Foundations of Computer Science in Python

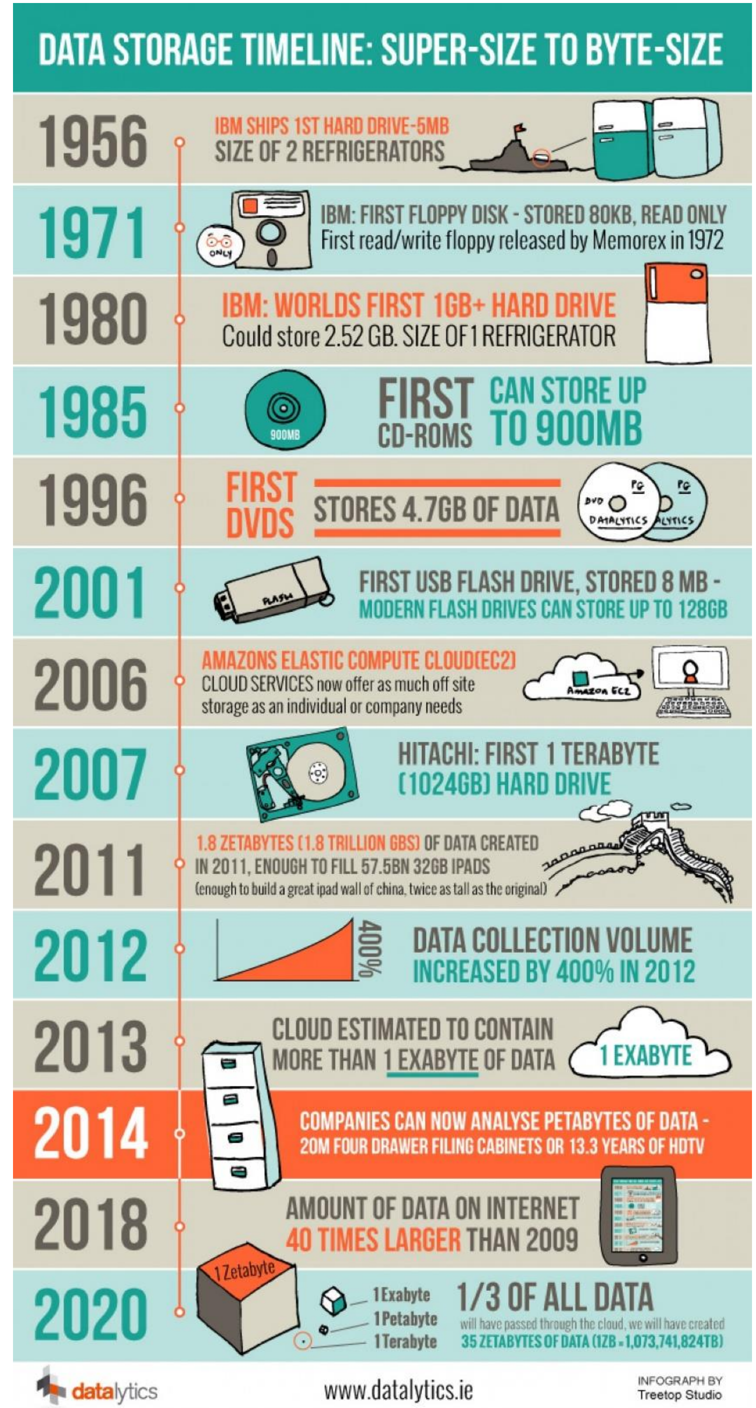
Lecture 12:

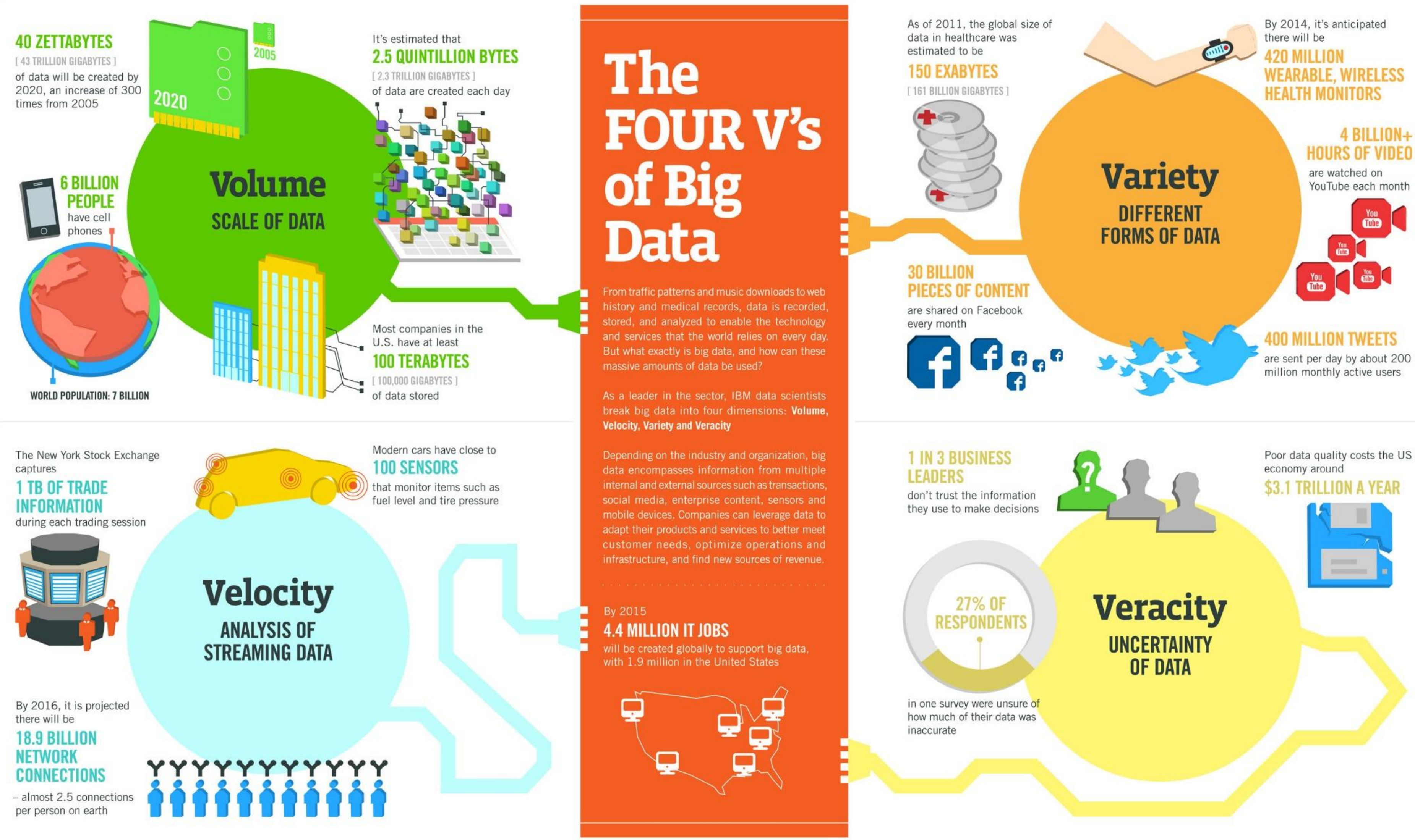
NumPy , SciPy & Matplotlib

The Big Data Revolution

The Big Data Revolution is the result of great advancements in 4 areas:

- Data generation volume
- Storage capacity
- Computing speed
- Internet speed



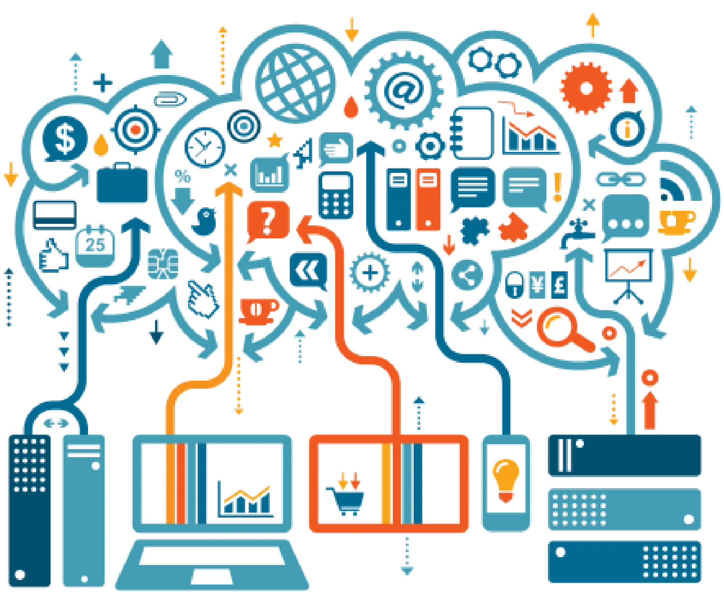


Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

IBM

Today's plan: scientific computing

- Introduction to ***NumPy*** & ***SciPy***
- Plotting using ***MatPlotLib***
- Data Analysis Example



NumPy and SciPy

- **NumPy** and **SciPy** are packages for scientific computing that provide fast pre-compiled mathematical and numerical functions.
- **NumPy** (Numeric Python) - package provides basic routines for manipulating large arrays and matrices of numeric data.
- **SciPy** (Scientific Python) - package extends the functionality of NumPy with a large collection of useful algorithms, e.g.:
  - minimization, regression, and other applied mathematical techniques

NumPy and SciPy

- **NumPy** and **SciPy** are open-source, and therefore provide a *free* **Matlab** alternative.
  - **Matlab** – widely used, but expensive

- **NumPy** and **SciPy** are popular among scientists, researchers and engineers who want to apply various mathematical methods on large datasets.

Read more here: *http://docs.scipy.org/doc/numpy/reference/*

Importing the required modules

- The packages we need are already included within the installation of Anaconda.
- Make sure you import any needed package at the beginning of your program in order to use its classes….
- For example:

```
import numpy as np, matplotlib, scipy
```

Alias

NumPy’s main object is the **Array**

- NumPy’s main object is the homogeneous multidimensional **array**.
- **Array**: It is a table of elements (usually numbers), all of the **same type**, indexed by a tuple of positive integers.
- In Numpy dimensions are called ***axes***.
- ***Rank*** *:the number of axes (=dimensions)*
    - *A **vector** is an array of rank 1*
    - *A 2D **matrix** is an array of rank 2.*

Python list vs. NumPy’s Array

- Both used to store any data type, can be indexed and iterated through
- Arrays have to be declared while lists don't (part of Python's syntax)
- So what are Arrays good for?
  - Performing arithmetic functions on the elements
  - Storing data more compactly and effectively

The Array object - Creation

List to initialize

```
>>> import numpy as np
>>> a = np.array([0, 1, 2])
# 1D array of size 3
>>> a
array([0,1,2])
```

0	1	2

Nested list

```
>>> b = np.array([[0, 1, 2], [3, 4, 5]])
# 2D array of size 2x3
>>> b
array([[0,1,2],
       [3,4,5]])
```

0	1	2
3	4	5

Creating an array of zeros or ones

```
>>> np.zeros(7, dtype=int)
array([0, 0, 0, 0, 0, 0, 0])

>>> np.ones((2,3), dtype=float)
 array([[ 1., 1., 1.],
        [ 1., 1., 1.]])
```

Think: ***why*** is this useful?

Creating arrays containing number sequences

```
>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.arange(2,7)
array([2, 3, 4, 5, 6])
>>> np.arange(2,7,2)
# start, end (exclusive), step
array([2, 4, 6])
```

Like range(10), but returns a **numpy array** object

Creating an Array of a **specific** type

```
>>> a = np.array(range(10), dtype=float)
>>> a
array([ 0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
>>> a.dtype
dtype('float64')
>>> type(a)
numpy.ndarray
```

Additional ways to create arrays in NumPy:
http://docs.scipy.org/doc/numpy-noitaerc-yarra-senituor#lmth.noitaerc-yarra.senituor/ecnerefer/1.10.1

Creating arrays containing number sequences

```
>>> print(np.arange(1, 5.5, 0.5))
# start, end (exclusive), step
[1.  1.5 2.  2.5 3.  3.5 4.  4.5 5. ]

# slicing is similar to standard lists
>>> print(np.arange(1, 5.5, 0.5)[3:1:-1])
[2.5 2. ]

# start, end, number of points
>>> print(np.linspace(1, 5, 9))
[1.  1.5 2.  2.5 3.  3.5 4.  4.5 5. ]

# Arrays with random values between 0 and 1
>>> print(np.random.random(5))
[0.16857547 0.31380802 0.00347811 0.87055124 0.23589622]
```

Array – Attributes & Methods

```
>>> mat = np.array([[0, 1, 2], [3, 4, 5]])
# Creates a 2 x 3 array
>>> mat
array([[0,1,2],
       [3,4,5]])
>>> mat.ndim  #number of dimensions
2
>>> mat.shape  #dimension sizes
(2, 3)
>>> len(mat) # returns the size of the first dimension
2
>>> mat.T           #Transpose
array([[0, 3],
       [1, 4],
       [2, 5]])
```

0	1	2
3	4	5

No parentheses

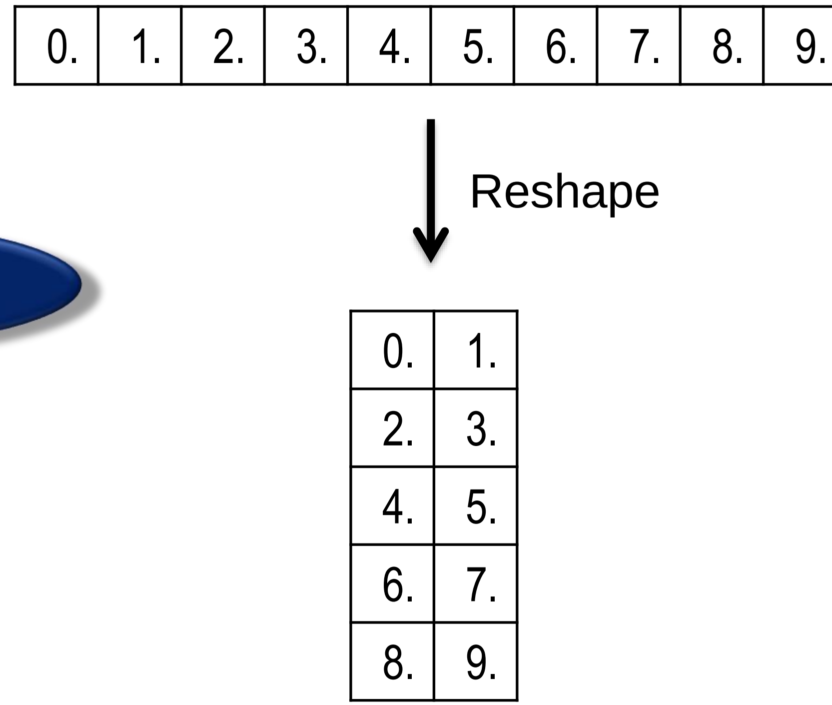
Same as lists

0	3
1	4
2	5

Reshaping an Array

```
>>> a = np.arange(10.0)
>>> a
array([ 0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
>>> a = a.reshape((5, 2))
>>> a
array([[ 0., 1.],
       [ 2., 3.],
       [ 4., 5],
       [ 6., 7.],
        [ 8., 9.]])
>>> a.shape
(5, 2)
```

Creates a new array



Indexing and slicing: similar to lists

```
>>> a = np.diag(np.arange(3))
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 2]])
>>> a[1, 1]
1
>>> a[2, 1] = 10
>>> a[1, :]  # Row at index 1    a[1] also works
array([ 0, 1, 0])
>>> a[:, 1]  # Column at index 1
array([ 0,  1, 10])
>>> a[2, 1:]
array([10,  2])
```

Diagonal matrix:

All non-diag elements are 0

Slicing: specify which rows/columns to take by A[rows, columns]

Indexing and slicing: matrices

How do we create this array?

```
array([[ 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]])
```

```
In [65]: a=np.arange(0,100).reshape(10,10)
         a=a[:6,:6]
         a
```

Indexing and slicing: matrices

Let’s select the colored areas

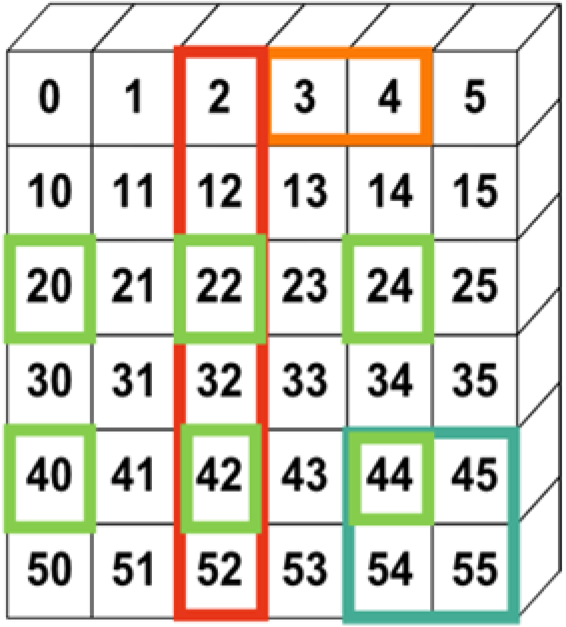
```
>>> a[0,3:5]
array([3,4])
```

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

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

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

Also: `a[2::2,0::2]`



Example

```
>>> x = np.arange(10)
>>> x[2]
2
>>> x[-2]
8
>>> x.shape = (2,5) # Setting x's dimensions to (2,5)
>>> x[1,3]
8
>>> x[1,-1]
9
>>> x[0]
array([0, 1, 2, 3, 4])
```

Array Arithmetic

- Arithmetic operators on arrays apply *elementwise*.
- A new array is created and filled with the result.
- Very useful for vector calculations

```
>>> x = np.array([1, 5, 2])
>>> y = np.array([7, 4, 1])
>>> x + y
array([8, 9, 3])
>>> x * y      # element by element multiplication!
# Use np.dot(x,y) for matrix multiplication.
array([ 7, 20,  2])
>>> x - y
array([-6,  1,  1])
>>> x // y
array([0, 1, 2])
>>> x % y
array([1, 1, 0])
```

Note that here, x and y have the same size

Broadcasting

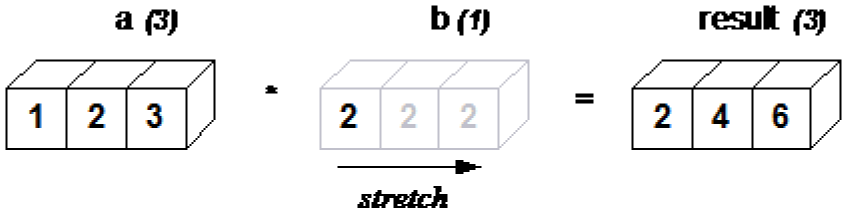
- NumPy operations are usually done element-by-element which requires two arrays to have exactly the same shape:

```
>>> a = np.array([1.0,2.0,3.0])
>>> b = np.array([2.0,2.0,2.0])
>>> a * b
array([ 2.,  4.,  6.])
```

- But this will also work:

```
>>> a = np.array([1.0,2.0,3.0])
>>> b = 2.0
>>> a * b
array([ 2.,  4.,  6.])
```

Same result

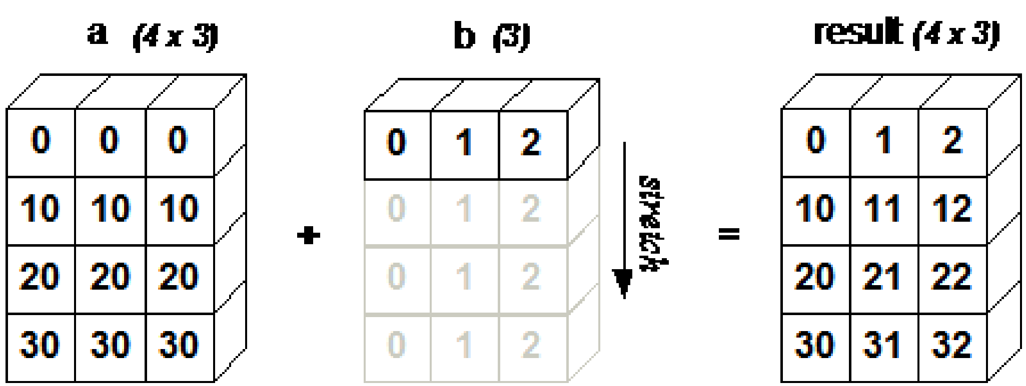


Broadcasting

- One array can be “stretched” along the auxiliary axis, to fit the other

```
a = np.array([[ 0.0, 0.0, 0.0],
              [10.0,10.0,10.0],
              [20.0,20.0,20.0],
              [30.0,30.0,30.0]])
b=np.array([0.0,1.0,2.0])
a+b
```

```
array([[ 0.,  1.,  2.],
       [10., 11., 12.],
       [20., 21., 22.],
       [30., 31., 32.]])
```



Broadcasting

- Be careful of dimension mismatch

```
a=np.array([1,2,3,4])
a+b
```

```
---------------------------------------------------------
--
ValueError                                           Tra
t)
<ipython-input-97-36ab9968ed7b> in <module>
      1 a=np.array([1,2,3,4])
----> 2 a+b

ValueError: operands could not be broadcast t
)
```

Comparisons and Boolean operations

```
>>> a = np.random.randint(0, 11, 15)
array([ 4,  9,  1,  7,  0,  9,  8, 10,  1,  0,  7,  7,  9,  5,  1])
```

`random.randint(low, high=None, size=None, dtype=int)`

return random integers from low (inclusive) to high (exclusive).
Documentation - search **`np.random.randint`** in google

Comparisons and Boolean operations

```
>>> a = np.array([1, 4, 3, 5, 2, 3])
>>> b = np.array([1, 2, 4, 5, 2, 1])

>>> a==b
array([ True, False, False, True, True, False], dtype=bool)

        # array of Booleans, elementwise comparison

>>> comp=a==b
```

Comparisons and Boolean operations

```
>>> a = np.random.randint(0, 11, 15)
>>> a
array([0, 9, 5, 3, 10, 10, 8, 4, 1, 5, 1, 6, 5, 10, 5])
>>> b = np.arange(15)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
>>> comp1 = a==b       # array of Booleans, elementwise comparison
>>> comp1
array([ True, False, False,  True, False, False, False, False,
False, False, False, False, False, False, False])
>>> comp1.any()
True
>>> comp1.all()
False
>>> comp1.nonzero()
(array([0, 3], dtype=int64),)
>>> comp1.sum()
2
```

Any is true?

Are all true?

Get the True indices

How many TRUEs are in comp1?

Applying a logical operation to an Array

```
a = np.random.randint(0, 20, 15)
a
```

```
array([12,  6,  1, 12, 15,  2, 17, 11,  4, 14, 16, 12,  3,  5, 11])
```

```
a%3==0
```

What will be the result?
What will be the type of the new object?

```
array([ True,  True, False,  True,  True, False, False, False, False,
       False, False,  True,  True, False, False])
```

Using logic for indexing

```
a = np.random.randint(0, 20, 15)
a
```

```
array([12,  6,  1, 12, 15,  2, 17, 11,  4, 14, 16, 12,  3,  5, 11])
```

```
a[a%3==0]
```

```
array([12,  6, 12, 15, 12,  3])
```

Extremely useful –
for getting only the array elements you want

Also for assignments

```
a[a%3==0]=99
a
```

```
array([99, 99,  1, 99, 99,  2, 17, 11,  4, 14, 16, 99, 99,  5, 11])
```

Indexing with a list

```
a[a%3==0]=99
a
```

```
array([99, 99,  1, 99, 99,  2, 17, 11,  4, 14, 16, 99, 99,  5, 11])
```

```
a[[0,1,7,2]]
```

Note the [[

```
array([99, 99, 11,  1])
```

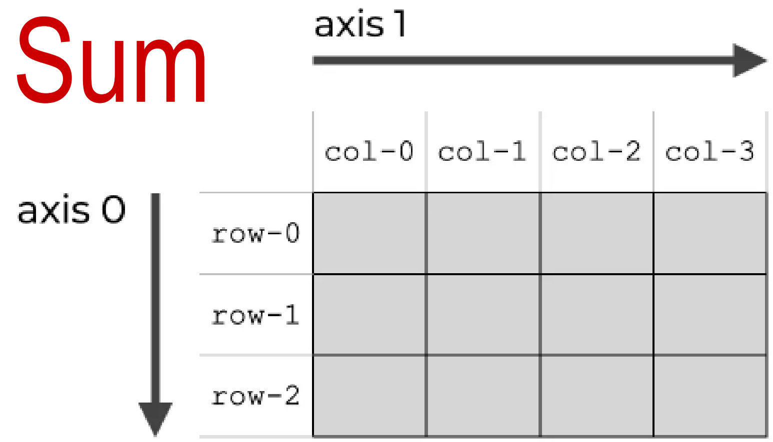
Fancy indexing: Logical indexing

```
>>> a = np.random.randint (0, 20, 15)
array([10, 3, 8, 0, 19, 10, 11, 9, 10, 6, 0, 19, 12, 7, 14])
>>> (a % 3 == 0)
array([False, True, False, True, False, False, False, True,
   False,True, True, False, True, False, False], dtype=bool)
>>> mask = (a % 3 == 0)
# extract sub-array: this is a copy!
>>> extract_from_a = a[mask]
array([ 3, 0, 9, 6, 0, 12])
# change sub-array
>>> a[mask] = -1
array([10, -1, 8, -1, 19, 10, 11, -1, 10, -1, -1, 19, -1, 7, 14])
```

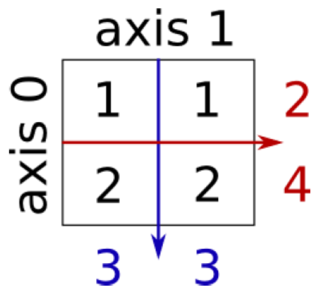
Fancy indexing II

```
# Indexing with array/list of integers
>>> a = np.arange(0, 100, 10)
>>> a
array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
>>> a[[2, 3, 2, 4, 2]]
array([20, 30, 20, 40, 20])
# note: [2, 3, 2, 4, 2] is a Python list
```

Reductions: Sum




```
>>> x = np.array([[1, 1], [2, 2]])
>>> x
array([[1, 1],
       [2, 2]])
>>> x.sum() # works on the entire matrix
6

>>> x.sum(axis=0) # sum the columns along axis 0
array([3, 3])
>>> x[:, 0].sum(), x[:, 1].sum()
(3, 3)

>>> x.sum(axis=1) # sum the rows along axis 1
array([2, 4])
>>> x[0, :].sum(), x[1, :].sum()
(2, 4)
```



Also works with x.min, x.max, x.mean etc.

Sorting along an axis

```
>>> a = np.array([[4, 3, 5], [1, 2, 1]])
>>> b = np.sort(a, axis=1)
>>> b
array([[3, 4, 5],
       [1, 1, 2]])
```

axis 1

axis 0			
	4	3	5
	1	2	1

```
>>> a.sort(axis=1)
>>> a
array([[3, 4, 5],
       [1, 1, 2]])
```

a.sort does not return a value

argsort

Use ***argsort*** to retrieve the indices that would sort the array

```
>>> a = np.array([9, 8, 6, 7])
>>> a
array([9, 8, 6, 7])
>>> ids = np.argsort (a)
>>> ids
array([2, 3, 1, 0], dtype=int64)
>>> a[ids]
array([6, 7, 8, 9])
```

```
>>> a = np.array([[4, 3, 5], [1, 2, 1]])
>>> b = np.argsort(a, axis=1)
>>> b
array([[1, 0, 2],
       [0, 2, 1]])
```

Array stacking

- How to concatenate arrays?
  - a+b : doesn't work…
  - a.append(b) : bad
- Solution: hstack

```
a = np.array([1,2,3])
b = np.array([2,3,4])
a
```

```
array([1, 2, 3])
```

Tuple or list

```
np.hstack([a,b])
```

```
array([1, 2, 3, 2, 3, 4])
```

```
c=np.array([[1],[2],[3]])
d=np.array([[2],[3],[4]])
c
```

```
array([[1],
       [2],
       [3]])
```

```
c=np.array([[1],[2],[3]])
d=np.array([[2],[3],[4]])
np.hstack([c,d])
```

```
array([[1, 2],
       [2, 3],
       [3, 4]])
```

Vertical Stacking

```
a = np.array([1,2,3])
b = np.array([2,3,4])
a
```

```
array([1, 2, 3])
```

```
c=np.array([[1],[2],[3]])
d=np.array([[2],[3],[4]])
c
```

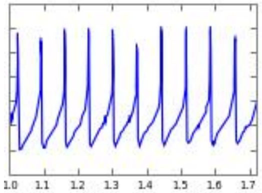
```
array([[1],
       [2],
       [3]])
```

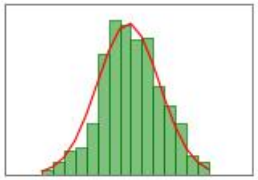
```
np.vstack((a,b))
```

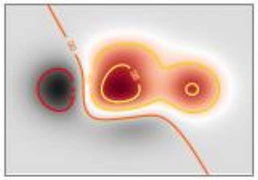
```
array([[1, 2, 3],
       [2, 3, 4]])
```

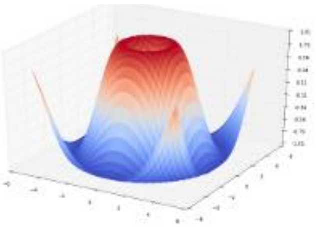
```
np.vstack((c,d))
```

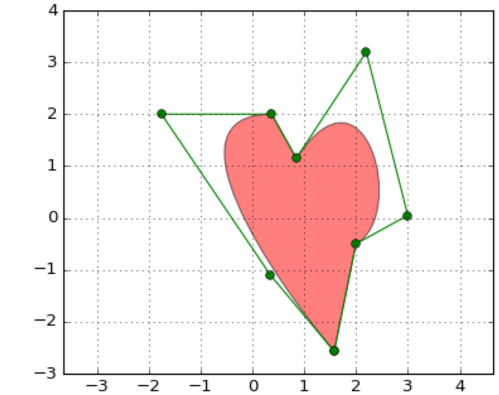
```
array([[1],
       [2],
       [3],
       [2],
       [3],
       [4]])
```







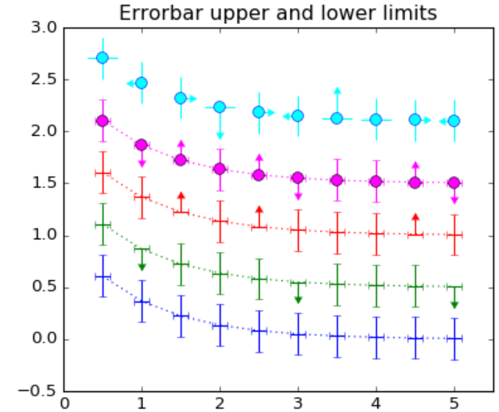


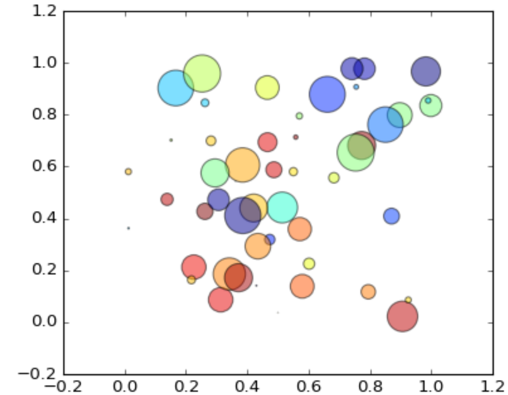


Plotting

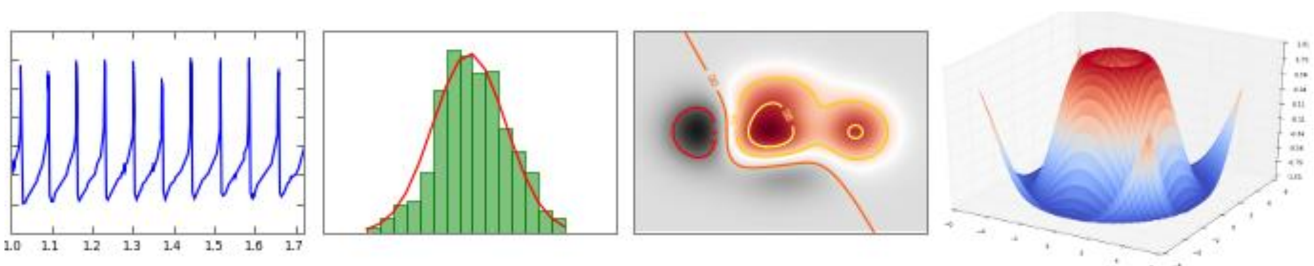
matplotlib

"A plot is worth a thousand words"



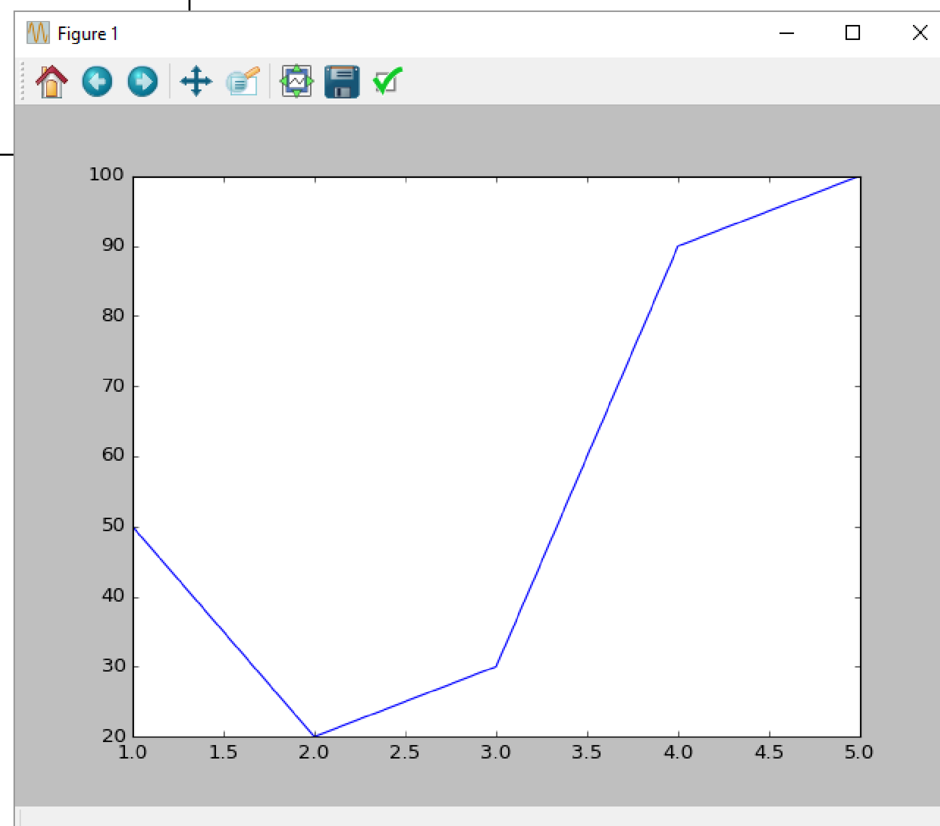


- **matplotlib** is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and in an interactive manner.
- You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code.
- Check this out: http://matplotlib.org/gallery.html



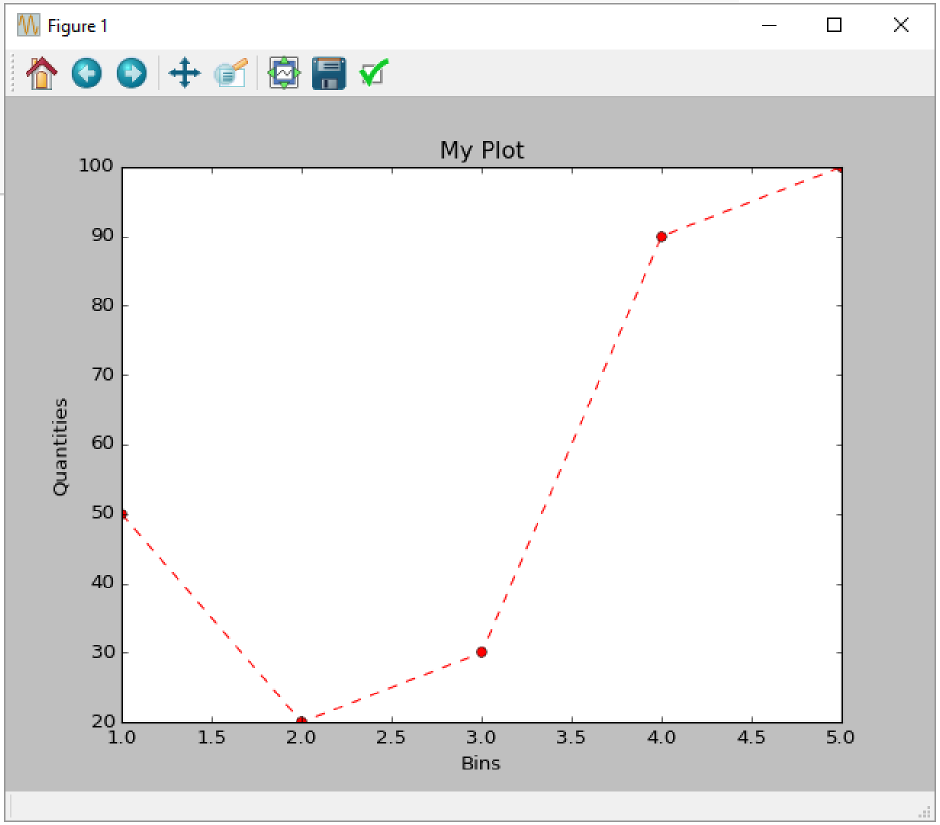
Example 1

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

x = np.array([5 ,4 ,3 ,2 ,1])
y = np.array([100 ,90 ,30 ,20 ,50])
plt.plot (x,y)
plt.show()
```



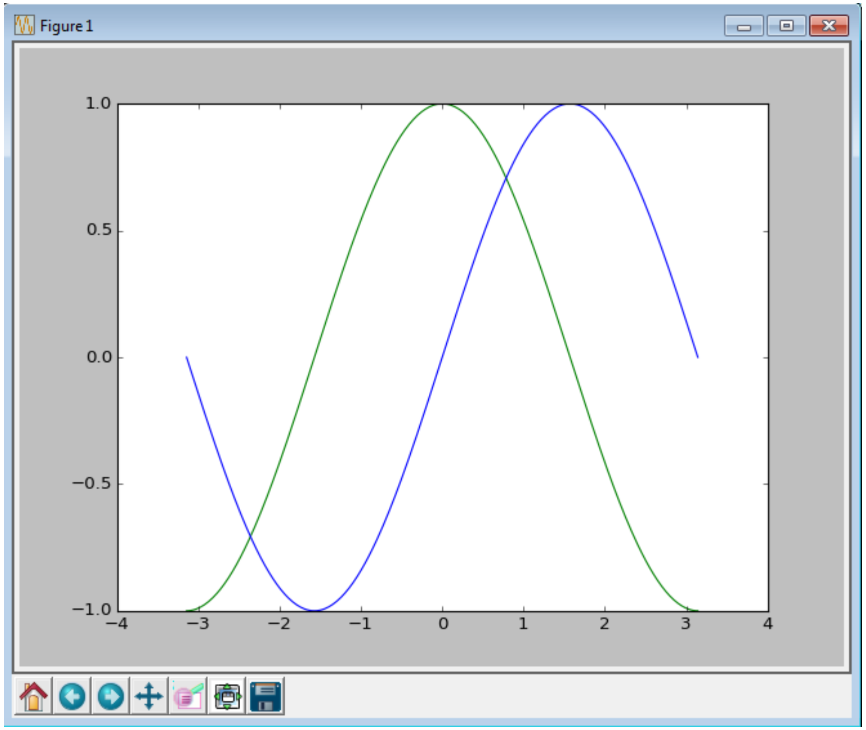
Different line styles

```
x=np.array([1,2,3,4,5])
y=np.array([50,20,30,90,100])
plt.plot(x, y, color="red", ls='--', marker='o')
plt.xlabel('Bins')
plt.ylabel('Quantities')
plt.title('My Plot')
plt.show()
```



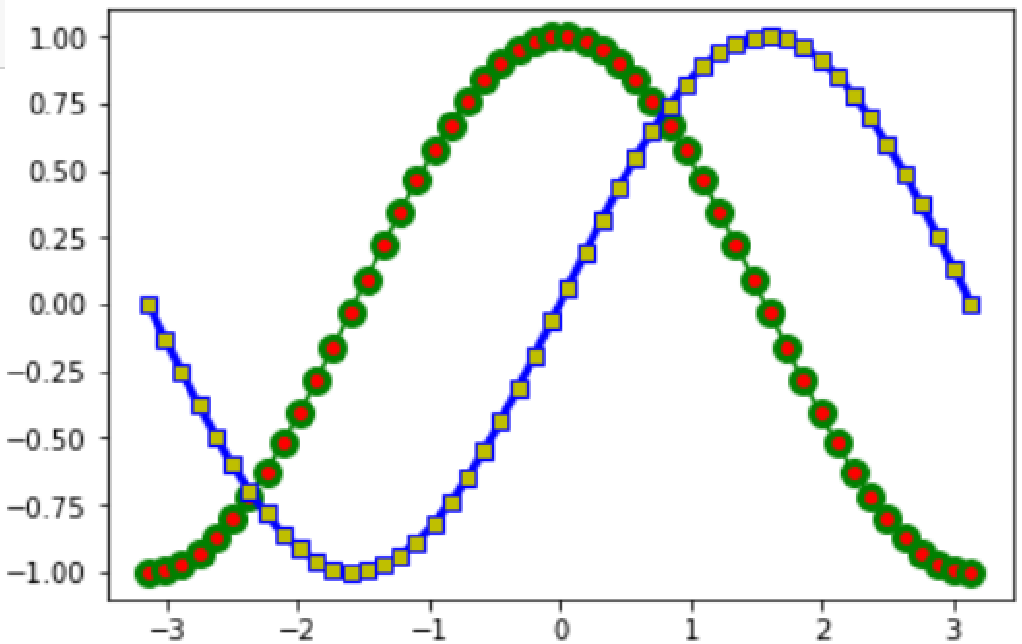
Example 2 – Sin and Cos

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

x = np.linspace(-np.pi, np.pi, 256)
y, z = np.cos(x), np.sin(x)

plt.plot(x, y, color = 'green')
plt.plot(x, z, color = 'blue')
plt.show()
```



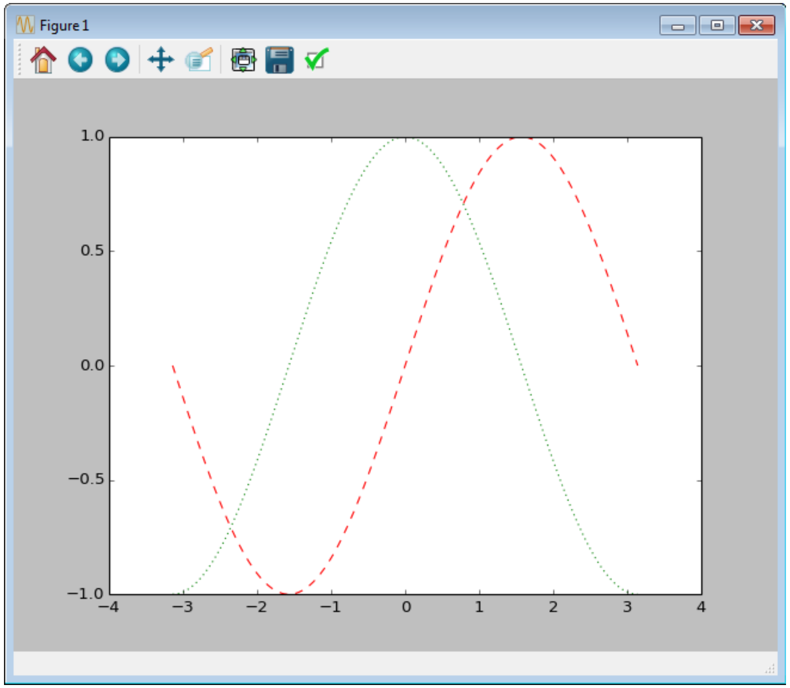
Example 2b- Sin and Cos with Style

```
x = np.linspace(-np.pi, np.pi)
y, z = np.cos(x), np.sin(x)
plt.plot(x, y, color = 'green', ls='-',
         marker='o',markersize=8,
         markeredgecolor="green", markeredgewidth=3,
         markerfacecolor="red")
plt.plot(x, z, color = 'b', ls='-',lw=3,marker='s',
         markerfacecolor="y")

plt.show()
```



Example 2c

```
x = np.linspace(-np.pi, np.pi)
y, z = np.cos(x), np.sin(x)
plt.plot(x, y, 'g:')
plt.plot(x, z, 'r--')
plt.show()
```



character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker

Example:
Simple data analysis using Numpy
Processing medical data

Real data analysis example

- Look at the file inflammation-01.csv
  - (download the lecture's code)
- Tables as **C**SV files
  - Text files
  - Each row holds the same number of columns
  - Values are separated by **commas**

```
0,0,1,3,1,2,4,7,8,3,3,3,10,5,7,4,7,7,12,18,6,13,11,11,7,7,4,6,8,8,4,4,5,7,3,4,2,3,0,0
0,1,2,1,2,1,3,2,2,6,10,11,5,9,4,4,7,16,8,6,18,4,12,5,12,7,11,5,11,3,3,5,4,4,5,5,1,1,0,1
0,1,1,3,3,2,6,2,5,9,5,7,4,5,4,15,5,11,9,10,19,14,12,17,7,12,11,7,4,2,10,5,4,2,2,3,2,2,1,1
0,0,2,0,4,2,2,1,6,7,10,7,9,13,8,8,15,10,10,7,17,4,4,7,6,15,6,4,9,11,3,5,6,3,3,4,2,3,2,1
0,1,1,3,3,1,3,5,2,4,4,7,6,5,3,10,8,10,6,17,9,14,9,7,13,9,12,6,7,7,9,6,3,2,2,4,2,0,1,1
```

Real data analysis example

- Look at the file inflammation-01.csv
- The data: inflammation level in patients after a treatment (CSV) format
  - Each row holds information for a single patient
  - The columns represent successive days.
  - The first few lines in the file:

Days

Patients

```
0,0,1,3,1,2,4,7,8,3,3,3,10,5,7,4,7,7,12,18,6,13,11,11,7,7,4,6,8,8,4,4,5,7,3,4,2,3,0,0
0,1,2,1,2,1,3,2,2,6,10,11,5,9,4,4,7,16,8,6,18,4,12,5,12,7,11,5,11,3,3,5,4,4,5,5,1,1,0,1
0,1,1,3,3,2,6,2,5,9,5,7,4,5,4,15,5,11,9,10,19,14,12,17,7,12,11,7,4,2,10,5,4,2,2,3,2,2,1,1
0,0,2,0,4,2,2,1,6,7,10,7,9,13,8,8,15,10,10,7,17,4,4,7,6,15,6,4,9,11,3,5,6,3,3,4,2,3,2,1
0,1,1,3,3,1,3,5,2,4,4,7,6,5,3,10,8,10,6,17,9,14,9,7,13,9,12,6,7,7,9,6,3,2,2,4,2,0,1,1
```

Reading the data using NumPy

```
import numpy as np
```

```
fname = "inflammation-01.csv"
data = np.loadtxt(fname, delimiter=',')
print(data)
```

```
[[0. 0. 1. ... 3. 0. 0.]
 [0. 1. 2. ... 1. 0. 1.]
 [0. 1. 1. ... 2. 1. 1.]
 ...
 [0. 1. 1. ... 1. 1. 1.]
 [0. 0. 0. ... 0. 2. 0.]
 [0. 0. 1. ... 1. 1. 0.]]
```

Properties of the data

```
# properties of the data
print(type(data))
print(data.shape)
print("first value in data",data[0,0])
print("middle value in data:", data[30, 20])
```

```
<class 'numpy.ndarray'>
(60, 40)
first value in data 0.0
middle value in data: 13.0
```

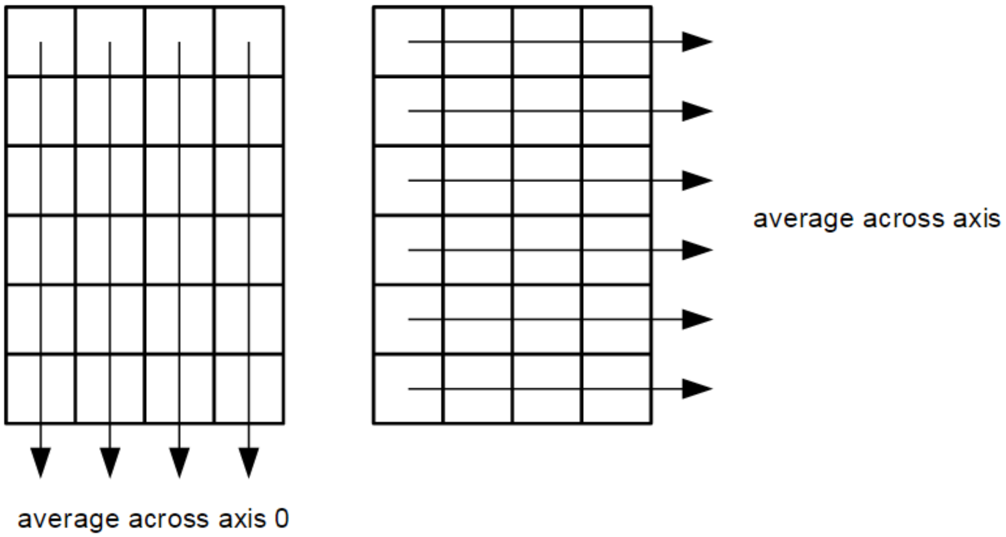
Tasks

- Remove the first and last 10 days
- Plot the average, min, and max inflammation score per day

Data trimming and plots

```
# remove the 10 first and last days
n,m = data.shape
data = data[:,10:(m-10)]
```

How can we get the average/min/max of each day?



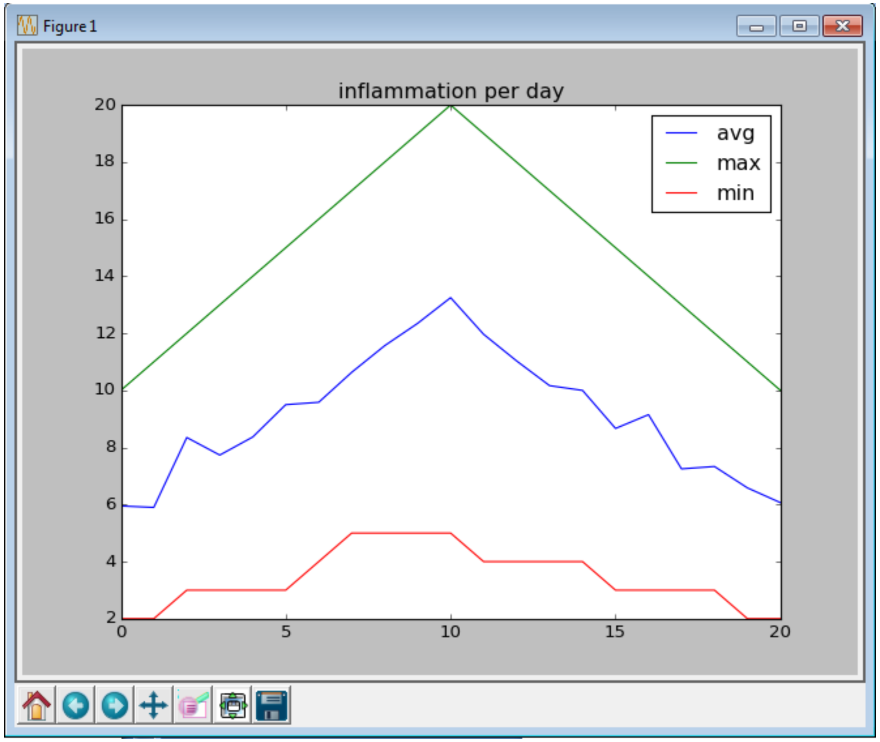
Removing first and last 10 days

```
n,m = data.shape
data = data[:,10:(m-10)]
data
```

```
array([[ 3.,  3., 10., ...,  8.,  8.,  4.],
       [10., 11.,  5., ..., 11.,  3.,  3.],
       [ 5.,  7.,  4., ...,  4.,  2., 10.],
       ...,
       [ 6.,  5.,  6., ...,  5.,  6.,  8.],
       [ 9., 10.,  8., ..., 10., 11., 10.],
       [ 9.,  3.,  3., ...,  5.,  2.,  8.]])
```

Inflammation values per day

```
import matplotlib.pyplot as plt
plt.plot(data.mean(axis=0), label='avg')
plt.plot(data.max(axis=0), label='max')
plt.plot(data.min(axis=0), label='min')
plt.title('inflammation per day')
plt.legend()
```



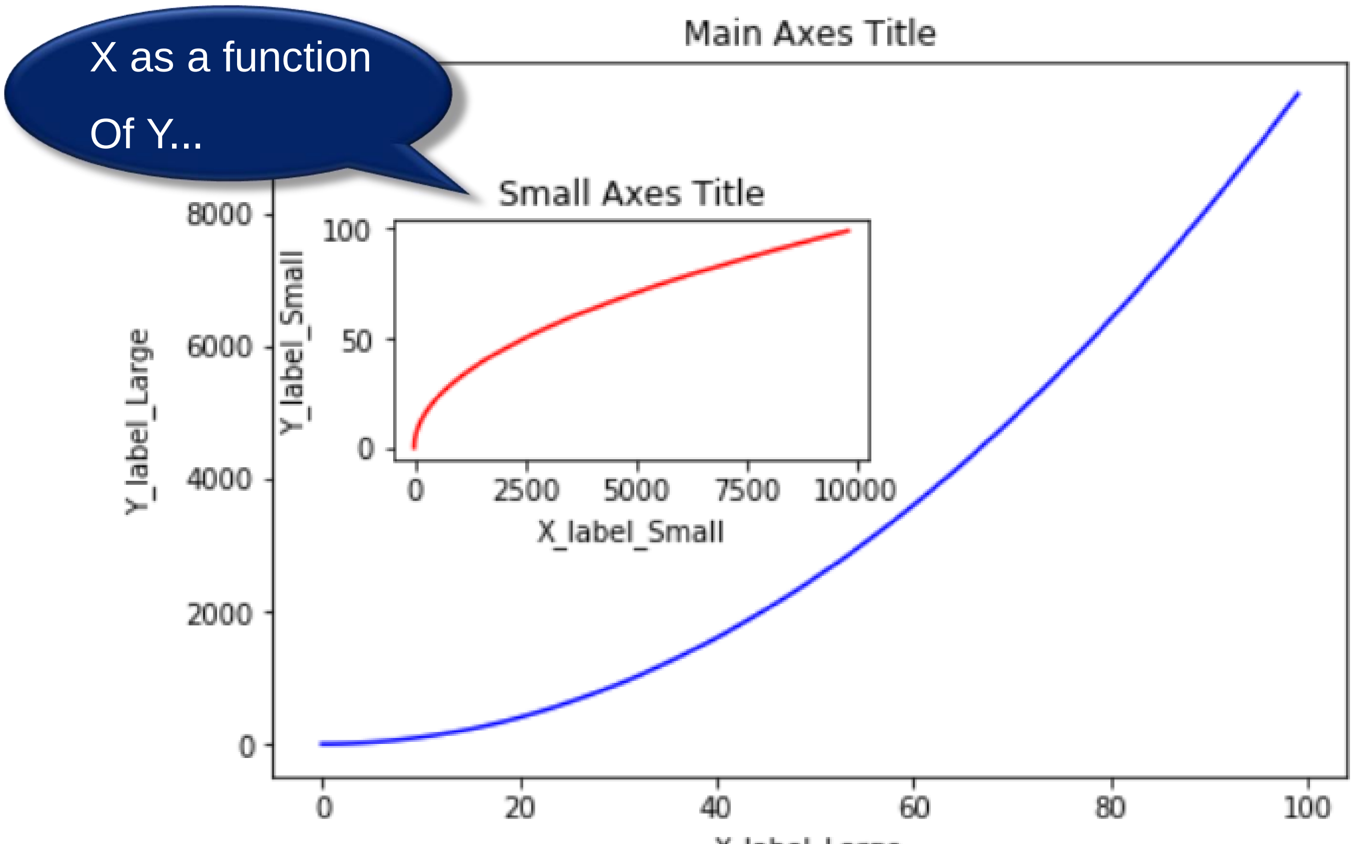
Supplement: Multiple Axes in same Figure

```
x=np.arange(100)
y=x**2
# Creates blank canvas
fig = plt.figure()

outer = fig.add_axes([0.1, 0.1, 0.9, 0.9]) # main axes
inner = fig.add_axes([0.2, 0.5, 0.4, 0.3]) # "inner" axes
```

```
# Larger Figure Axes 1
outer.plot(x, y, 'b')
outer.set_xlabel('X_label_Large')
outer.set_ylabel('Y_label_Large')
outer.set_title('Main Axes Title')

# Insert Figure Axes 2
inner.plot(y, x, 'r')
inner.set_xlabel('X_label_Small')
inner.set_ylabel('Y_label_Small')
inner.set_title('Small Axes Title');
```

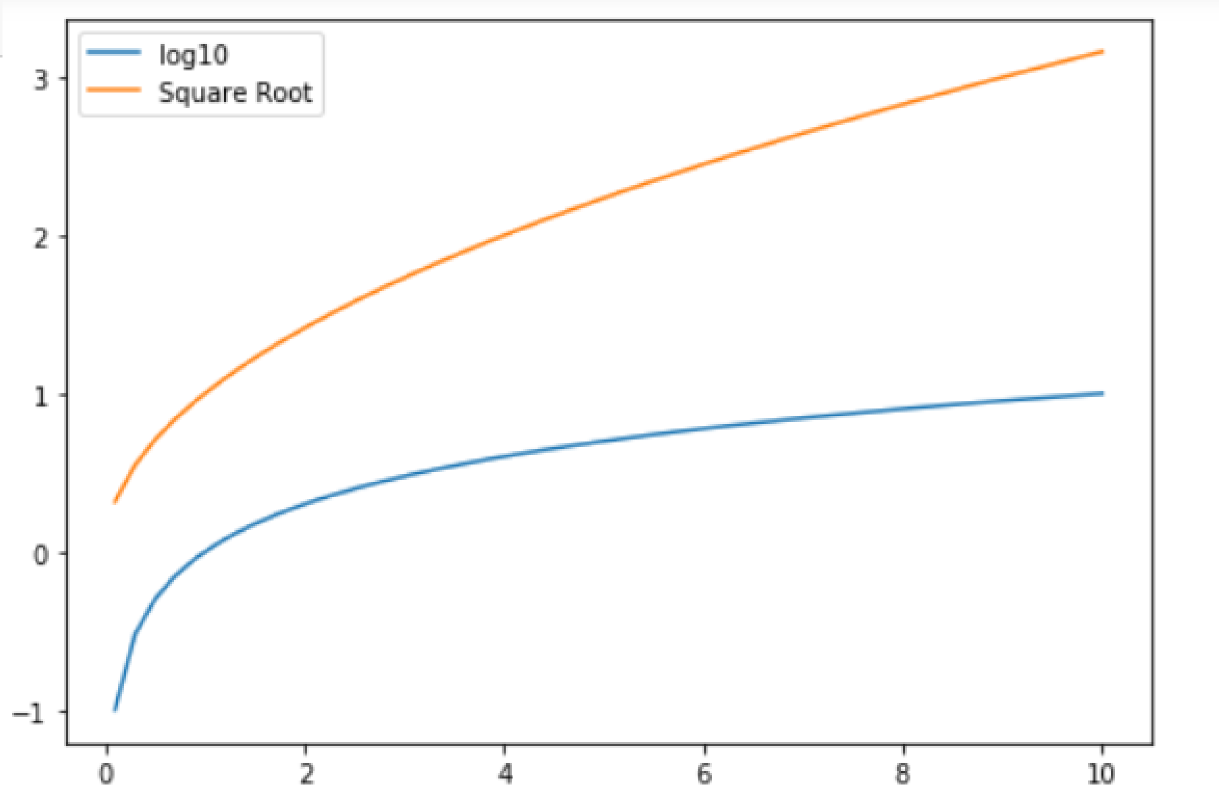
```python
# Larger Figure Axes 1
outer.plot(x, y, 'b')
outer.set_xlabel('X_label_Large')
outer.set_ylabel('Y_label_Large')
outer.set_title('Main Axes Title')

# Insert Figure Axes 2
inner.plot(y, x, 'r')
inner.set_xlabel('X_label_Small')
inner.set_ylabel('Y_label_Small')
inner.set_title('Small Axes Title');
```

Result



Legends & Labels

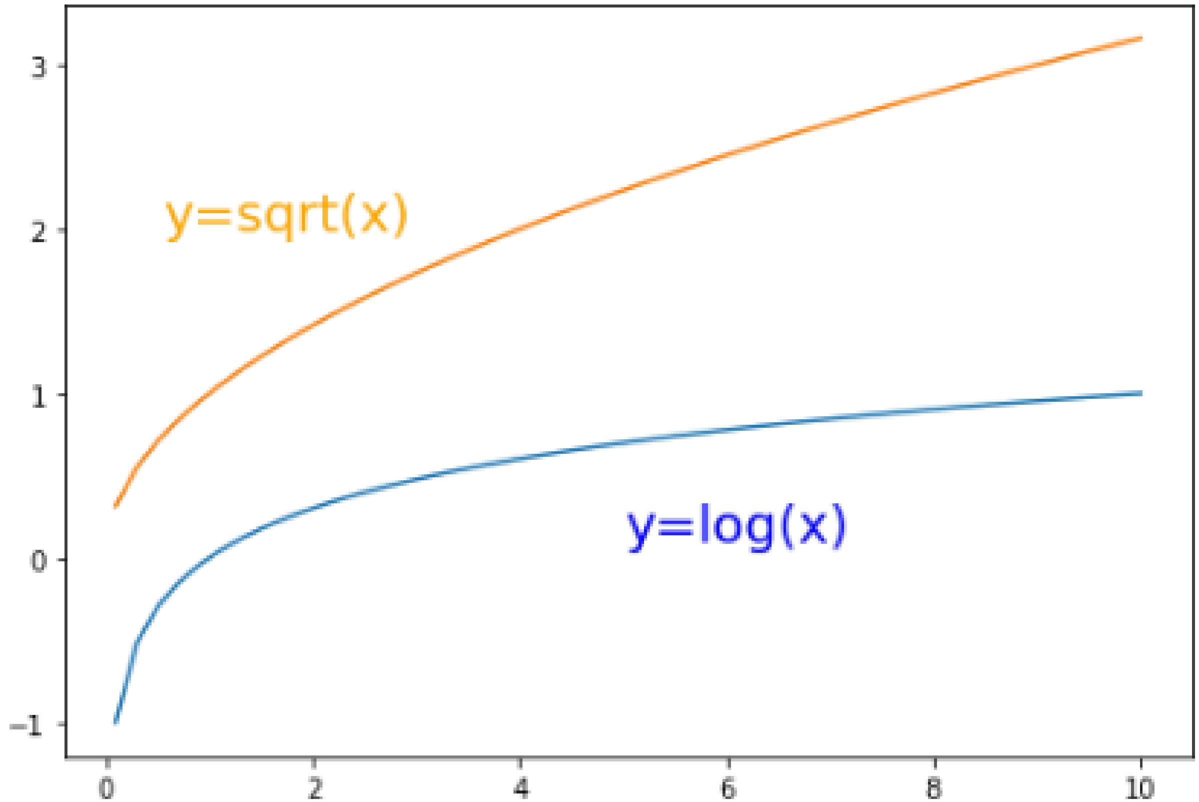
```
fig = plt.figure()
x=np.linspace(0.1,10.0,num=50)
ax = fig.add_axes([0,0,1,1])
import math
ax.plot(x, np.log10(x), label="log10")
ax.plot(x, x**0.5, label="Square Root")
ax.legend()
```



Values on Axes

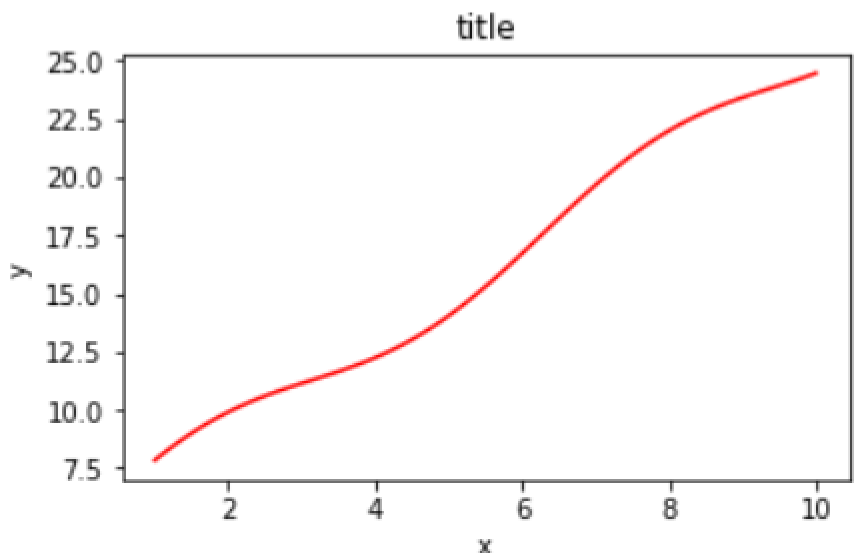
Annotating

```
ax.text(0.55, 2, "y=sqrt(x)", fontsize=20, color="orange")
ax.text(5, 0.1, "y=log(x)", fontsize=20, color="blue")
```



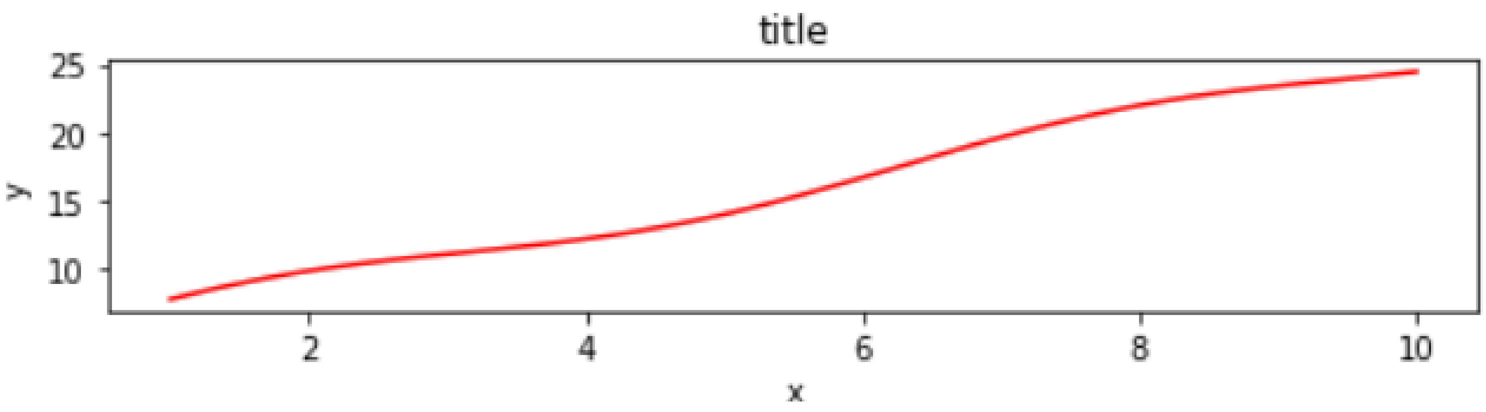
Controlling Figure size

```
fig, axes = plt.subplots(figsize=(5,3))
x=np.linspace(1.0,10.0,num=100)
y=2*x+np.sin(x)+5
axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```



Controlling Figure size

```
fig, axes = plt.subplots(figsize=(8,1.5))
x=np.linspace(1.0,10.0,num=100)
y=2*x+np.sin(x)+5
axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```



Saving Figures

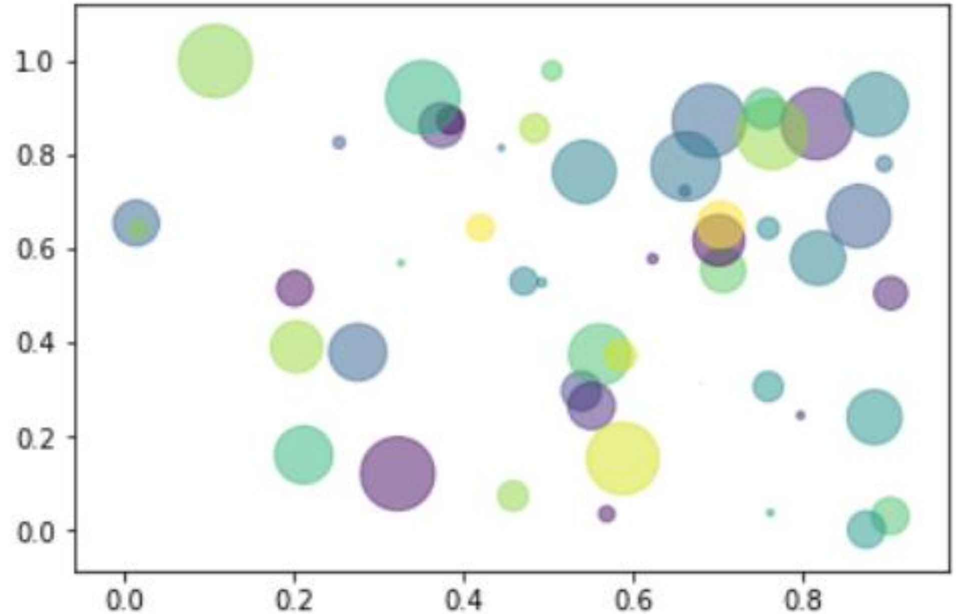
Multiple picture formats supported, including PDF

```
fig.savefig("filename1.png")
```

```
fig.savefig("filename2.pdf", dpi=150)
```

Some Useful Plots: **Scatter**

```
import random

N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N))**2

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```



Some Useful Plots: **Histogram**

Generate 100 values from Normal distribution (with randn)

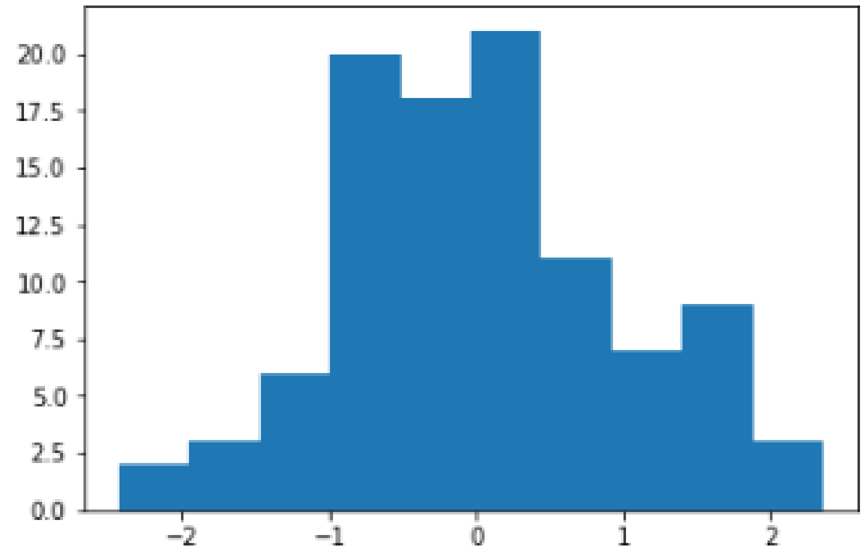
Insert them into 10 bins and plot

Remember

bucket sort

```
import random
vals = np.random.randn(100)
plt.hist(vals, bins=10)
```

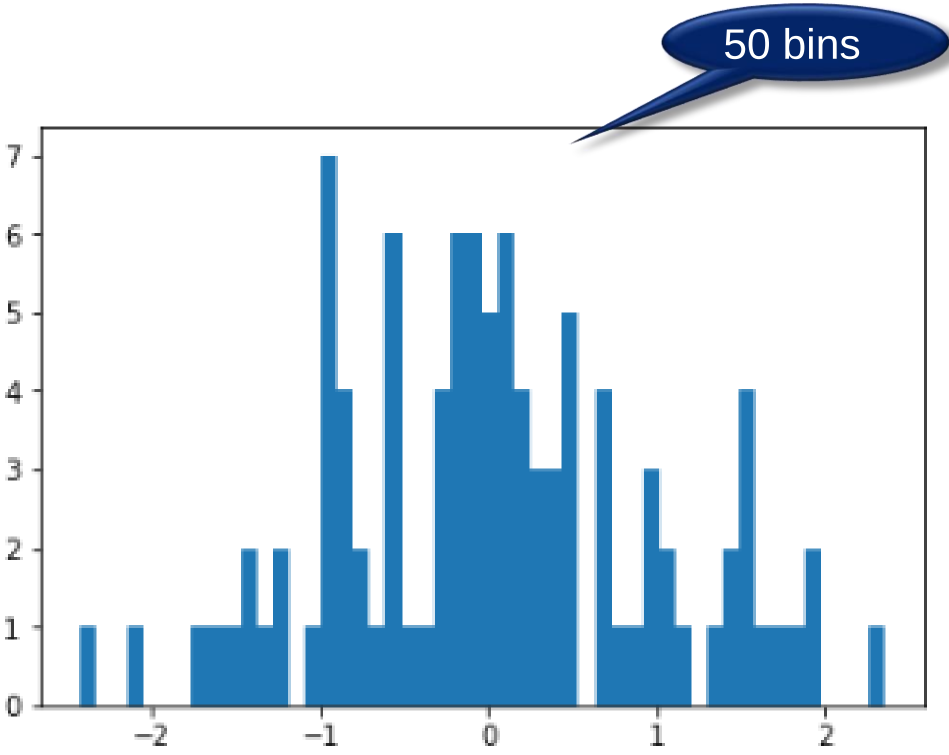
```
(array([ 3.,  1.,  6., 14.,  8., 13.,
 array([-2.45099972, -2.04513586, -1.
        -0.42168042, -0.01581656,  0.
         1.60763888]),
 <a list of 10 Patch objects>)
```

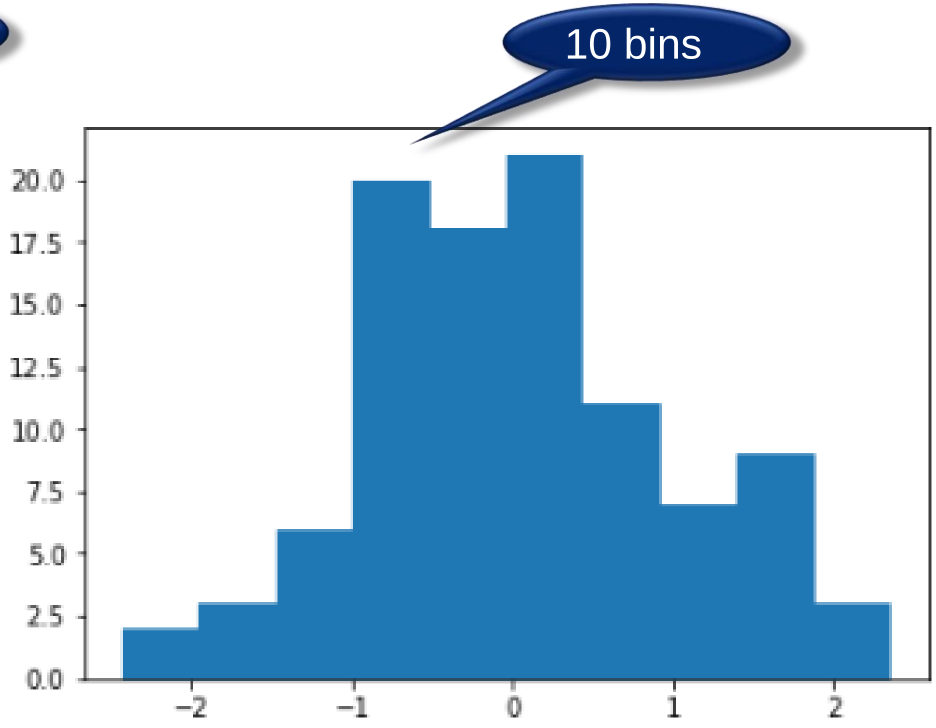


Histogram – more bins

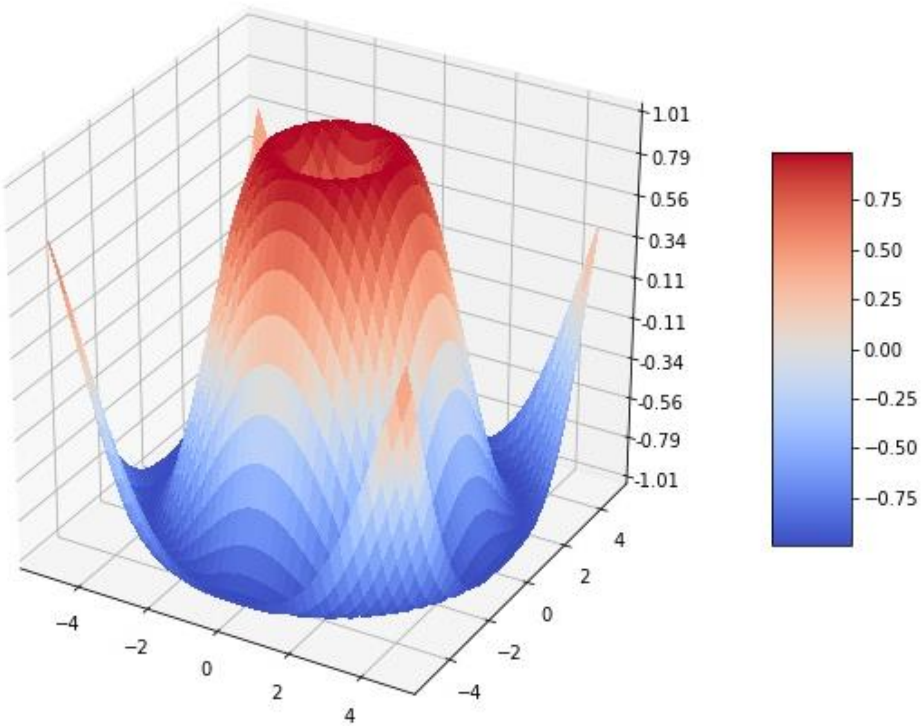
Same values – 50 bins

```
plt.hist(vals, bins=50)
```





3D Plots



Need to define :

- Imports
- Figure with 3d
- X and Y, Z
- Surface
- Optional: Colors for surface

3D Plots - Data

```
XX = np.arange(-5, 5, 0.25)
YY = np.arange(-5, 5, 0.25)
```

```
XX
```

```
array([-5.  , -4.75, -4.5 , -4.25, -4.  , -3.75, -3.5 , -3.25, -3.  ,
       -2.75, -2.5 , -2.25, -2.  , -1.75, -1.5 , -1.25, -1.  , -0.75,
       -0.5 , -0.25,  0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,
        1.75,  2.  ,  2.25,  2.5 ,  2.75,  3.  ,  3.25,  3.5 ,  3.75,
        4.  ,  4.25,  4.5 ,  4.75])
```

```
YY
```

```
array([-5.  , -4.75, -4.5 , -4.25, -4.  , -3.75, -3.5 , -3.25, -3.  ,
       -2.75, -2.5 , -2.25, -2.  , -1.75, -1.5 , -1.25, -1.  , -0.75,
       -0.5 , -0.25,  0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,
        1.75,  2.  ,  2.25,  2.5 ,  2.75,  3.  ,  3.25,  3.5 ,  3.75,
        4.  ,  4.25,  4.5 ,  4.75])
```

3D Plots - meshgrid

```
X, Y = np.meshgrid(XX, YY)
X
```

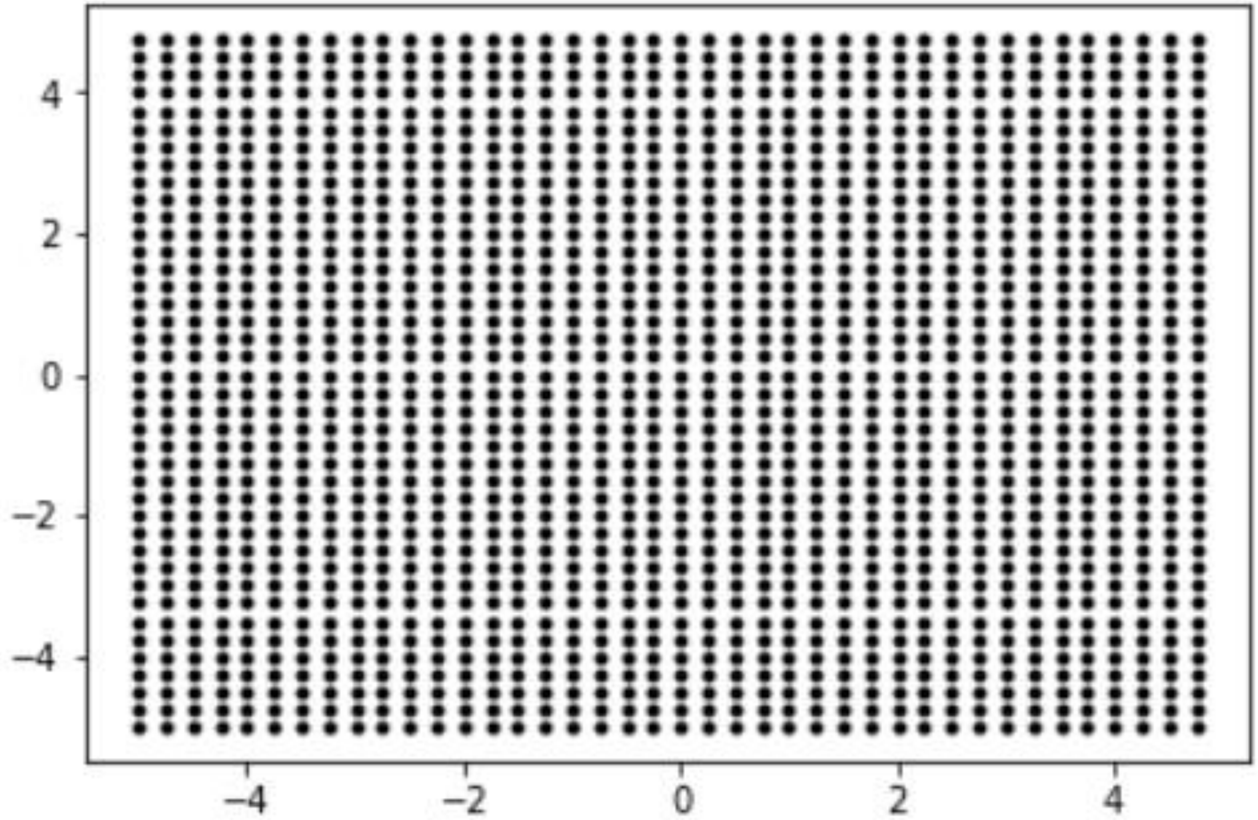
```
array([[-5.  , -4.75, -4.5 , ...,  4.25,  4.5 ,  4.75],
       [-5.  , -4.75, -4.5 , ...,  4.25,  4.5 ,  4.75],
       [-5.  , -4.75, -4.5 , ...,  4.25,  4.5 ,  4.75],
       ...,
       [-5.  , -4.75, -4.5 , ...,  4.25,  4.5 ,  4.75],
       [-5.  , -4.75, -4.5 , ...,  4.25,  4.5 ,  4.75],
       [-5.  , -4.75, -4.5 , ...,  4.25,  4.5 ,  4.75]])
```

```
Y
```

```
array([[-5.  , -5.  , -5.  , ..., -5.  , -5.  , -5.  ],
       [-4.75, -4.75, -4.75, ..., -4.75, -4.75, -4.75],
       [-4.5 , -4.5 , -4.5 , ..., -4.5 , -4.5 , -4.5 ],
       ...,
       [ 4.25,  4.25,  4.25, ...,  4.25,  4.25,  4.25],
       [ 4.5 ,  4.5 ,  4.5 , ...,  4.5 ,  4.5 ,  4.5 ],
       [ 4.75,  4.75,  4.75, ...,  4.75,  4.75,  4.75]])
```

3D Plots - meshgrid

```
plt.plot(X,Y, marker='.', color='k', linestyle='none')
```



3D Plots - imports

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np
fig = plt.figure(figsize=(10,8))
ax = fig.gca(projection='3d')
```

3D Plots - data

```python
# Make data.
XX = np.arange(-5, 5, 0.25)
YY = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(XX, YY)
R = np.sqrt(X**2 + Y**2)
Z=np.sin(R)
```

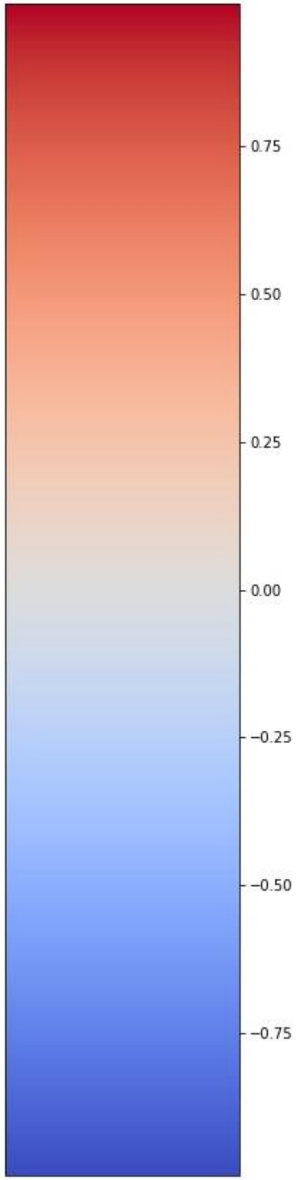
Z = sin (sqrt (x^2 + y^2))

3D Plots - surface

```
fig = plt.figure(figsize=(10,8))
ax = fig.gca(projection='3d')
# Plot the surface.
surf = ax.plot_surface(X, Y, Z,
                       cmap=cm.coolwarm,
                       linewidth=0,
                       antialiased=False)

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

3D Plots - colorbar

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



3D Plots – final plot

```
plt.show()
```