Matplotlib

March 2, 2020

Week 9 of CS 41

Today in CS41

- Pyplot
  - Basic Plotting
  - Customizations
  - Imperative and Object-Oriented Plotting
  - Histograms
  - Scatter Plots

Pyplot (`matplotlib.pyplot`)

MATLAB-style plotting in Python

Installation

You should have Matplotlib installed already, if you followed our installation instructions at the beginning of the quarter.

If not:

```
pip install matplotlib
```

What am I getting?

- Python 2D plotting library (with a MATLAB-esque plotting interface)
- Integrates with the rest of Python (for example, images in Flask)

A Sample Plot

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4])

plt.ylabel('some numbers')

plt.show()
```

Import the pyplot module

Add a plot to the display (which we don't see yet)

Modify a display parameter

Run the GUI main loop and draw all of the figures

Let's try it!

Watch Out: `plt.show()`

`plt.show()`

- Tells Matplotlib to draw all of the figure windows created so far and start the mainloop.
- The mainloop is blocking by default (script execution is paused until the user closes all of the GUI windows).
  - If you want to generate images without having a window appear, you can use `plt.savefig`.
- As a result, `show()` should only be called once per script and it isn't threadsafe.

Interactive Mode

`plt.ion()` starts interactive mode.

- Pyplot functions automatically draw to the screen.

`plt.ioff()` turns it off.

- Use non-interactive mode in scripts in which you want to generate one or more figures and then display them.

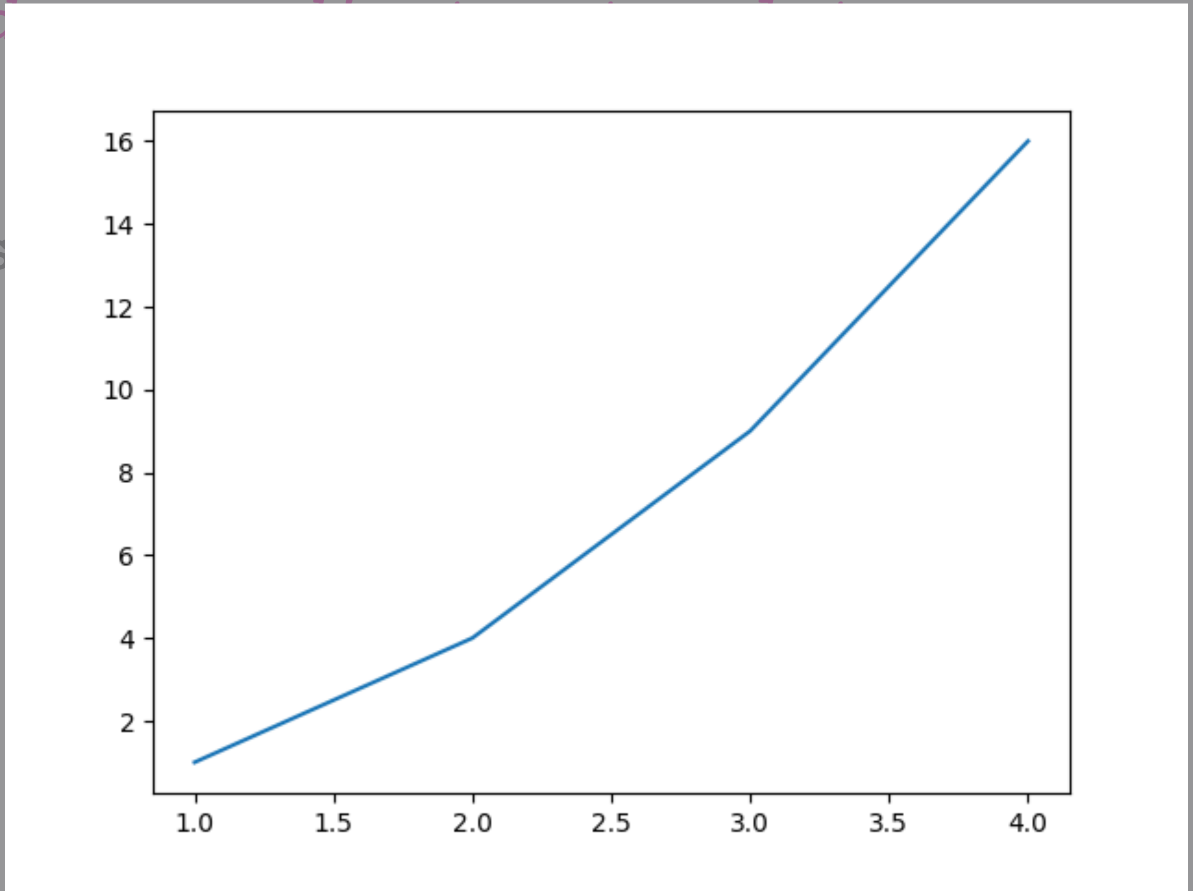
Let's see an example!

Customizations

```python
# Specify x and y coordinates to plot
xs = [1, 2, 3, 4]
ys = [1, 4, 9, 16]
plt.plot(xs, ys)
```

Customizations

```
# Specify x and
xs = [1, 2, 3,
ys = [1, 4, 9,
plt.plot(xs, ys
```



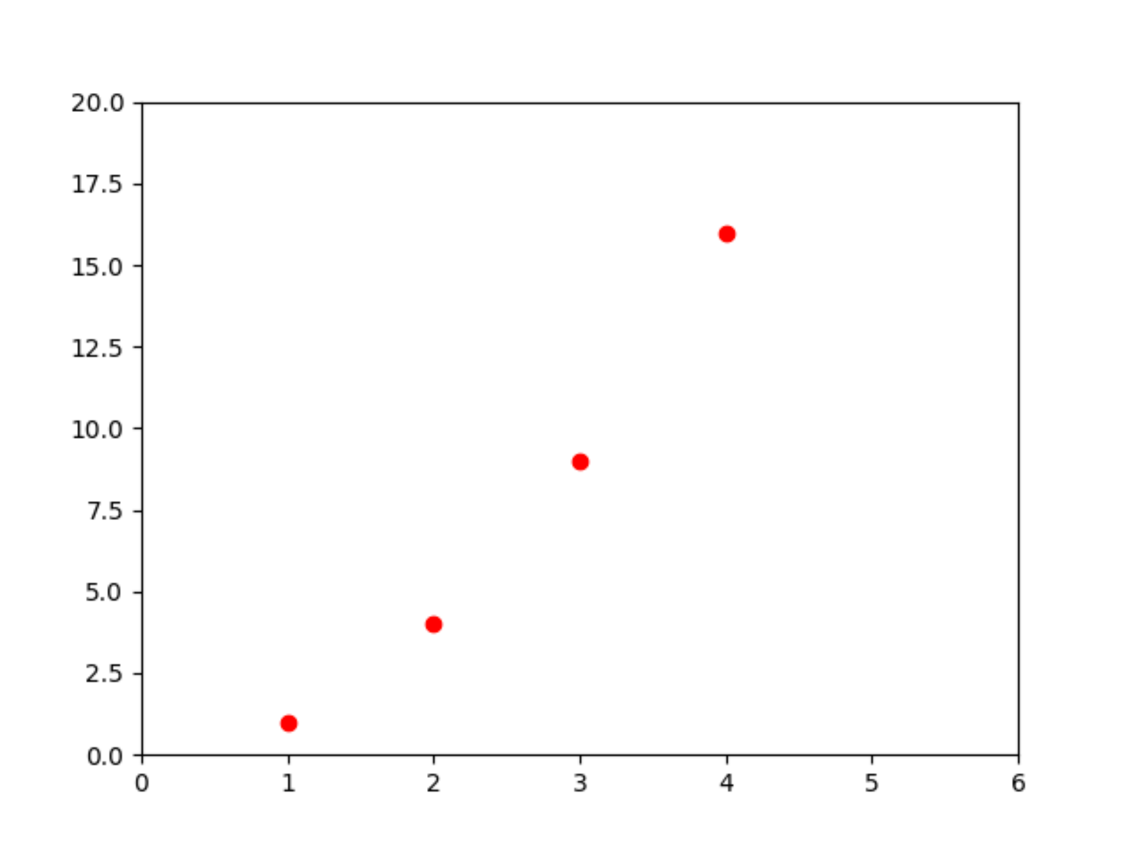
Customizations

```python
# Specify x and y coordinates to plot
xs = [1, 2, 3, 4]
ys = [1, 4, 9, 16]
plt.plot(xs, ys)

# Change the style of the plot
plt.plot(xs, ys, 'ro')
```

See the documentation of `plot` for more styles

Customizations

```
# Specify x and y coordinates to plot
xs = [1, 2, 3, 4
ys = [1, 4, 9, 1
plt.plot(xs, ys)

# Change the sty
plt.plot(xs, ys,
```



See the documentation of `plot` for more styles

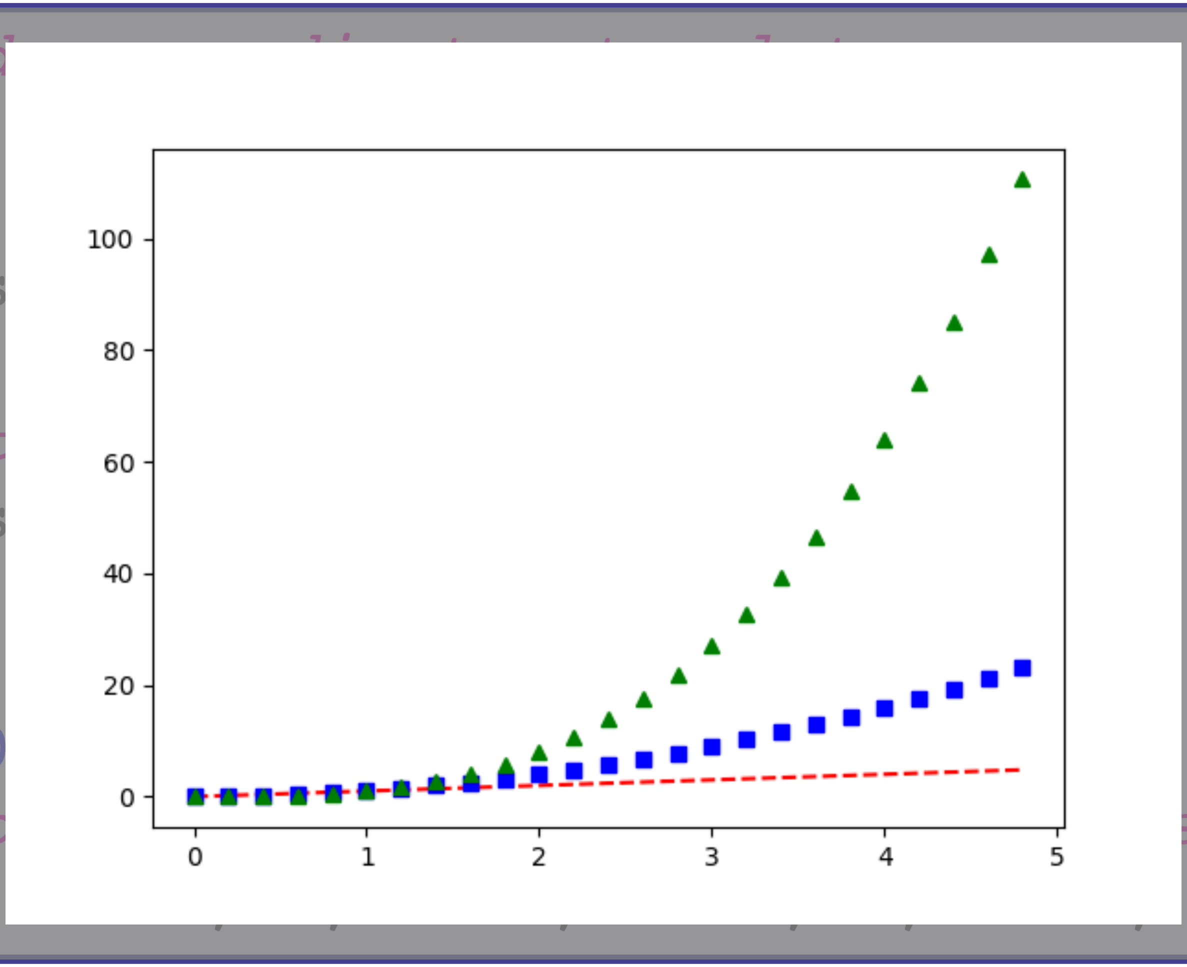
Customizations

```
# Specify x and y coordinates to plot
xs = [1, 2, 3, 4]
ys = [1, 4, 9, 16]
plt.plot(xs, ys)

# Change the style of the plot
plt.plot(xs, ys, 'ro')

# Use numpy!
t = np.arange(0., 5., 0.2)
# Red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
```

See the documentation of `plot` for more styles

Customizations

```
# Specify x and
xs = [1, 2, 3,
ys = [1, 4, 9,
plt.plot(xs, ys

# Change the st
plt.plot(xs, ys

# Use numpy!
t = np.arange(0
# Red dashes, b                                  s
plt.plot(t, t,                                  'g^')
```



See the documentation of `plot` for more styles

Multiple Plots and Figures

Use the `plt.subplot` function to create multiple sub-plots in one figure.

`plt.subplot(nrows, ncols, index, **kwargs)`

- `nrows` — The number of rows in the figure.
- `ncols` — The number of columns in the figure.
- `index` — The index of the subplot you want to work on
  - Between 1 and `nrows * ncols`.

```
plt.subplot(2, 3, 5)
```

Multiple Plots and Figures

Use the `plt.subplot` function to create multiple sub-plots in one figure.

`plt.subplot(pos, **kwargs)`

- `pos` — A three-digit integer where the first digit is the number of rows, the second the number of columns, and the third the index of the subplot.

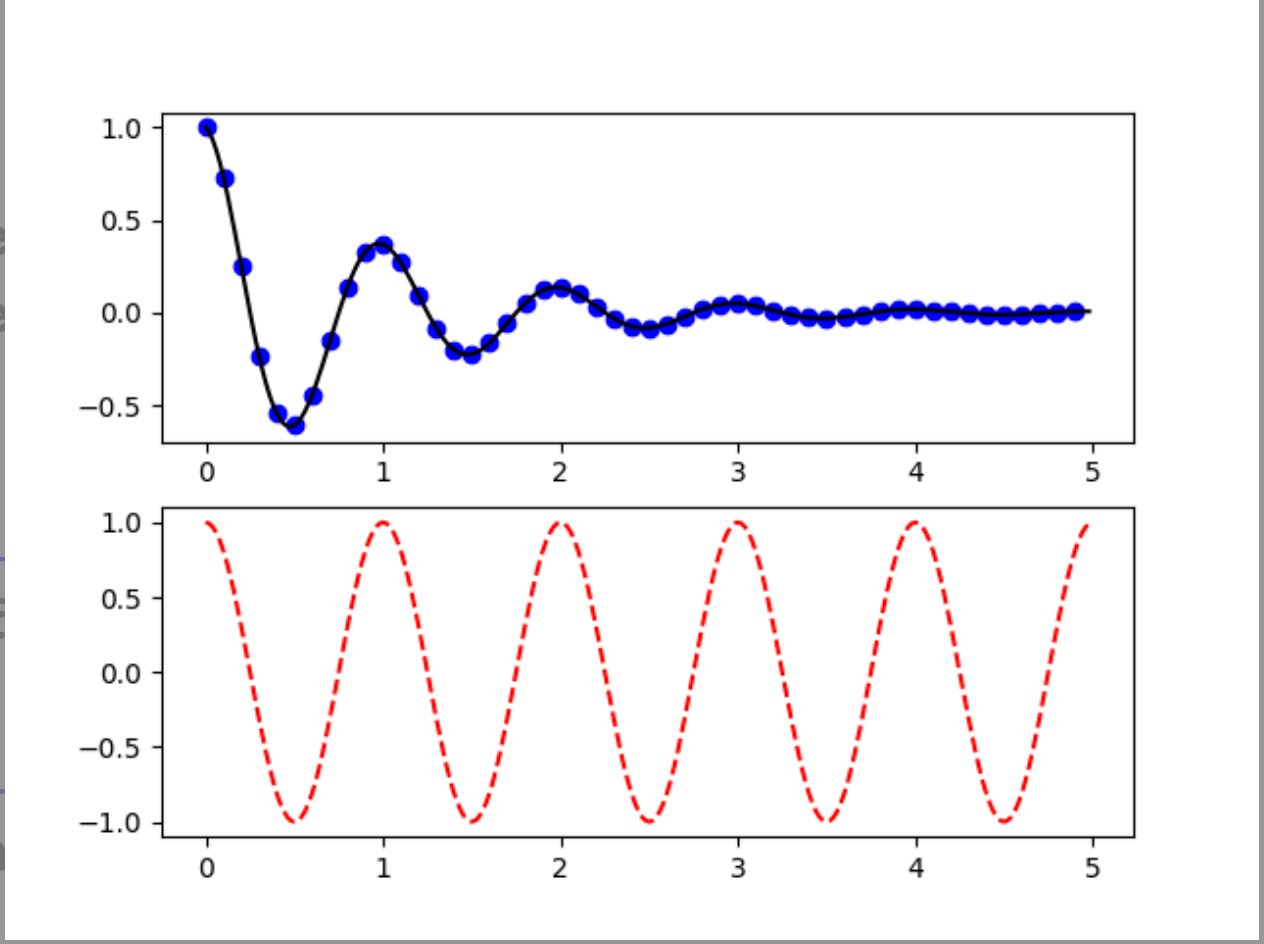
```
plt.subplot(235)
```

Multiple Plots and Figures

```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

Multiple Plots and Figures

```
def f(t):
    return np.

t1 = np.arange
t2 = np.arange

plt.figure()
plt.subplot(21
plt.plot(t1, f

plt.subplot(21
plt.plot(t2, n
plt.show()
```



Multiple Plots and Figures

```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

What's going on here?

Imperative and Object-Oriented Plotting

Matplotlib is imperative (what we've seen so far) and object-oriented!

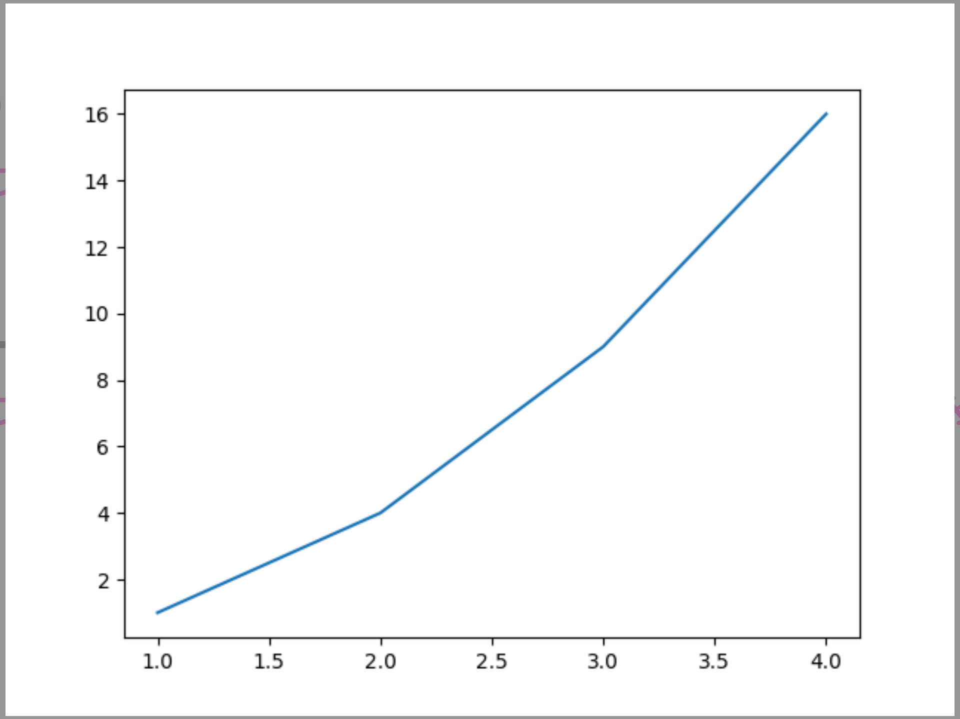
- Most of the configuration functions (`plt.subplot`, `plt.figure`) return *objects* whose methods can be used for plotting.
- A *figure* object (the entire plot window) can contain one or more *axes* objects.
    - Customize title, size, etc. of the figure.
    - Plot on the axes.
- Get the current axes (figure) with `plt.gca()` (`plt.gcf()`)

Multiple Plots and Figures

```
fig = plt.figure()
type(fig) # => matplotlib.figure.Figure

ax = fig.add_subplot(1, 1, 1)
type(ax)  # => matplotlib.axes._subplots.AxesSubplot

ax.plot([1, 2, 3, 4])
plt.show()
```

Multiple Plots and Figures

```
fig = plt.figure()
type(fig) # => mat

ax = fig.add_subpl
type(ax)  # => mat                          esSubplot

ax.plot([1, 4, 9,
plt.show()
```



Other Plots

Matplotlib has **so many** different plots: there's most likely one that'll suit your needs!

A brief tour:

- Histograms (`plt.hist`)
- Scatter plots (`plt.scatter`)
- Bar plots (`plt.bar`)
- Pie charts (`plt.pie`)
- 3D plotting (`mplot3d` toolkit)

We won't talk about these (they're very well documented, so you should read the docs to learn)!

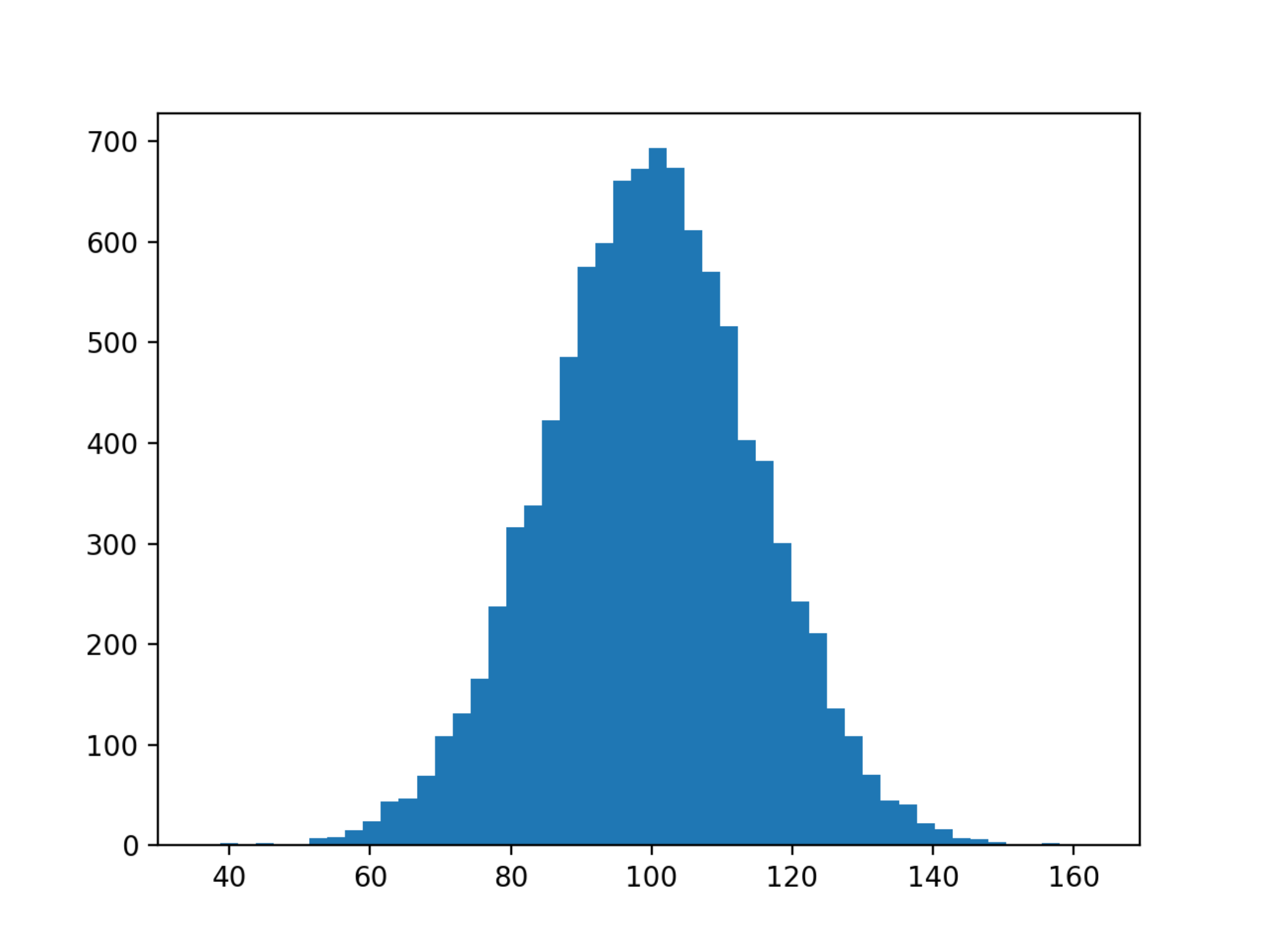
Alternatively, come see us if you're interested!

Histograms

```
# Fixing random state for reproducibility
np.random.seed(20081203)

mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

plt.hist(x, 50) # data, number of bins
plt.show()
```

Histograms

```
# Fixing rando
np.random.seed

mu, sigma = 10
x = mu + sigma

plt.hist(x, 50
plt.show()
```

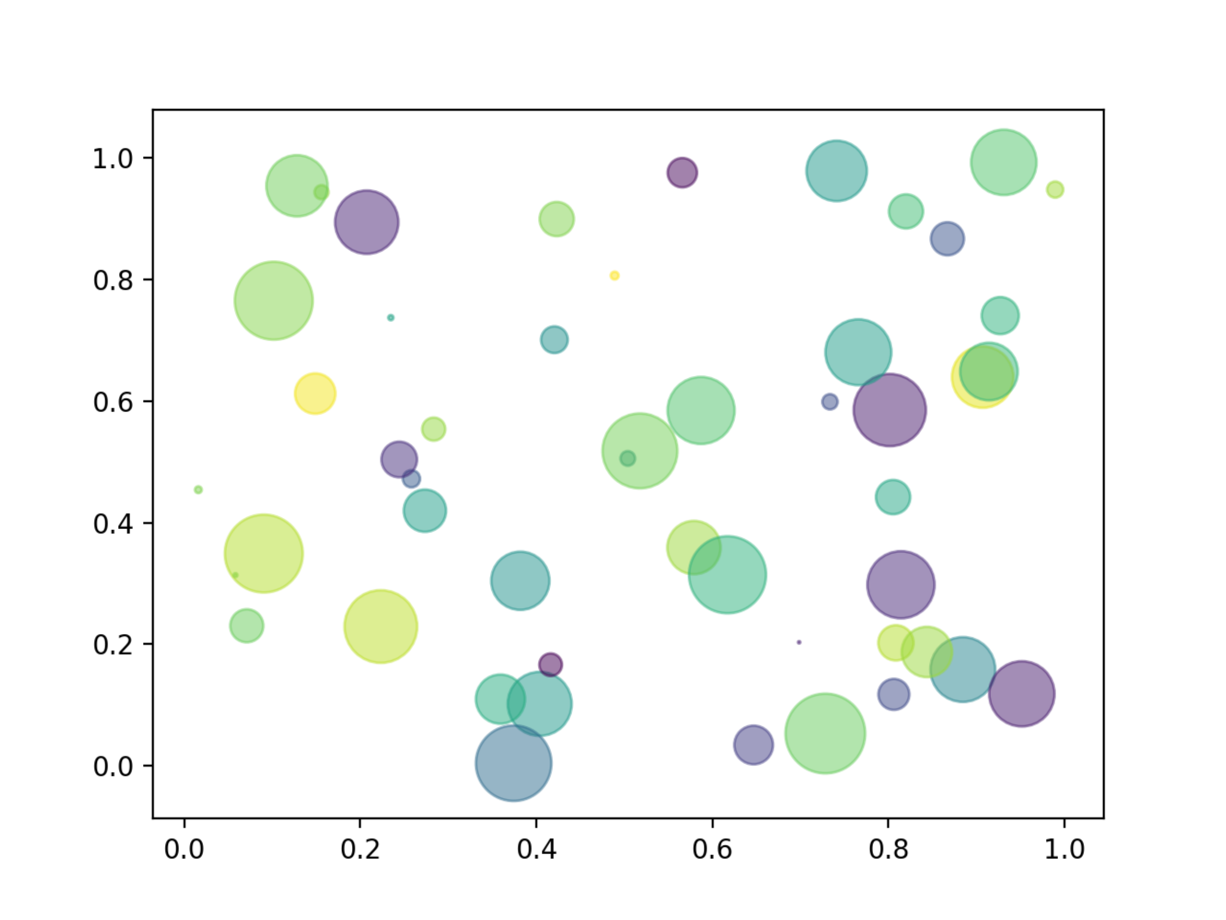


Scatter Plots

```
np.random.seed(20081203)

N = 50
x = np.random.rand(N)               # x-coordinates
y = np.random.rand(N)               # y-coordinates
colors = np.random.rand(N)          # color spectra values
area = (30 * np.random.rand(N))**2  # radii for points
alpha = 0.5                         # transparency

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

Scatter Plots

```
np.random.seed

N = 50
x = np.random.                       nates
y = np.random.                       nates
colors = np.ra                       ectra values
area = (30 * n                       r points
alpha = 0.5                          ency

plt.scatter(x,                       a)
plt.show()
```



1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0