Plotting and More About Classes

Chapter 13

A slight course correction

- The most popular Python plotting package is matplotlib
- So we are going to focus on it
  - https://matplotlib.org/
  - https://matplotlib.org/faq/usage_faq.html

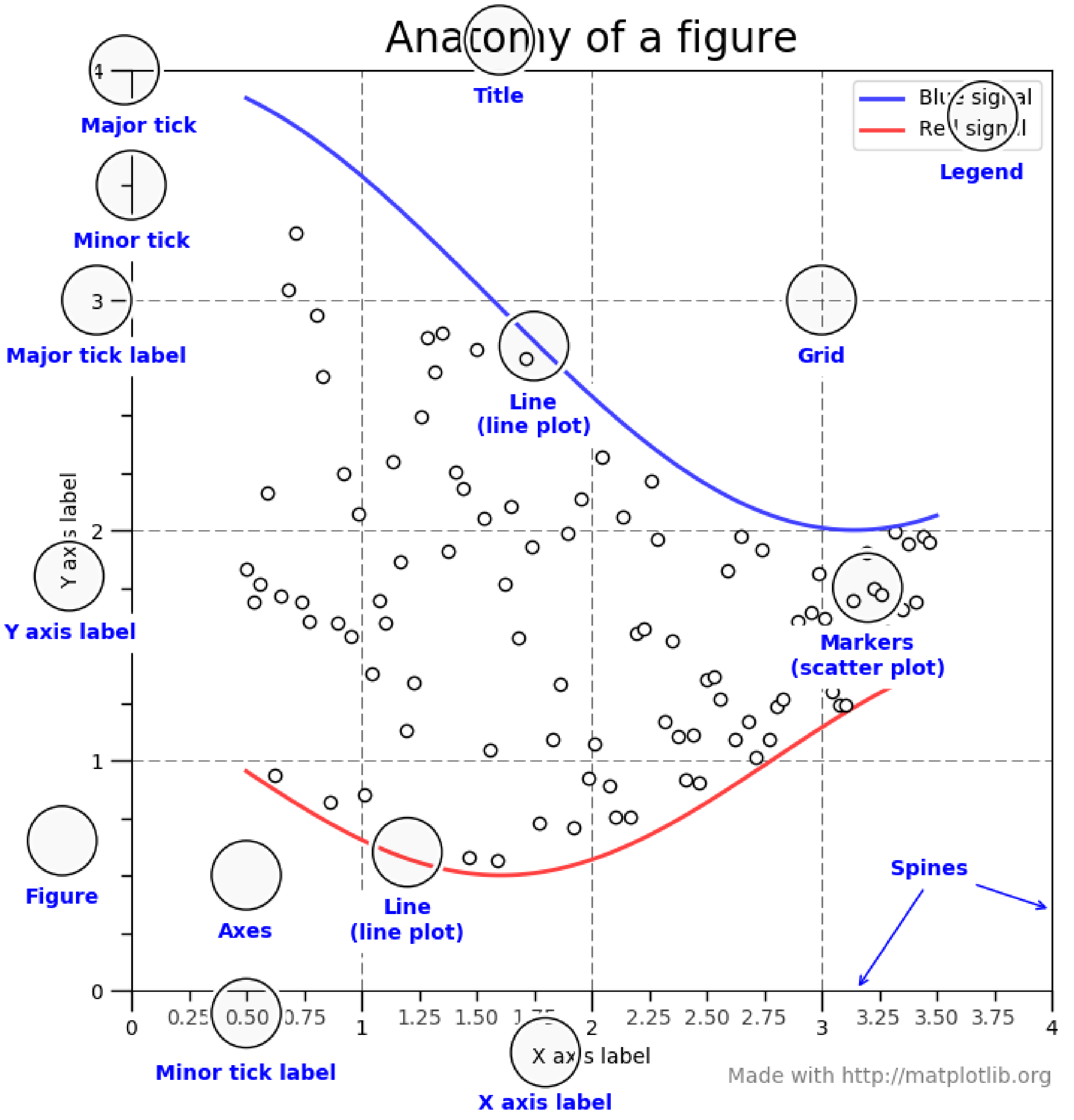
matplotlib
Version 3.0.2

home | examples | tutorials | API | docs »

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

Matplotlib

- Most popular Python plotting package
  - Began for only two dimensional plotting
  - Add-ons allow for a number of other types of plotting
    - mplot3D – 3 dimensional plotting
- Open Source
  - Available on github
  - Uses PSF (Python Software License)
  - Many third-party extensions available
- Range of complexity
  - General – plot this function
  - Specific – set this pixel
- `matplotlib.pyplot` tracks hierarchy as a **state machine**



Anatomy of a figure
Title
Major tick
Minor tick
Major tick label
Grid
Legend
Blue signal
Red signal
Line
(line plot)
Y axis label
Markers
(scatter plot)
Figure
Axes
Line
(line plot)
Spines
Minor tick label
X axis label
Made with http://matplotlib.org
0
1
2
3
4
0.25
0.50
0.75
1.25
1.50
1.75
2.25
2.50
2.75
3.25
3.50
3.75

What does a figure contain?

- **Canvas** mostly invisible to us as users
- A set of **artist**s
  - Figure Title
  - Figure Legend
- A set of **Axes**

- `fig = plt.figure()`

What, then are Axes?

- An **Axes** contains what we think of when we think of a plot or a graph
  - 2 (or 3) **axis** objects
  - Title
  - Legend
  - Limits on each axis
  - Labels for each axis
  - `fig, ax_lst = plt.subplots(2, 2)  # a figure with a 2x2 grid of Axes`

- An **Axis** looks like a number line
  - Positioned by a **Locator**
  - Ticks are controlled by a **Formatter**

Everybody is an Artist!

- Pretty much everything you see on a Figure is an Artist
  - Axes
  - Axis
  - Lines
  - Text

Choosing a backend

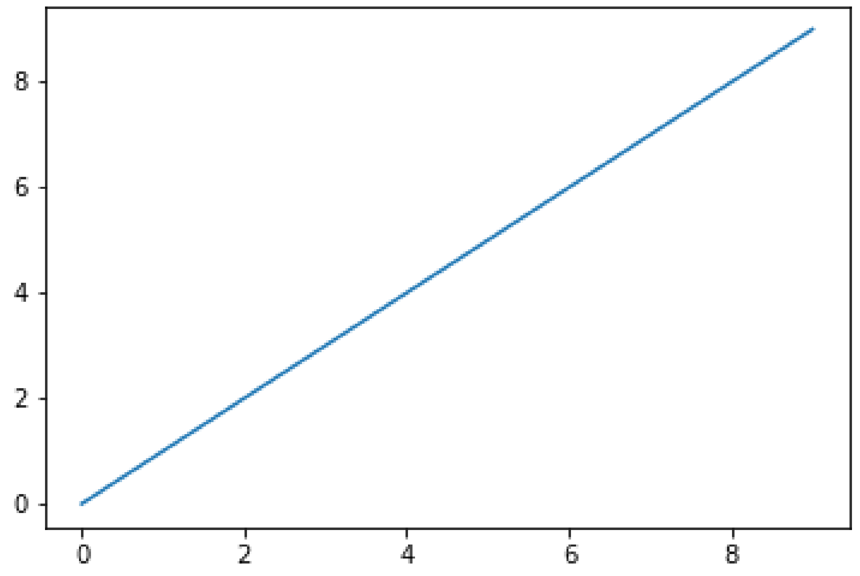
- The **backend** help pyplot render images
  - Agg – Anti-Grain Geometry raster graphics (png files)
  - PS – Post Script vector graphics (ps files)
  - PDF – Vector graphics (Portable Document File)
  - GTK3Agg – Common toolkit for interactive applications
  - macosx – Cocoa rendering for OSx

- Setting up a backend

```
import matplotlib
#matplotlib.use('Agg')
import matplotlib.pyplot as plt
```

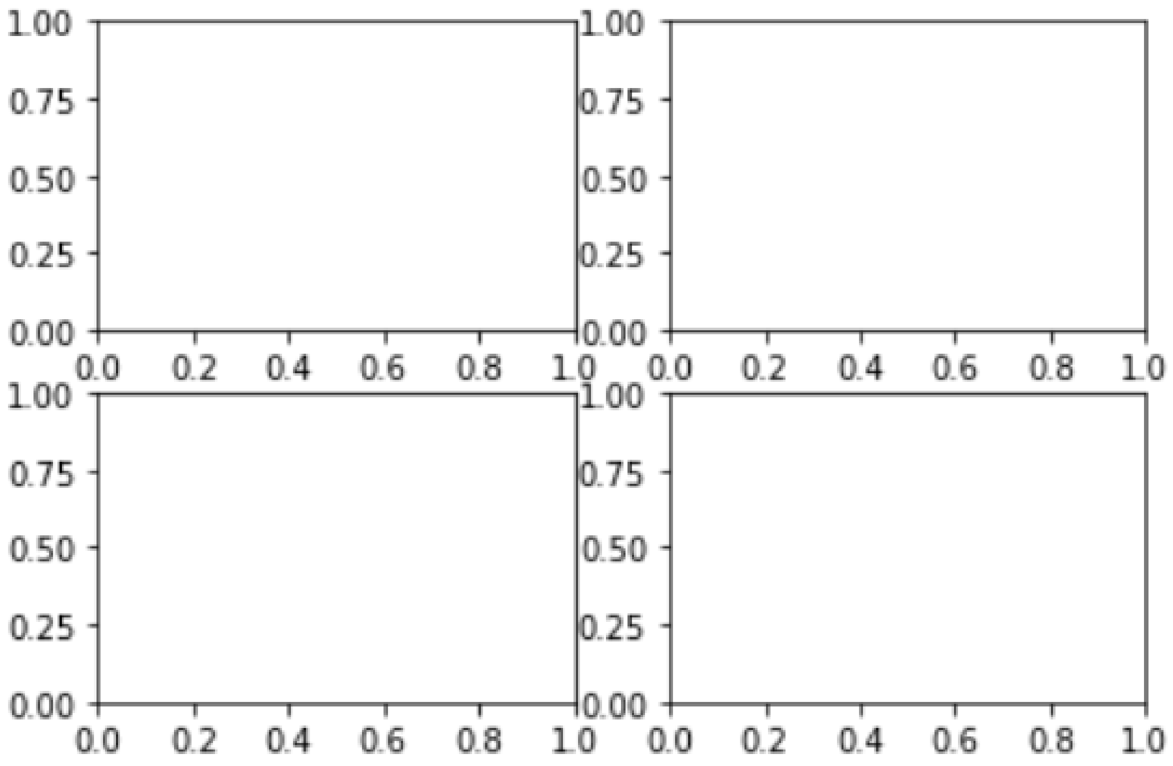
An Example

```
#%matplotlib inline
import matplotlib as mpl   # using alias for matplotlib
#mpl.use('Agg')
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)  # 1x1 grid, first subplot
# or ax = fig.add_subplot(1,1,1)
ax.plot(range(10))
fig.savefig('firstplot.png')
```



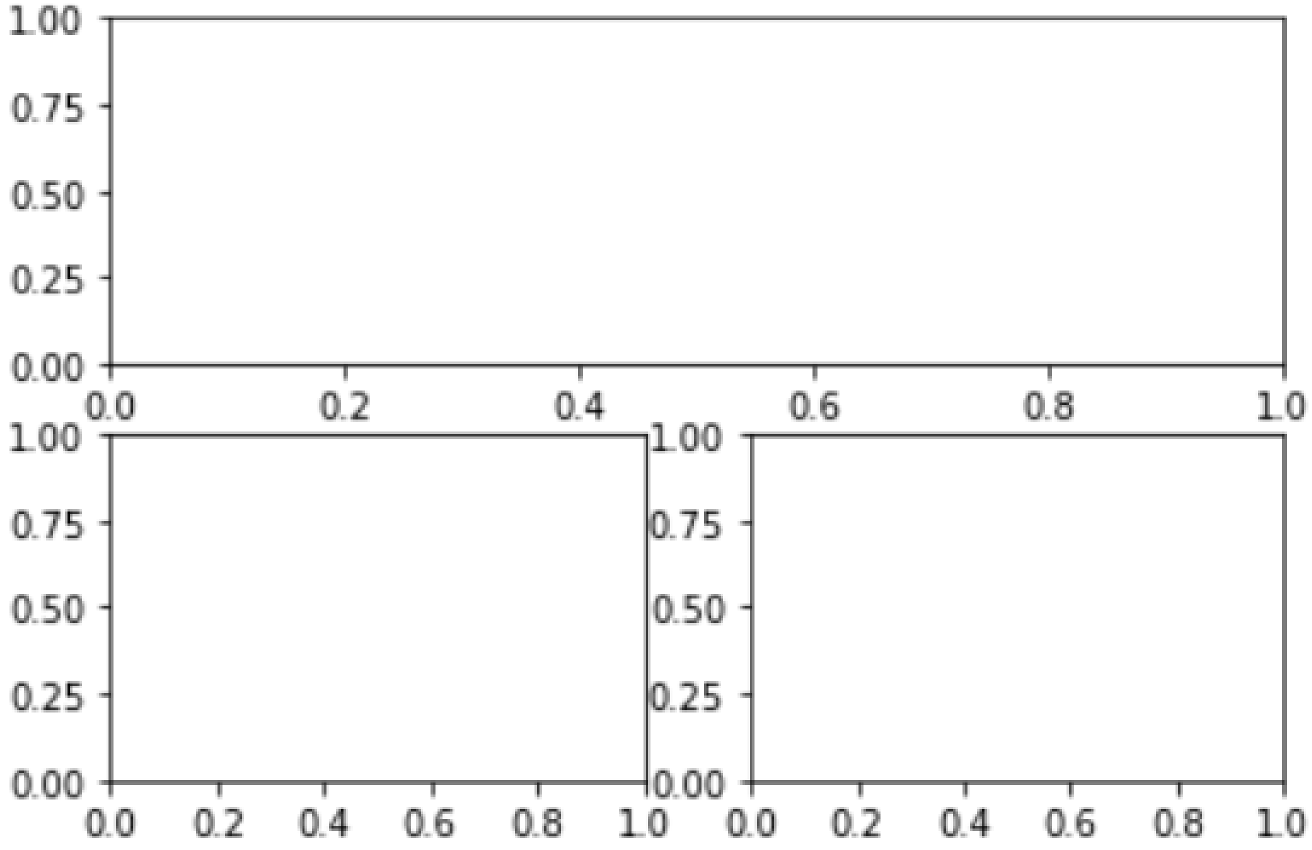
Another example

```
import matplotlib.pyplot as plt
plt.close()
fig1 = plt.figure()
fig1.add_subplot(221)   #top left
fig1.add_subplot(222)   #top right
fig1.add_subplot(223)   #bottom left
fig1.add_subplot(224)   #bottom right
plt.show()
```



One more

```
plt.close()
fig2 = plt.figure()
fig2.add_subplot(223)   #bottom left
fig2.add_subplot(224)   #bottom right
plt.subplot2grid((2,2),(0,0), colspan=2)
plt.show()
```



Let’s plot something interesting

```
plt.close()
principal = 10000
interestRate = 0.20
years = 20
values = []
for i in range(years+1):
    values.append(principal)
    principal += principal*interestRate
plt.plot(values)
plt.title('20% growth, compounded annually')
plt.xlabel('Years of compounding')
plt.ylabel('Value of principal ($)')
plt.show()
```

Colors and shapes

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

Markers

character	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker

Line Styles

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

Plotting Complexity

- The Python **math** package provides some basic math functions

And now **NumPy**

- NumPy is the fundamental package for scientific computing with Python. It contains among other things:
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform, and random number capabilities
- Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.
- NumPy is licensed under the BSD license, enabling reuse with few restrictions.
- http://numpy.org

Why use NumPy

- Lists and tuples are okay for storing and manipulating small data sets
    - Can’t (easily) do math on them
        - We can use list comprehension to act on each element of a list
        - But how do we multiply two lists?
    - Have to iterate and initialize to create a list of fixed size
    - Difficult to work with mutli-dimensional data
        - [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]

What NumPy can do for you!

- NumPy easily creates multi-dimensional arrays
- NumPy uses storage much more efficiently than lists
  - A 100x100x100 matrix of single precision floating point numbers
    - 20MB (approx.) as lists of lists
    - 4MB (approx.) as a NumPy ndarray
- NumPy accesses storage much quicker
- Functionality such as FFT, basic statistics, linear algebra can be accomplished with simple function calls
- Data is mutable – but must be the same type (restriction)

Vectors from lists

```python
# Creating ndarrays from a list
import numpy
a = numpy.array([1,3,5,7,9])
b = numpy.array([3,5,6,7,9])
c = a + b
print(c)
print(type(c))
print(c.shape)
```

```
[ 4  8 11 14 18]
<class 'numpy.ndarray'>
(5,)
```

Creating matrices from lists

```
# Creating a matrix from a list of lists
list = [[1, 2, 3], [3, 6, 9], [2, 4, 6]]
a = numpy.array(list)
print(a)
print(a.shape)
print(a.dtype)

# or directly as a matrix
M = numpy.array([[1, 2], [3, 4]])
print(M.shape)
```

```
[[1 2 3]
 [3 6 9]
 [2 4 6]]
(3, 3)
int32
(2, 2)
```

Accessing matrix elements

```
# Some oprations on matrices
print(a[0])                # Single row
print(a[1,2])              # Single element (row major)
print(a[1,1:3])            # Slide
a[:, 0] = [0,9,8]          # Replace a slice
print(a)
```

```
[1 2 3]
9
[6 9]
[[0 2 3]
 [9 6 9]
 [8 4 6]]
```

Some functions to create matrices

```
x = numpy.arange(0, 10, 1)       # arguments: start, stop, step
print(x)
y = numpy.linspace(0, 10, 10)   # arguments: first, last, number of values
print(y)
```

```
[0 1 2 3 4 5 6 7 8 9]
[  0.           1.11111111   2.22222222   3.33333333   4.44444444
   5.55555556   6.66666667   7.77777778   8.88888889  10.        ]
```

```
print(numpy.logspace(0, 10, 10, base=numpy.e))
print(numpy.logspace(0, 10, 10, base=2))
print(numpy.logspace(0, 10, 11, base=10))
```

```
[  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]
[  1.00000000e+00   2.16011948e+00   4.66611616e+00   1.00793684e+01
   2.17726400e+01   4.70315038e+01   1.01593667e+02   2.19454460e+02
   4.74047853e+02   1.02400000e+03]
[  1.00000000e+00   1.00000000e+01   1.00000000e+02   1.00000000e+03
   1.00000000e+04   1.00000000e+05   1.00000000e+06   1.00000000e+07
   1.00000000e+08   1.00000000e+09   1.00000000e+10]
```

Some common square matrices

```
diag = numpy.diag([1,2,3])
print(diag)
zeros = numpy.zeros(5)
print(zeros)
print(zeros.dtype)
zerosints = numpy.zeros(5, dtype=numpy.int)
print(zerosints)
print(zerosints.dtype)
ones = numpy.ones((3,3))
print(ones)
```

```
[[1 0 0]
 [0 2 0]
 [0 0 3]]
[ 0.  0.  0.  0.  0.]
float64
[0 0 0 0 0]
int32
[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
```

ndarray type safety

- Once created an ndarray element can't be assigned a new type

```
d = numpy.arange(5) # just like range()
print(d)        # [0 1 2 3 4]
d[1]= 9.7
print(d)        # [0 9 2 3 4]
```

- But operations can change type

```
print(d*0.4)   # [ 0.   3.6  0.8  1.2  1.6]
```

Type driven explicitly or by ‘step’

```
d = numpy.arange(5, dtype=numpy.float)
print(d)
d = numpy.arange(3, 7, 0.5)
print(d)
```

```
[ 0.  1.  2.  3.  4.]
[ 3.   3.5  4.   4.5  5.   5.5  6.   6.5]
```

NumPy Files

- Reading files

```
votes = numpy.genfromtxt('votes.csv', delimiter=',',skip_header=1)
print(votes.shape)
print(votes[0:5])
votes = numpy.genfromtxt('votes.csv', delimiter=',',skip_header=1,usecols=(1,2,
print(votes.shape)
print(votes[0:5])
```

- Writing

```
numpy.savetxt('votes2.pipe', votes, delimiter='|',newline='\r\n'
```

Random numbers

```
# Random Numbers
randomNums = numpy.random.rand(3,3) # of 3x3 matrix shape
print(randomNums)

# to generate a random integer within two integers
randomInts = numpy.random.randint(1,100)
print(randomInts)
```

More random number functions

rand(d0, d1, …, dn)	Random values in a given shape.
randn(d0, d1, …, dn)	Return a sample (or samples) from the "standard normal" distribution.
randint(low[, high, size, dtype])	Return random integers from *low* (inclusive) to *high* (exclusive).
random_integers(low[, high, size])	Random integers of type np.int between *low* and *high*, inclusive.
random_sample([size])	Return random floats in the half-open interval [0.0, 1.0).
random([size])	Return random floats in the half-open interval [0.0, 1.0).
ranf([size])	Return random floats in the half-open interval [0.0, 1.0).
sample([size])	Return random floats in the half-open interval [0.0, 1.0).
choice(a[, size, replace, p])	Generates a random sample from a given 1-D array
bytes(length)	Return random bytes.

NumPy basic statistics

- NumPy provides a variety of stats
  - Order
  - Averages and Variances
  - Correlating
  - Histograms

```
a = numpy.array([1, 4, 3, 8, 9, 2, 3], float)
print(numpy.median(a))
```

```
3.0
```

```
import numpy as np

a = np.array([[1, 2, 1, 3], [5, 3, 1, 8]], dtype=np.float)
c = np.corrcoef(a)
print(c)

# .T returns the transposed version of the array
x = np.array([[0, 2], [1, 1], [2, 0]]).T
print(x)
print(np.cov(x))
```

```
[[ 1.          0.72870505]
 [ 0.72870505  1.        ]]
[[0 1 2]
 [2 1 0]]
[[ 1. -1.]
 [-1.  1.]]
```

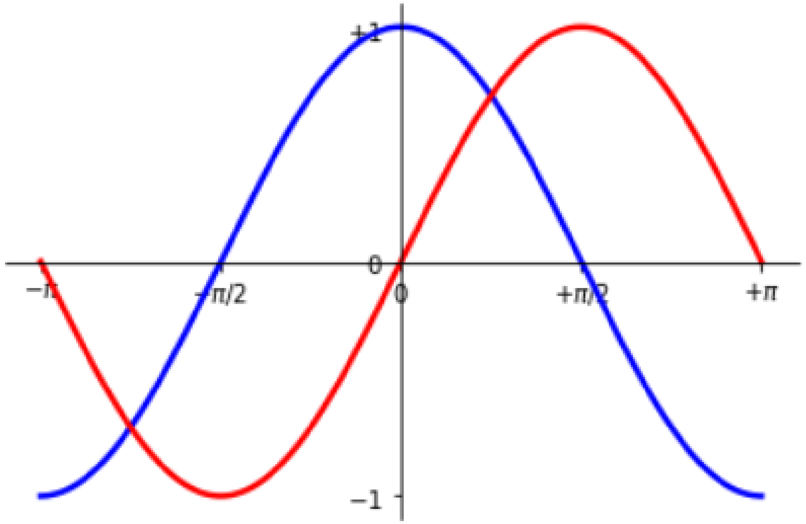
- https://docs.scipy.org/doc/numpy/reference/routines.statistics.html

Putting it together

Matpollib & NumPy

A pretty 'regular' plot

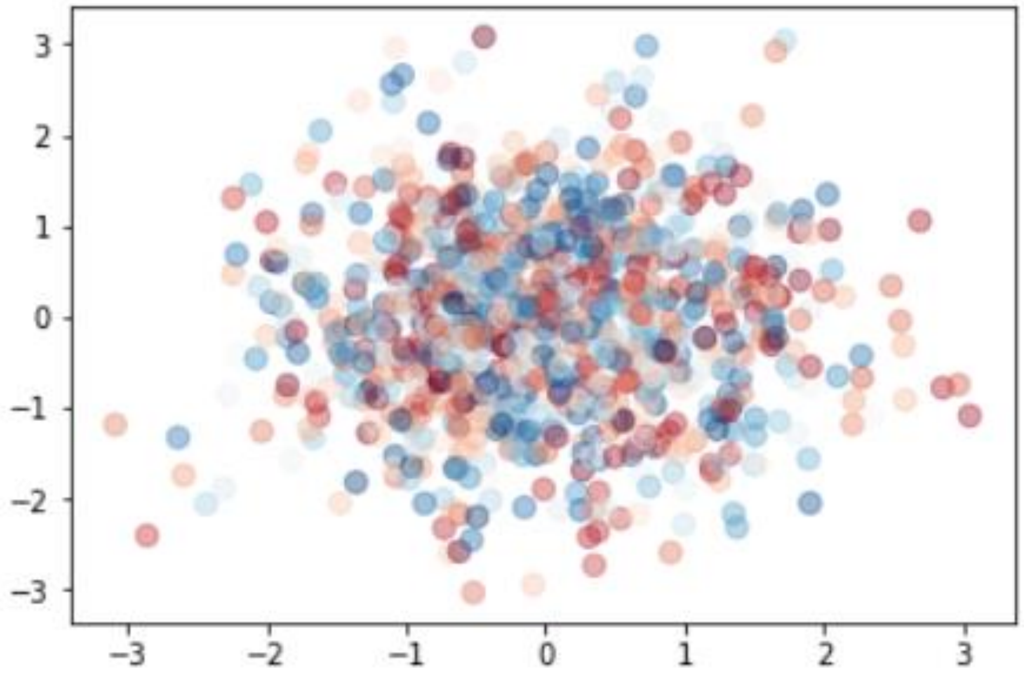
```
import matplotlib.pyplot as plt
import numpy as np

X = np.linspace(-np.pi, np.pi, 256,endpoint=True)
C = np.cos(X)
S = np.sin(X)

plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red", linewidth=2.5, linestyle="-")

ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))

plt.xlim(X.min() * 1.1, X.max() * 1.1)
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
          [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])

plt.ylim(C.min() * 1.1, C.max() * 1.1)
plt.yticks([-1, 0, +1],
          [r'$-1$', r'$0$', r'$+1$'])

plt.show()
```



Scatter plot

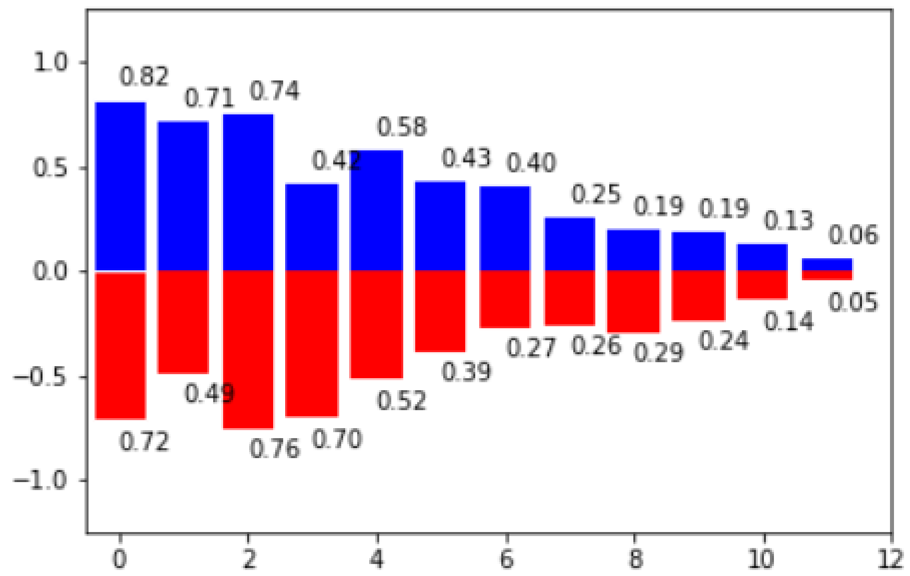
- Scatter
  - At least X and Y

```
plt.close()
n = 1024
X = np.random.normal(0,1,n)
Y = np.random.normal(0,1,n)
C = np.random.randint(0, 256**3, n)
cmap = plt.get_cmap("RdBu")

plt.scatter(X, Y, s=60, alpha=0.4, c=C, cmap=cmap)
plt.show()
```



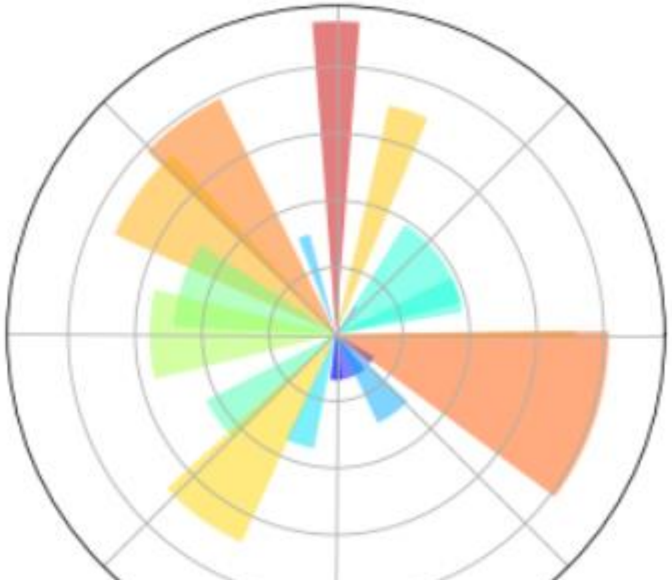
Bar chart

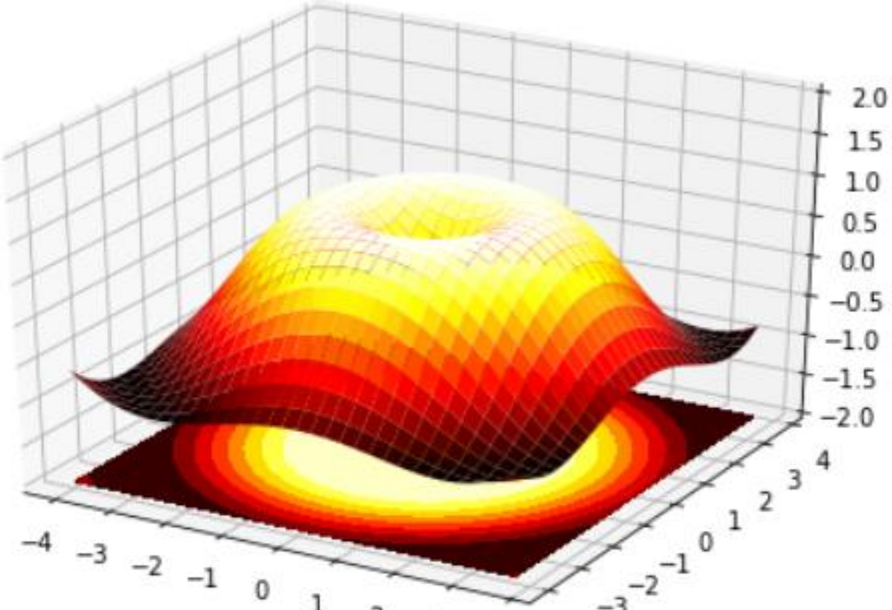
```
n = 12
X = np.arange(n)
Y1 = (1 - X / float(n)) * np.random.uniform(0.5, 1.0, n)
Y2 = (1 - X / float(n)) * np.random.uniform(0.5, 1.0, n)

plt.bar(X, +Y1, facecolor='blue', edgecolor='white')
plt.bar(X, -Y2, facecolor='red', edgecolor='white')

for x, y in zip(X, Y1):
    plt.text(x + 0.4, y + 0.05, '%.2f' % y, ha='center', va= 'bottom')

for x, y in zip(X, Y2):
    plt.text(x + 0.4, -y - 0.05, '%.2f' % y, ha='center', va= 'top')

plt.xlim(-.5, n)
plt.ylim(-1.25, 1.25)

plt.show()
```



Other graphs

- Pie Chart
- Polar Chart
- 3-D
- Animations





Plotting Mortgages

- Please be sure to read and understand this.
- Work through the code using matplotlib.pyplot and numpy

Interactive Plotting

- Let’s go to the code