Focus on Data Science packages: Numpy, Pandas, Scipy

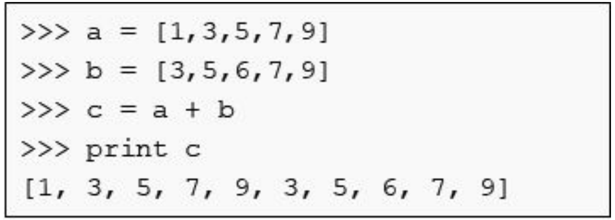
What is Numpy ?

Website: http://numpy.scipy.org

- Features required for Scientific Python:
  - Fast, multidimensional arrays
  - Libraries of reliable, tested scientific functions

NumPy is at the core of nearly every scientific Python application or module since it provides a fast N-d array datatype that can be manipulated in a vectorized form

Example: native python list adequate for small one-dimensional data



```
>>> a = [1,3,5,7,9]
>>> b = [3,5,6,7,9]
>>> c = a + b
>>> print c
[1, 3, 5, 7, 9, 3, 5, 6, 7, 9]
```

➔ But, can’t use directly with arithmetical operators (+, -, *, /, …)

➔ Need efficient arrays with arithmetic and better multidimensional tools

Why Numpy ?

Traditional version

```
>>> import time
>>> def trad_version () :
          t1 = time . time ()
          X = range (10000000)
          Y = range (10000000)
          Z = []
          for i in range (len ( X ) ) :
                    Z . append ( X [ i ] + Y [ i ])
          return time . time () - t1
>>> trad_version ()
1.9738149642944336
```

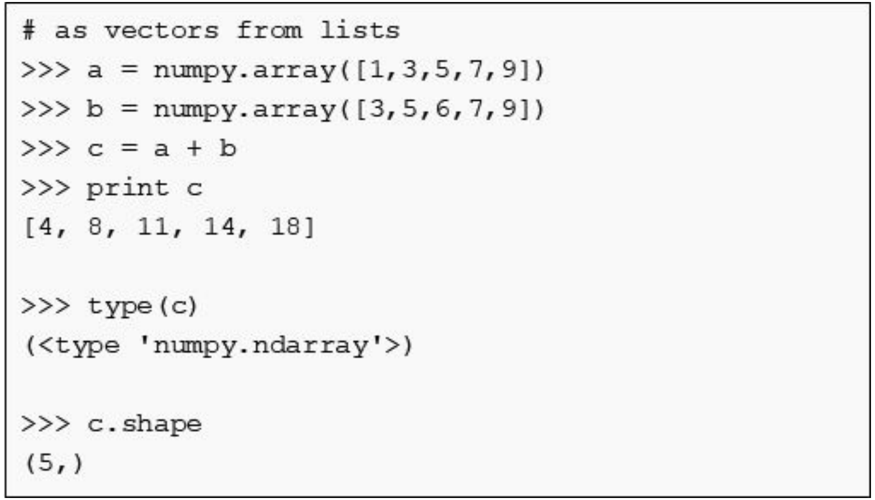
Numpy version

```
>>> def numpy_version () :
          t1 = time . time ()
          X = arange (10000000)
          Y = arange (10000000)
          Z = X + Y
          return time . time () - t1
>>> numpy_version ()
0.059307098388671875
```

Numpy arrays

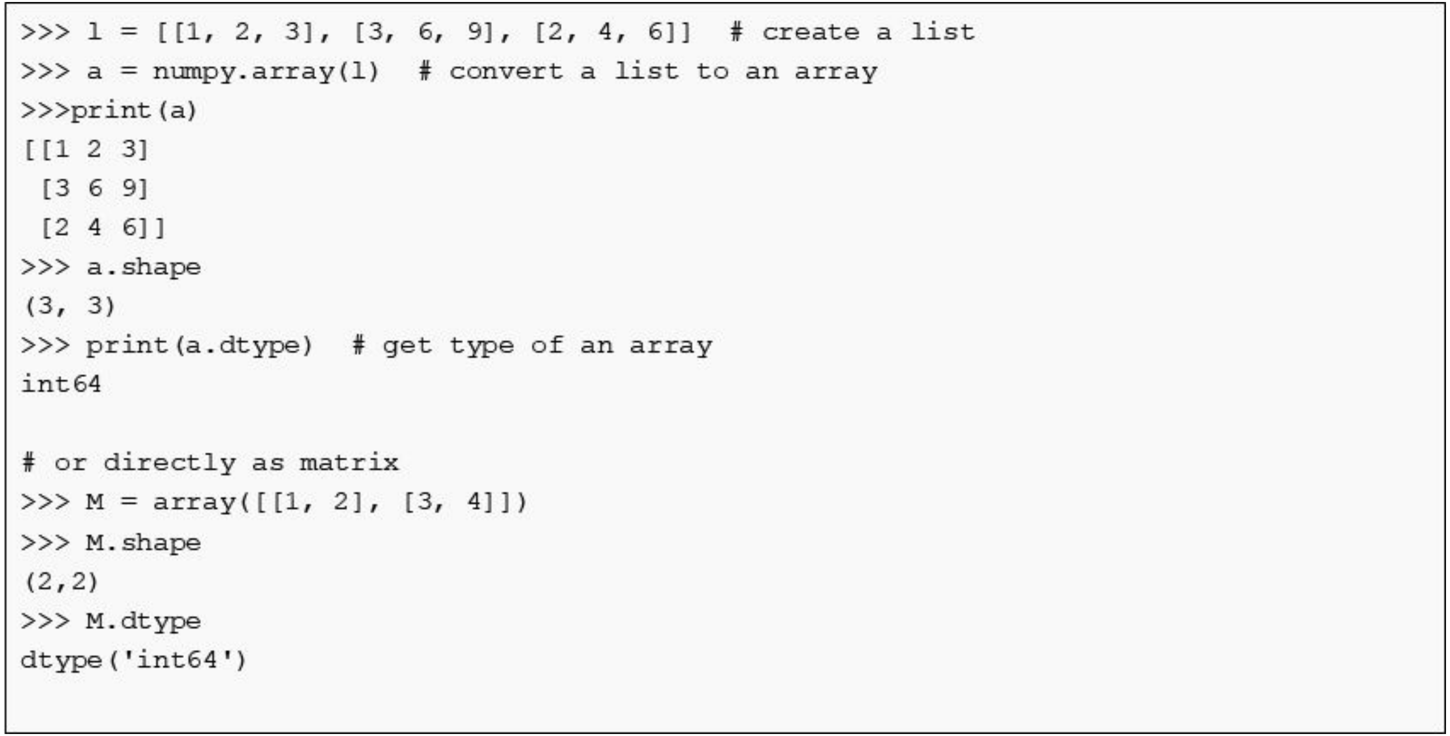
There are a number of ways to initialize new numpy arrays, for example from

- a Python list or tuples
- using functions that are dedicated to generating numpy arrays, such as arange, linspace, etc.
- reading data from files



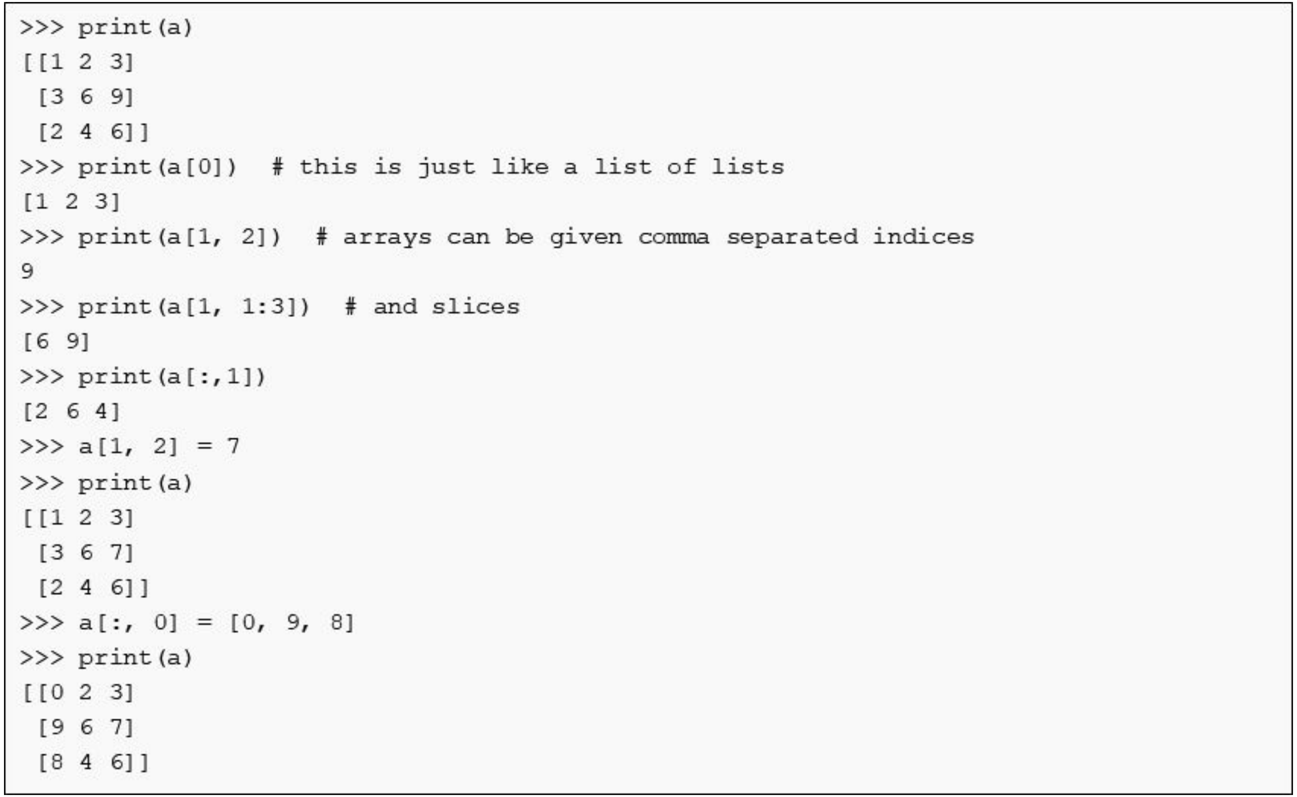
```
# as vectors from lists
>>> a = numpy.array([1,3,5,7,9])
>>> b = numpy.array([3,5,6,7,9])
>>> c = a + b
>>> print c
[4, 8, 11, 14, 18]

>>> type(c)
(<type 'numpy.ndarray'>)

>>> c.shape
(5,)
```

Numpy matrices

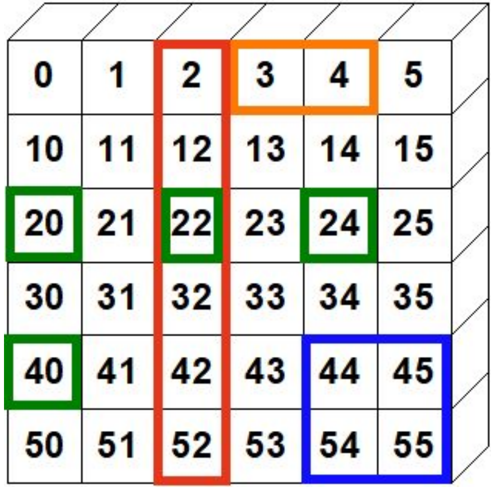


```
>>> l = [[1, 2, 3], [3, 6, 9], [2, 4, 6]]  # create a list
>>> a = numpy.array(l)  # convert a list to an array
>>>print(a)
[[1 2 3]
 [3 6 9]
 [2 4 6]]
>>> a.shape
(3, 3)
>>> print(a.dtype)  # get type of an array
int64

# or directly as matrix
>>> M = array([[1, 2], [3, 4]])
>>> M.shape
(2,2)
>>> M.dtype
dtype('int64')
```

Numpy arrays manipulation

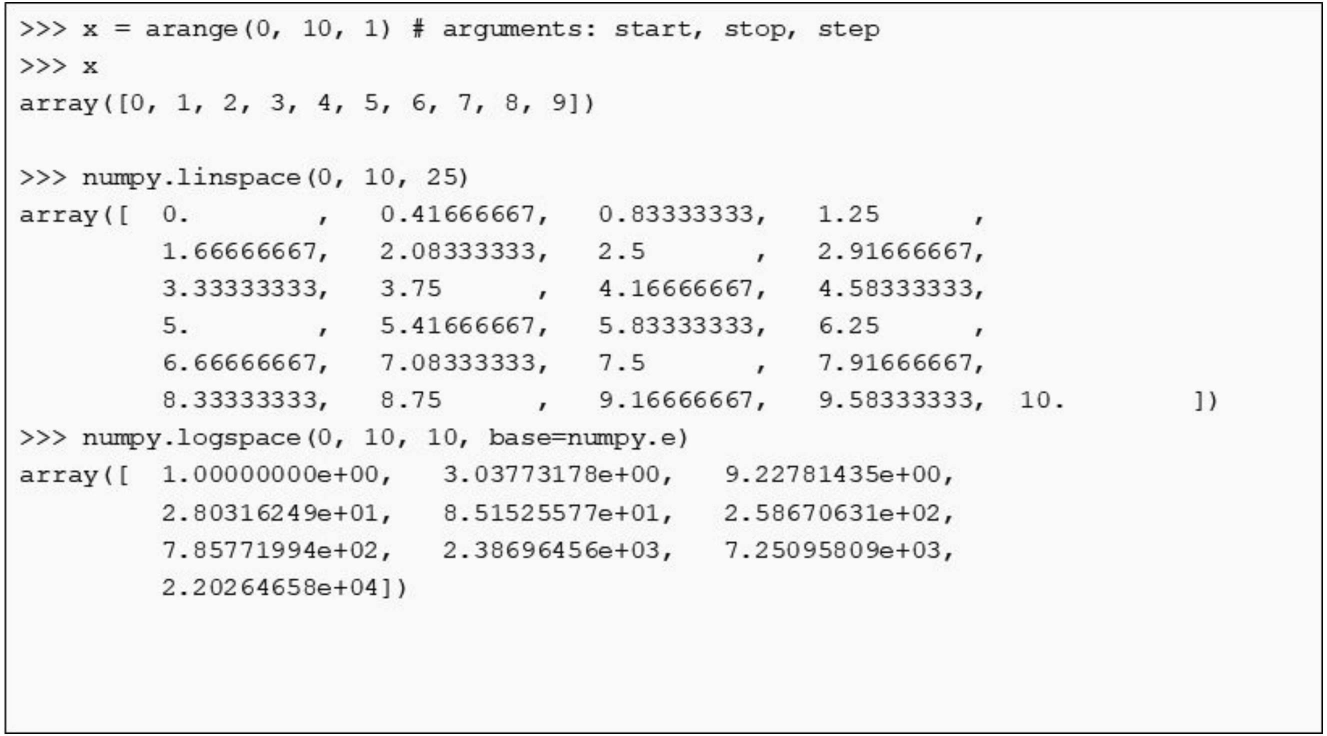


```
>>> print(a)
[[1 2 3]
 [3 6 9]
 [2 4 6]]
>>> print(a[0])  # this is just like a list of lists
[1 2 3]
>>> print(a[1, 2])  # arrays can be given comma separated indices
9
>>> print(a[1, 1:3])  # and slices
[6 9]
>>> print(a[:,1])
[2 6 4]
>>> a[1, 2] = 7
>>> print(a)
[[1 2 3]
 [3 6 7]
 [2 4 6]]
>>> a[:, 0] = [0, 9, 8]
>>> print(a)
[[0 2 3]
 [9 6 7]
 [8 4 6]]
```



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Numpy useful functions to generate arrays



```
>>> x = arange(0, 10, 1) # arguments: start, stop, step
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> numpy.linspace(0, 10, 25)
array([  0.        ,   0.41666667,   0.83333333,   1.25      ,
         1.66666667,   2.08333333,   2.5       ,   2.91666667,
         3.33333333,   3.75      ,   4.16666667,   4.58333333,
         5.        ,   5.41666667,   5.83333333,   6.25      ,
         6.66666667,   7.08333333,   7.5       ,   7.91666667,
         8.33333333,   8.75      ,   9.16666667,   9.58333333,  10.        ])
>>> numpy.logspace(0, 10, 10, base=numpy.e)
array([  1.00000000e+00,   3.03773178e+00,   9.22781435e+00,
         2.80316249e+01,   8.51525577e+01,   2.58670631e+02,
         7.85771994e+02,   2.38696456e+03,   7.25095809e+03,
         2.20264658e+04])
```

Numpy useful methods

```
>>> arr.sum()
145
>>> arr.mean()
14.5
>>> arr.std()
2.8722813232690143
>>> arr.max()
19
>>> arr.min()
10
>>> arr = numpy.array([4.5, 2.3, 6.7, 1.2, 1.8, 5.5])
>>> arr.sort()  # acts on array itself
>>> print(arr)
[ 1.2  1.8  2.3  4.5  5.5  6.7]
```

Numpy - Statistics

- In addition to the mean, var, and std functions, NumPy supplies several other methods for returning statistical features of arrays. The median can be found:

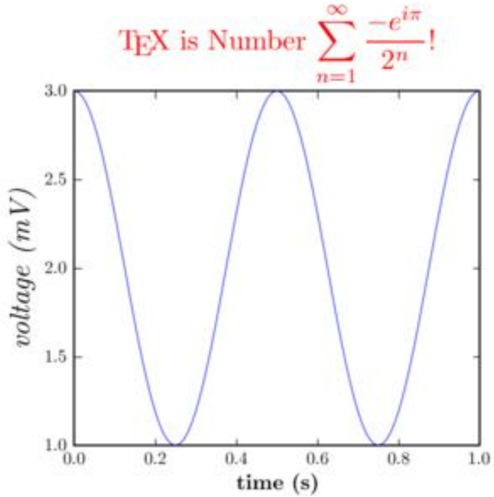
```
>>> a = np.array([1, 4, 3, 8, 9, 2, 3], float)
>>> np.median(a)
3.0
```

- The correlation coefficient for multiple variables observed at multiple instances can be found for arrays of the form [[x1, x2, …], [y1, y2, …], [z1, z2, …], …] where x, y, z are different observables and the numbers indicate the observation times:

```
>>> a = np.array([[1, 2, 1, 3], [5, 3, 1, 8]], float)
>>> c = np.corrcoef(a)
>>> c
array([[ 1.        ,  0.72870505],
     [ 0.72870505,  1.        ]])
```

What is Matplotlib ?

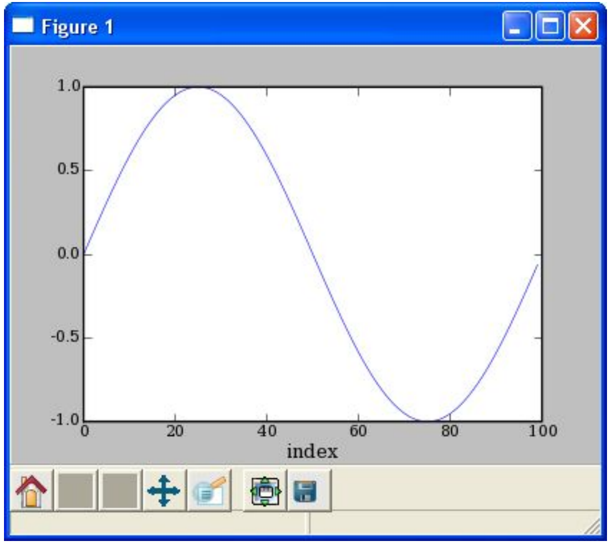
- Requires NumPy extension. Provides powerful plotting commands.
- http://matplotlib.sourceforge.net
- Matplotlib for day-to-day data exploration: it has a large community, tons of plot types, and is well integrated into ipython. It is the de-facto standard for ‘command line’ plotting from ipython.

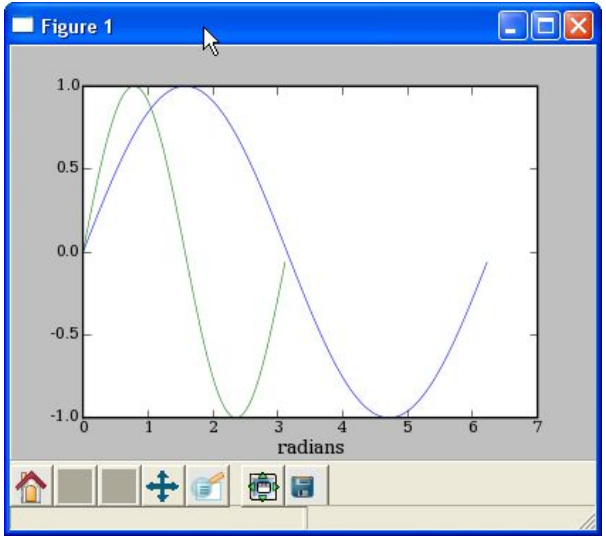


Matplotlib - Lineplot

```
>>> x = arange(50)*2*pi/50.
>>> y = sin(x)
>>> plot(y)
>>> xlabel(‘index’)
```

```
>>> plot(x,y,x2,y2)
>>> xlabel(‘radians’)
```

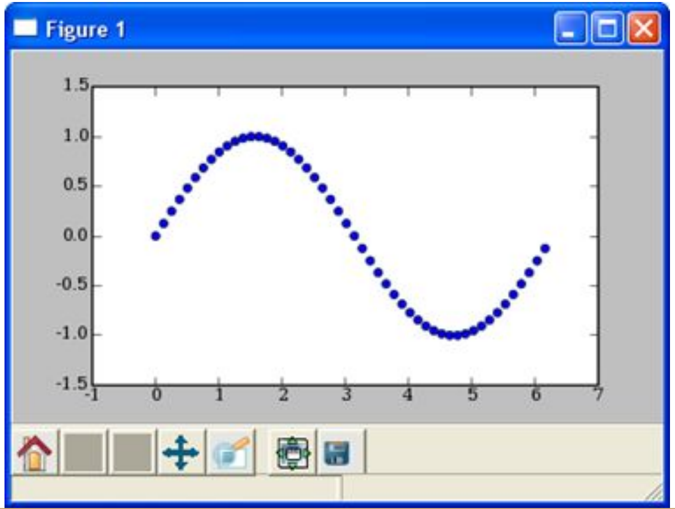


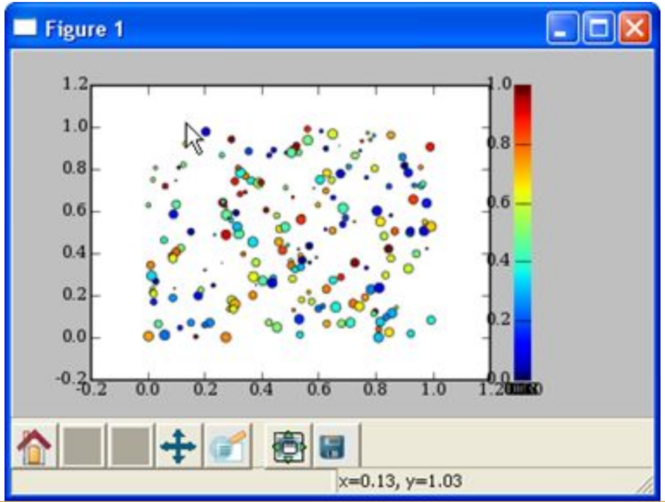


Matplotlib - Scatterplot

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

```
>>> x = rand(200)
>>> y = rand(200)
>>> size = rand(200)*30
>>> color = rand(200)
>>> scatter(x, y, size, color)
>>> colorbar()
```



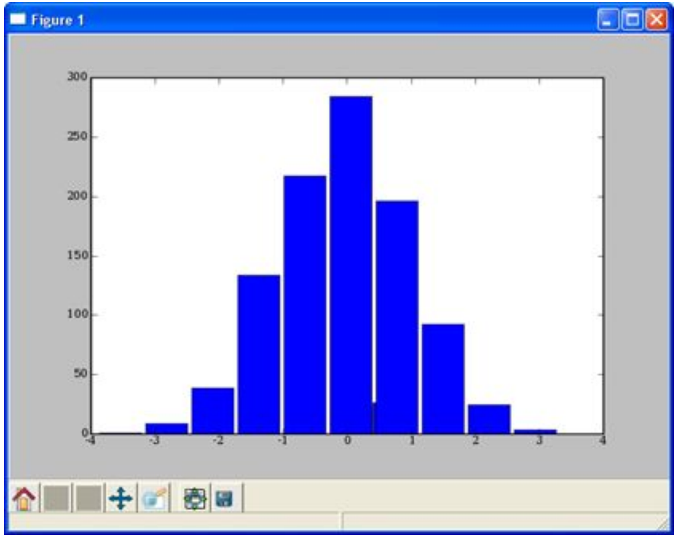


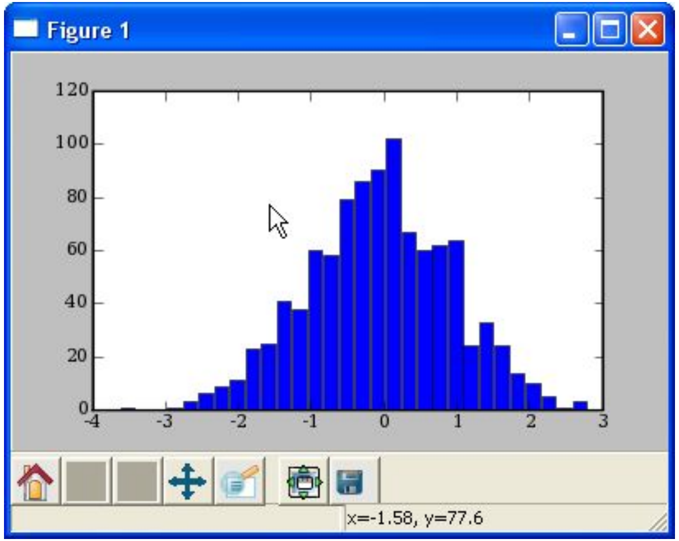
x=0.13, y=1.03

Matplotlib - Histograms

```
# plot histogram
# default to 10 bins
>>> hist(randn(1000))
```

```
# change the number of bins
>>> hist(randn(1000), 30)
```

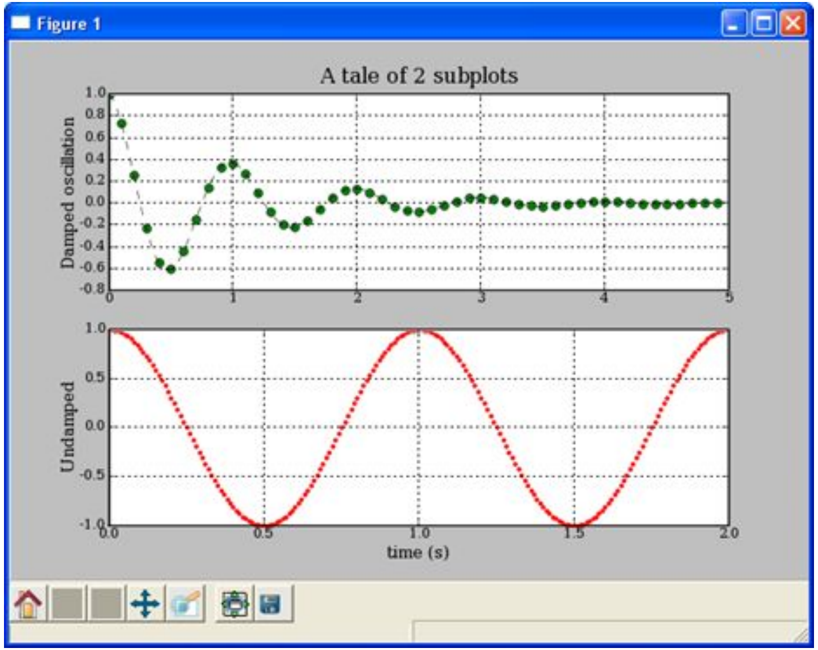




x=-1.58, y=77.6

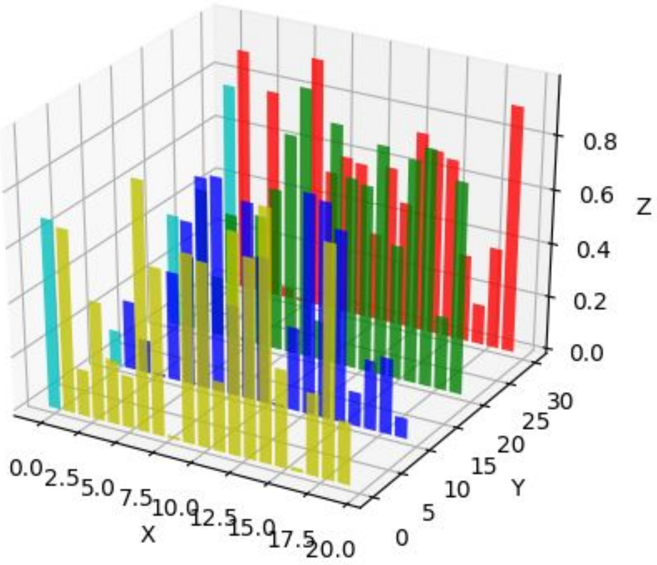
Matplotlib - Subplots

```
t1 = arange(0.0, 5.0, 0.1)
t2 = arange(0.0, 5.0, 0.02)
t3 = arange(0.0, 2.0, 0.01)

subplot(211)
l = plot(t1, f(t1), 'bo', t2, f(t2),
        'k--')
setp(l, 'markerfacecolor', 'g')
grid(True)
title('A tale of 2 subplots')
ylabel('Damped oscillation')

subplot(212)
plot(t3, cos(2*pi*t3), 'r.')
grid(True)
xlabel('time (s)')
ylabel('Undamped')
show()
```



Matplotlib - 3D plots

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
for c, z in zip(['r', 'g', 'b', 'y'], [30, 20,
10, 0]):
    xs = np.arange(20)
    ys = np.random.rand(20)
    cs = [c] * len(xs)
    cs[0] = 'c'
    ax.bar(xs, ys, zs=z, zdir='y', color=cs,
alpha=0.8)

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()
```



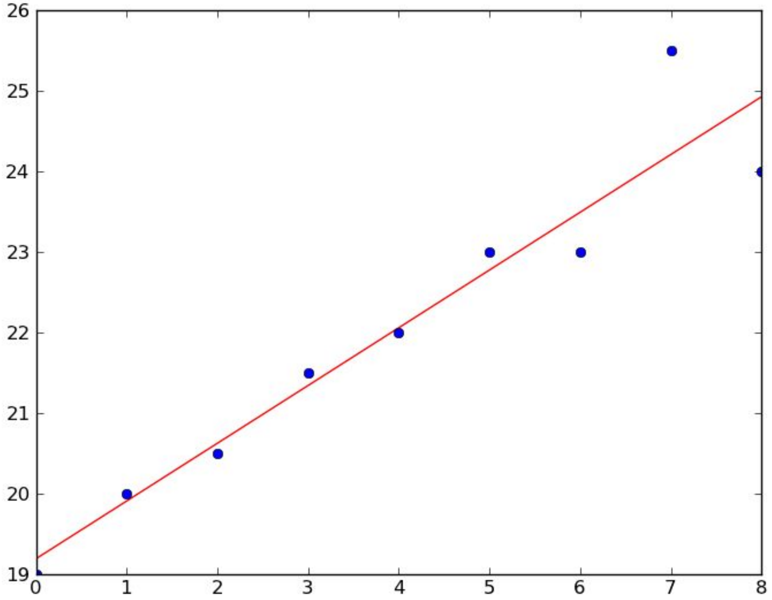
What is Scipy?

- Available at www.scipy.org
- “SciPy is a collection of mathematical algorithms and convenience functions built on the Numpy extension of Python.

Subpackage	**Description**
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions

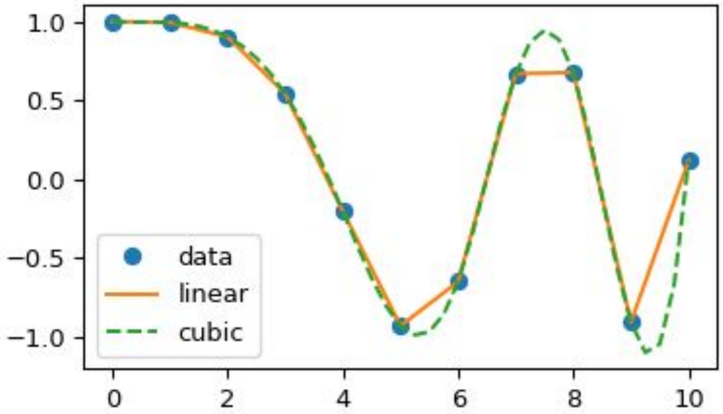
Scipy - Linear regressions

```
xi = arange(0,9)
# linearly generated sequence
y = [19, 20, 20.5, 21.5, 22, 23, 23, 25.5, 24]

slope, intercept, r_value, p_value, std_err =
stats.linregress(xi,y)

print 'r value', r_value
print  'p_value', p_value
print 'standard deviation', std_err

line = slope*xi+intercept
plot(xi,line,'r-',xi,y,'o')
show()
```



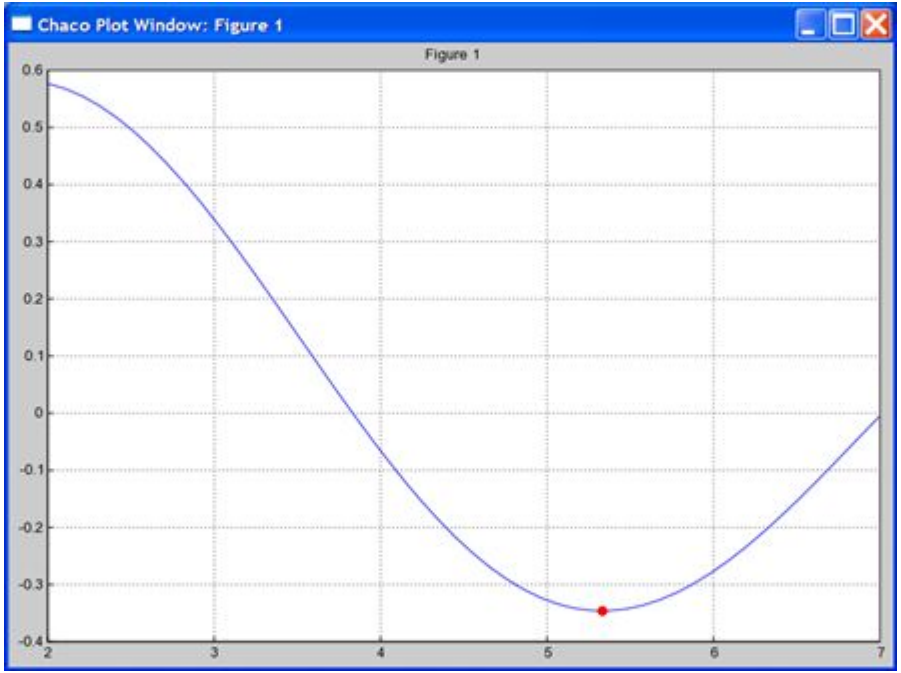
Scipy - Interpolation

```
>>> from scipy.interpolate import interp1d
>>> x = np.linspace(0, 10, num=11,
endpoint=True)
>>> y = np.cos(-x**2/9.0)
>>> f = interp1d(x, y)
>>> f2 = interp1d(x, y, kind='cubic')
>>> xnew = np.linspace(0, 10, num=41,
endpoint=True)


>>> import matplotlib.pyplot as plt
>>> plt.plot(x, y, 'o', xnew, f(xnew), '-',
xnew, f2(xnew), '--')
>>> plt.legend(['data', 'linear', 'cubic'],
loc='best')
>>> plt.show()
```



Scipy - Optimize

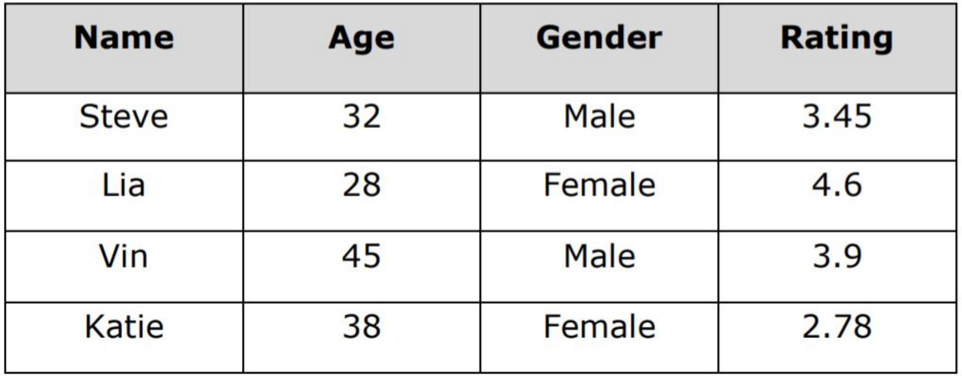
```
# minimize 1st order bessel
# function between 4 and 7
>>> from scipy.special import j1
>>> from scipy.optimize import \
 fminbound

>>> x = r_[2:7.1:.1]
>>> j1x = j1(x)
>>> plot(x,j1x,’-’)
>>> hold(True)
>>> x_min = fminbound(j1,4,7)
>>> j1_min = j1(x_min)
>>> plot([x_min],[j1_min],’ro’)
```



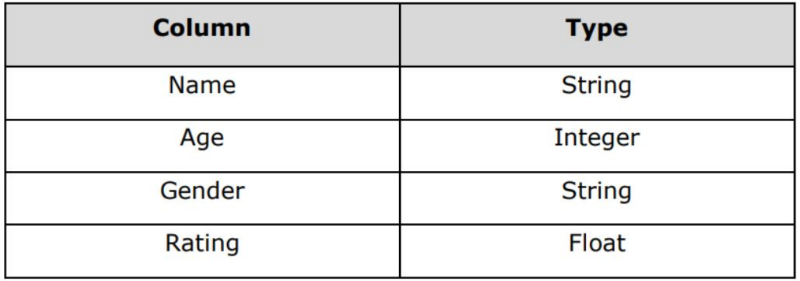
What is Pandas ?

- Pandas is an open source, BSD-licensed library
- High-performance, easy-to-use data structures and data analysis tools
- Offers data structures and operations for manipulating numerical tables, time series and dataframes

Example of Dataframe



Name	Age	Gender	Rating
Steve	32	Male	3.45
Lia	28	Female	4.6
Vin	45	Male	3.9
Katie	38	Female	2.78



Column	Type
Name	String
Age	Integer
Gender	String
Rating	Float

Pandas - Create a dataframe

```
>>>d = {’one ’ : pd . Series ([1. , 2. , 3.] ,
index =[ ’a’, ’b’, ’c’]) ,
’two ’ : pd . Series ([1. , 2. , 3. , 4.] , index
=[ ’a’, ’b’, ’c’, ’d’]) }
>>> df = pd . DataFrame ( d )
>>> df
one two
a 1.0 1.0
b 2.0 2.0
c 3.0 3.0
d NaN 4.0
```

Pandas - Create a data frame and write to a csv file

```
>>> names = ['Bob ','Jessica ','Mary ','John ','Mel ']
>>> births = [968 , 155 , 77 , 578 , 973]
#To merge these two lists together we will use the zip function .
>>> BabyDataSet = list (zip ( names , births ) )
>>> BabyDataSet
[( 'Bob ', 968) , ('Jessica ', 155) , ('Mary ',77) , ('John ', 578) , ('Mel ',
973) ]
```

```
>>> df = pd.DataFrame ( data = BabyDataSet,columns =[ 'Names ', 'Births '])
>>> df.to_csv ('births1880 .csv ', index = False , header = False )
```

Pandas - Import data from csv file

```
>>> df = pd.read_csv ( filename )

#Don ’t treat the first row as a header
>>> df = pd.read_csv ( Location , header = None )

# Provide specific names for the columns
>>> df = pd.read_csv ( Location , names =[ ’Names
’,’Births ’])
```

Pandas - Take a look at the data

```
>>> df.head (2)
Names Births
0 Bob 968
1 Jessica 155
>>> df.tail (2)
Names Births
3 John 578
4 Mel 973
>>> df.columns
Index ([u’Names ’, u’Births ’] , dtype =’object’)
>>> df.values
array ([[ ’Bob ’, 968] ,
       [’Jessica ’, 155] ,
       [’Mary ’, 77] ,
       [’John ’, 578] ,
       [’Mel ’, 973]] , dtype = object )
>>> df.index
Int64Index ([0 , 1 , 2 , 3 , 4] , dtype =’int64 ’)
```

Pandas - Working on the data

```
>>> df [’Births ’]. plot ()

# Maximum value in the data set
>>> MaxValue = df [’Births ’].max()

# Name associated with the maximum value
>>> MaxName = df [’Names ’][ df [’Births ’] ==
df [’Births ’]. max () ].values
```

Pandas - Add a column

```
>>>d = [0 ,1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9]

# Create dataframe
>>> df = pd.DataFrame(d)

# Name the column
>>> df.columns = [’Rev ’]

#Add another one and set the value in that column
>>> df [’NewCol ’] = 5
```

Pandas - Accessing and indexing the data

```
# Perform operations on columns
>>> df [’NewCol ’] = df [’NewCol ’] + 1

# Delete a column
>>>del df [’NewCol ’]

# Edit the index name
>>>i = [’a’,’b’,’c’,’d’,’e’,’f’,’g’,’h’,’i
’,’j’]
>>> df . index = i
```

Pandas - Accessing and indexing the data

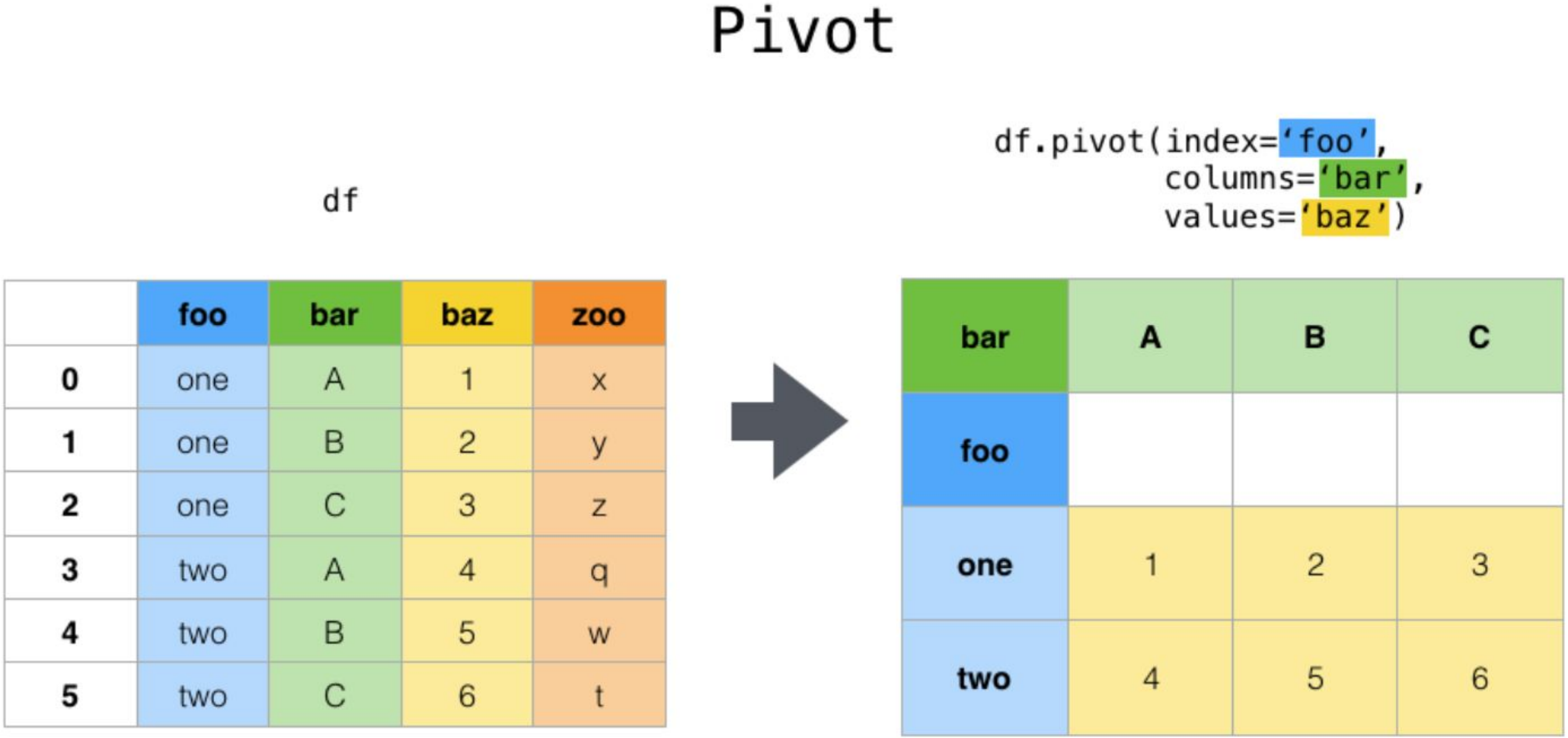
```
# Find based on index value
>>> df . loc ['a']
>>> df . loc ['a':'d']

#Do integer position based indexing
>>> df . iloc [0:3]

# Access using the column name
>>> df ['Rev ']

# Access multiple columns
>>> df [[ 'Rev ', 'test ']]

# Subset the data
>>> df . ix [:3 ,[ 'Rev ', 'test ']]
```

Pandas - Query the data / Apply Functions

Query Data

```
>>> df . query ('one > 0')
one two three
c 0.110718 -0.016733 -0.137009
e 0.153456 0.266369 -0.064127
f 1.709607 -0.424790 -0.792061

>>> df.query ('one > 0 & two > 0')
one two three
e 0.153456 0.266369 -0.064127
```

Pandas - Make Pivot Tables



Pivot

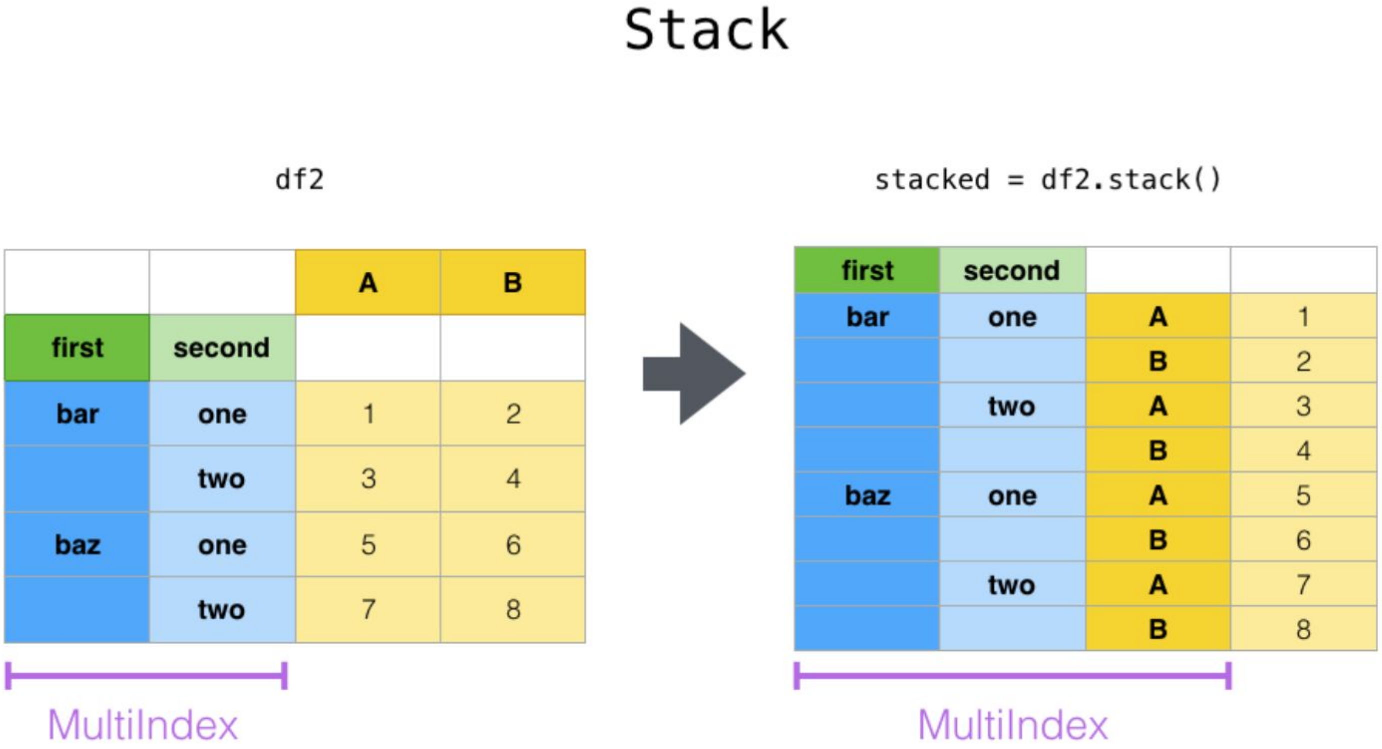
df

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

```
df.pivot(index='foo',
         columns='bar',
         values='baz')
```

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

Pandas - Stacking dataframes



Stack

```
df2
```

		A	B
first	second		
bar	one	1	2
	two	3	4
baz	one	5	6
	two	7	8

```
stacked = df2.stack()
```

first	second		
bar	one	A	1
		B	2
	two	A	3
		B	4
baz	one	A	5
		B	6
	two	A	7
		B	8