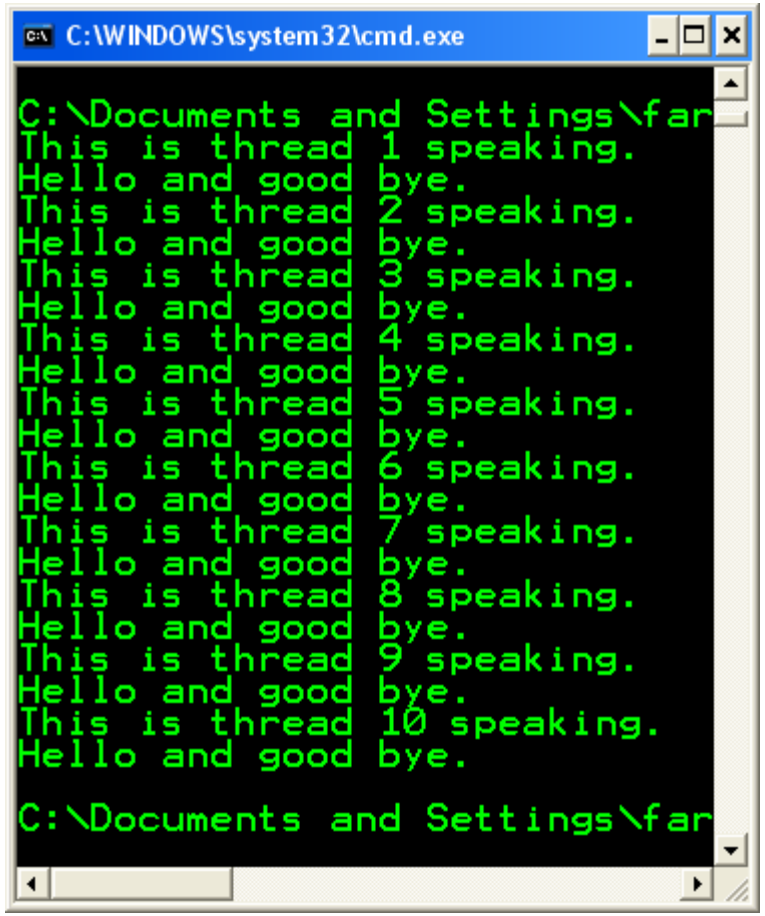
I/O

```
import os
print (os.getcwd()) #get "."
os.chdir('..')
import glob # file globbing
lst = glob.glob('*.txt') # get list of files
import shutil # mngmt tasks
shutil.copyfile('a.py','a.bak')

import pickle # serialization logic
ages = {"ron":18,"ted":21}
pickle.dump(ages,fout)
# serialize the map into a writable file
ages = pickle.load(fin)
# deserialize map from areadable file
```

```
# read binary records from a file
from struct import *
fin = None
try:
    fin = open("input.bin","rb")
    s = f.read(8)#easy to read in
    while (len(s) == 8):
        x,y,z = unpack(">HH<L", s)
        print ("Read record: " \
          "%04x %04x %08x"%(x,y,z))
        s = f.read(8)
except IOError:
    pass
if fin: fin.close()
```

Threading in Python

```
import threading
theVar = 1
class MyThread ( threading.Thread ):
   def run ( self ):
          global theVar
          print('This is thread ' + \
          str ( theVar ) + ' speaking.')
          print('Hello and good bye.' )
          theVar = theVar + 1
for x in range ( 10 ):
   MyThread().start()
```



Graphics to explore the data

Seaborn package is built on matplotlib but provides high level interface for drawing attractive statistical graphics, similar to ggplot2 library in R. It specifically targets statistical data visualization

To show graphs within Python notebook include inline directive:

In [ ]:
```
%matplotlib inline
```

Graphics

	description
distplot	histogram
barplot	estimate of central tendency for a numeric variable
violinplot	similar to boxplot, also shows the probability density of the data
jointplot	Scatterplot
regplot	Regression plot
pairplot	Pairplot
boxplot	boxplot
swarmplot	categorical scatterplot
factorplot	General categorical plot

Basic statistical Analysis

statsmodel and scikit-learn - both have a number of function for statistical analysis

The first one is mostly used for regular analysis using R style formulas, while scikit-learn is more tailored for Machine Learning.

statsmodels:

- linear regressions
- ANOVA tests
- hypothesis testings
- many more ...

scikit-learn:

- kmeans
- support vector machines
- random forests
- many more ...

See examples in the Tutorial Notebook