

VISUALIZATION OF DATA

VISUALIZING RESULTS

- earlier saw examples of different orders of growth of procedures
- used graphs to provide an intuitive sense of differences
- example of leveraging an existing library, rather than writing procedures from scratch
- Python provides libraries for (among other topics):
 - graphing
 - numerical computation
 - stochastic computation
- want to explore idea of using existing library procedures to guide processing and exploration of data

USING PYLAB

- can import library into computing environment

```
import pylab as plt
```

- allows me to reference any library procedure as `plt.<procName>`

- provides access to existing set of graphing/plotting procedures
- here will just show some simple examples; lots of additional information available in documentation associated with `pylab`
- will see many other examples and details of these ideas if you opt to take 6.00.2x

SIMPLE EXAMPLE

- basic function plots two lists as x and y values
 - other data structures more powerful, use lists to demonstrate
- first, let's generate some example data

```
mySamples = []
myLinear = []
myQuadratic = []
myCubic = []
myExponential = []

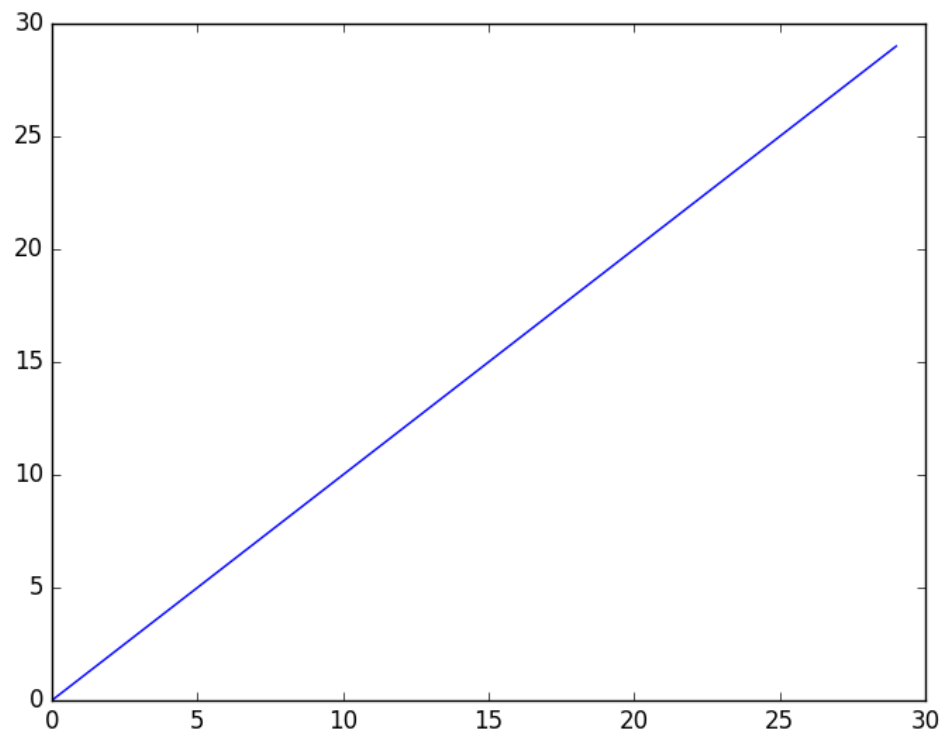
for i in range(0, 30):
    mySamples.append(i)
    myLinear.append(i)
    myQuadratic.append(i**2)
    myCubic.append(i**3)
    myExponential.append(1.5**i)
```

*selected 1.5 to keep displays
visible, more likely value for order
of growth example would be 2*

SIMPLE EXAMPLE

- to generate a plot, call `plt.plot(mySamples, myLinear)`
 - x values* (pointing to `mySamples`)
 - y values* (pointing to `myLinear`)
- arguments are lists of values (for now)
 - lists must be of the same length
- calling function in an **iPython** console will generate plots within that console
- calling function in a **Python** console will create a separate window in which plot is displayed

EXAMPLE DISPLAY



```
plt.plot(mySamples, myLinear)
```

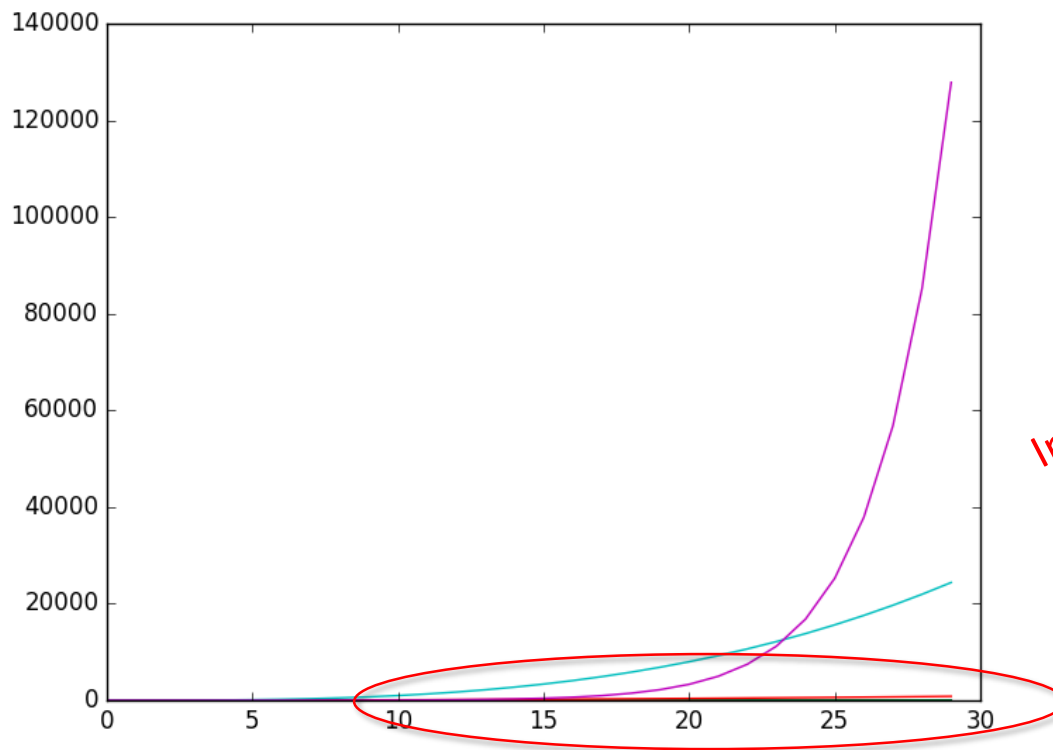

OVERLAPPING DISPLAYS

- suppose we want to display all of the graphs of the different orders of growth

- we could just call:

```
plt.plot(mySamples, myLinear)
plt.plot(mySamples, myQuadratic)
plt.plot(mySamples, myCubic)
plt.plot(mySamples, myExponential)
```


EXAMPLE OVERLAY DISPLAY



*Impossible to see linear
graph, or even
quadratic graph*

```
plt.plot(mySamples, myLinear)  
plt.plot(mySamples, myQuadratic)
```

```
plt.plot(mySamples, myCubic)  
plt.plot(mySamples, myExponential)
```

OVERLAPPING DISPLAYS

- not very helpful, can't really see anything but the biggest of the plots because the scales are so different
- can we graph each one separately?
- call
 - `plt.figure(<arg>)`
 - creates a new display with that name if one does not already exist
 - if a display with that name exists, reopens it for processing

gives a name to this figure; allows us to reference for future use

EXAMPLE CODE

```
plt.figure('lin')
```

```
plt.plot(mySamples, myLinear)
```

```
plt.figure('quad')
```

```
plt.plot(mySamples, myQuadratic)
```

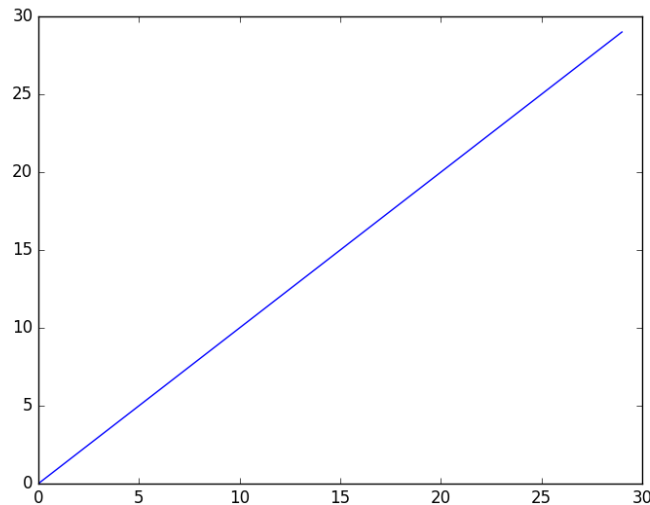
```
plt.figure('cube')
```

```
plt.plot(mySamples, myCubic)
```

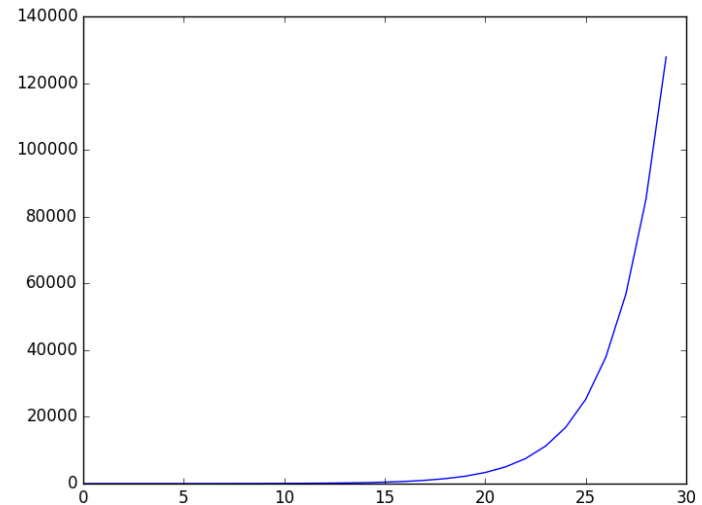
```
plt.figure('expo')
```

```
plt.plot(mySamples, myExponential)
```

SEPARATE PLOTS



```
plt.figure('lin')  
plt.plot(mySamples, myLinear)
```



```
plt.figure('expo')  
plt.plot(mySamples,  
myExponential)
```


PROVIDING LABELS

- Should really label the axes

```
plt.figure('lin')
```

```
plt.xlabel('sample points')  
plt.ylabel('linear function')
```

```
plt.plot(mySamples, myLinear)
```

```
plt.figure('quad')
```

```
plt.plot(mySamples, myQuadratic)
```

```
plt.figure('cube')
```

```
plt.plot(mySamples, myCubic)
```

```
plt.figure('expo')
```

```
plt.plot(mySamples, myExponential)
```

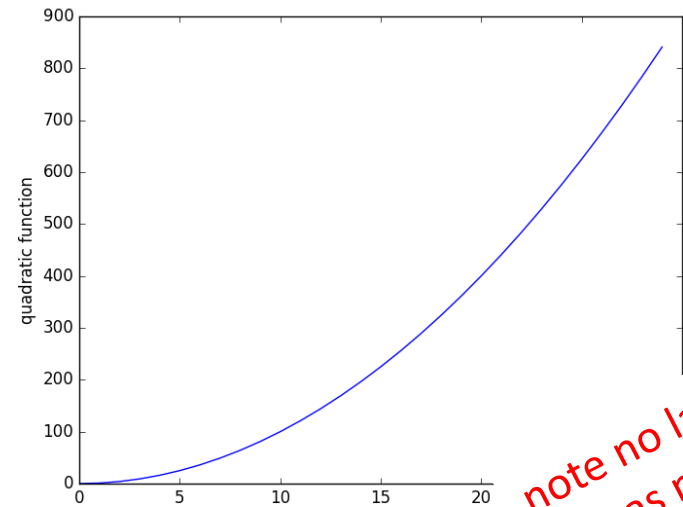
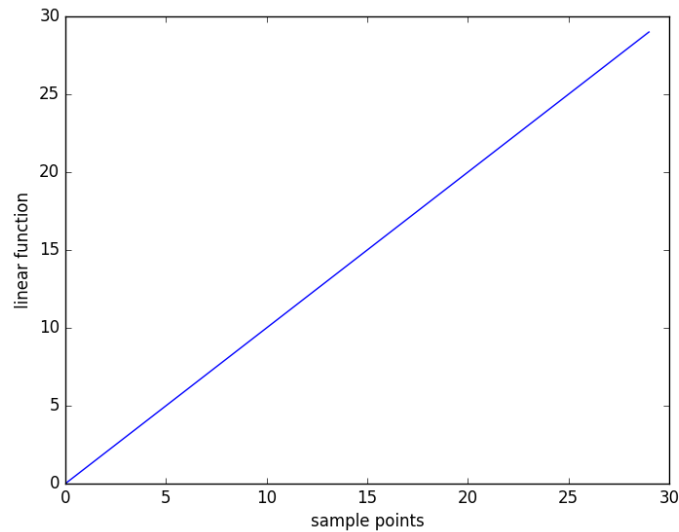
```
plt.figure('quad')
```

```
plt.ylabel('quadratic function')
```

functions to label axes

*note you must make figure
active before invoking labeling*

LABELED AXES



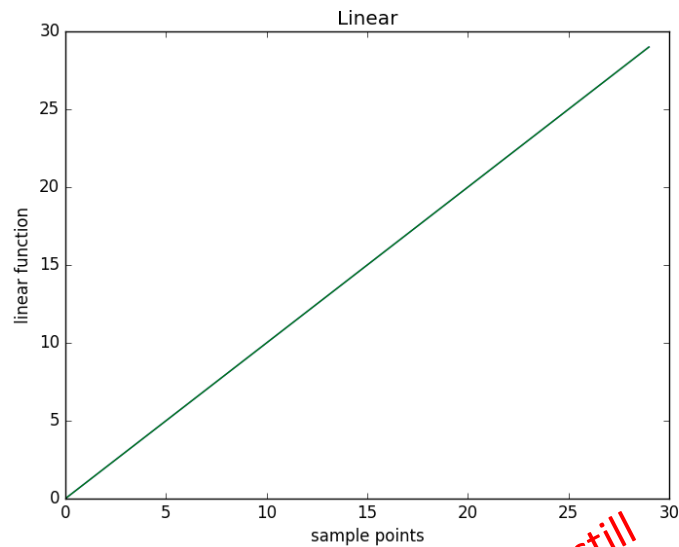
*note no label on x
axis as no invocation
while that display
was active*

ADDING TITLES

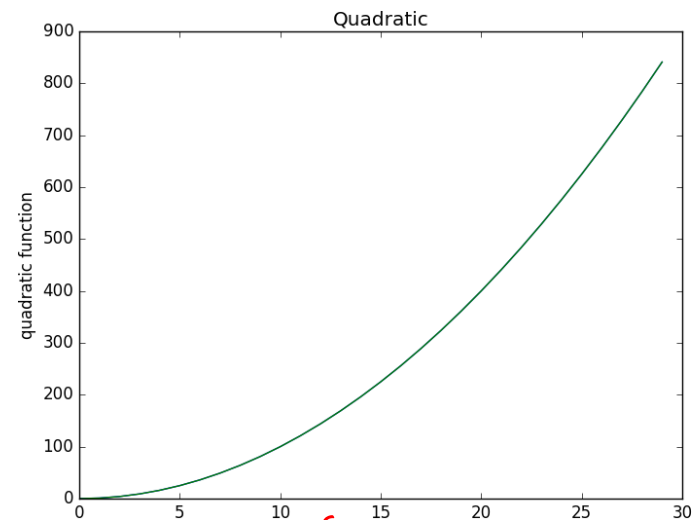
```
plt.figure('lin')
plt.plot(mySamples, myLinear)
plt.figure('quad')
plt.plot(mySamples, myQuadratic)
plt.figure('cube')
plt.plot(mySamples, myCubic)
plt.figure('expo')
plt.plot(mySamples, myExponential)
```

```
plt.figure('lin')
plt.title('Linear')
plt.figure('quad')
plt.title('Quadratic')
plt.figure('cube')
plt.title('Cubic')
plt.figure('expo')
plt.title('Exponential')
```


TITLED DISPLAYS



why are axes still
labeled?



why are colors
the same in the
two plots?

CLEANING UP WINDOWS

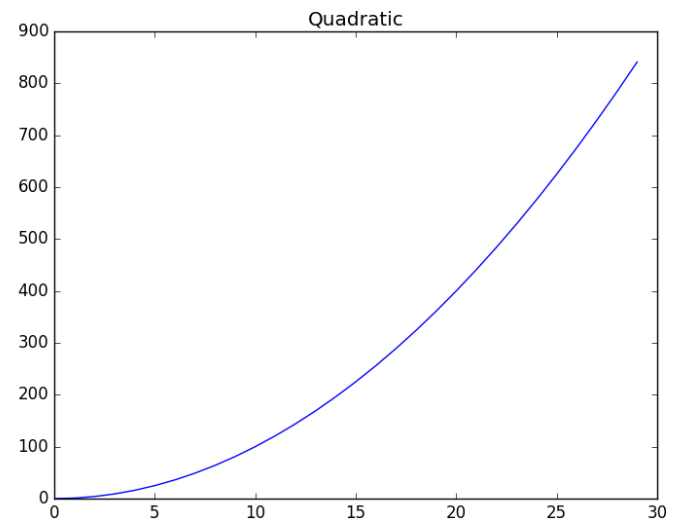
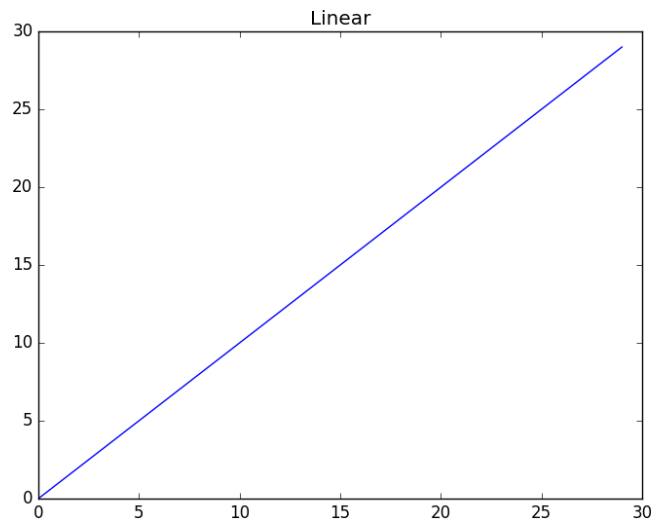
- we are reusing a previously created display window
- need to clear it before redrawing
- because we are calling plot in a new version of a window, system starts with first choice of color (hence the same); we can control (see later)

CLEANING WINDOWS

```
plt.figure('lin')
plt.clf()
plt.plot(mySamples, myLinear)
plt.figure('quad')
plt.clf()
plt.plot(mySamples, myQuadratic)
plt.figure('cube')
plt.clf()
plt.plot(mySamples, myCubic)
plt.figure('expo')
plt.clf()
plt.plot(mySamples, myExponential)
```

```
plt.figure('lin')
plt.title('Linear')
plt.figure('quad')
plt.title('Quadratic')
plt.figure('cube')
plt.title('Cubic')
plt.figure('expo')
plt.title('Exponential')
```

CLEARED DISPLAYS



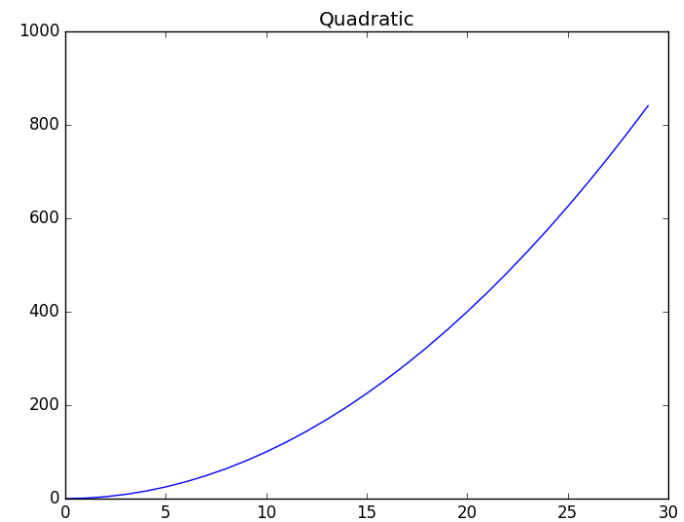
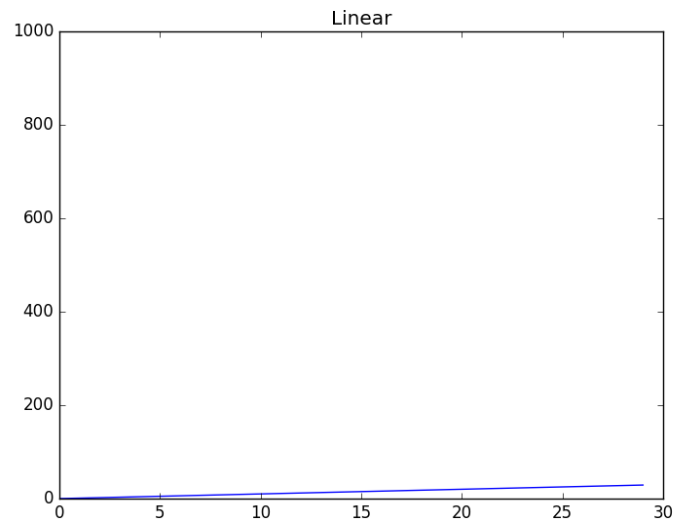
COMPARING RESULTS

- now suppose we would like to compare different plots
- in particular, the scales on the graphs are very different
- one option is to explicitly set limits on the axis or axes
- a second option is to plot multiple functions on the same display

CHANGING LIMITS ON AXES

```
plt.figure('lin')
plt.clf()
plt.ylim(0,1000)
plt.plot(mySamples, myLinear)
plt.figure('quad')
plt.clf()
plt.ylim(0,1000)
plt.plot(mySamples, myQuadratic)
plt.figure('lin')
plt.title('Linear')
plt.figure('quad')
plt.title('Quadratic')
```

CHANGING LIMITS ON AXES



OVERLAYING PLOTS

```
plt.figure('lin quad')  
plt.clf()
```

```
plt.plot(mySamples, myLinear)  
plt.plot(mySamples, myQuadratic)
```

*each pair of calls
within the same
active display
window*

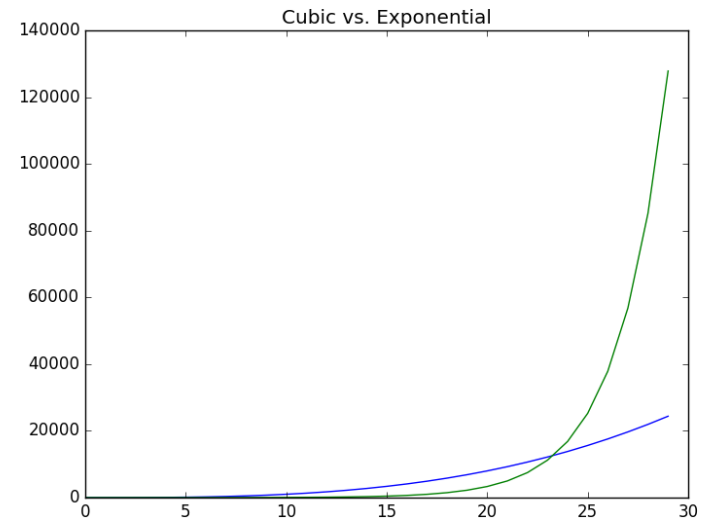
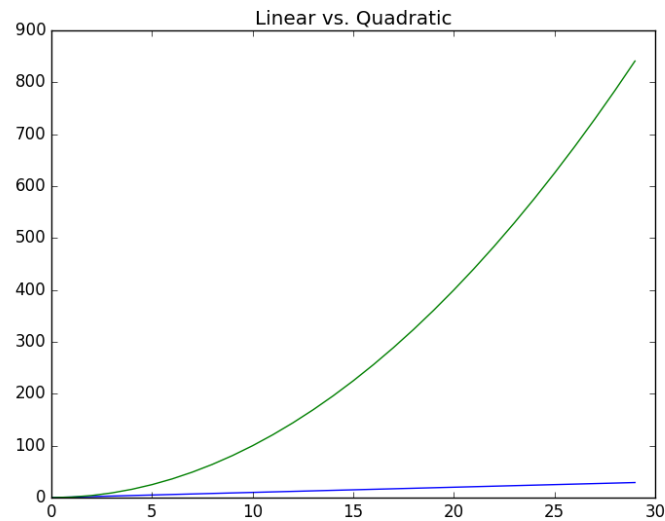
```
plt.figure('cube exp')  
plt.clf()
```

```
plt.plot(mySamples, myCubic)  
plt.plot(mySamples, myExponential)
```

```
plt.figure('lin quad')  
plt.title('Linear vs. Quadratic')  
plt.figure('cube exp')  
plt.title('Cubic vs. Exponential')
```

*each pair of calls
within the same
active display
window*

OVERLAYING PLOTS



ADDING MORE DOCUMENTATION

- can add a legend that identifies each plot

```
plt.figure('lin quad')
plt.clf()
plt.plot(mySamples, myLinear, label = 'linear')
plt.plot(mySamples, myQuadratic, label = 'quadratic')
plt.legend(loc = 'upper left')
plt.title('Linear vs. Quadratic')
```

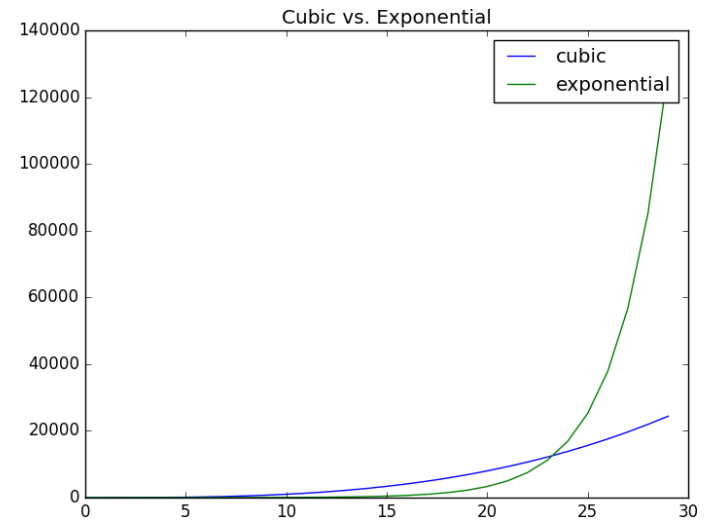
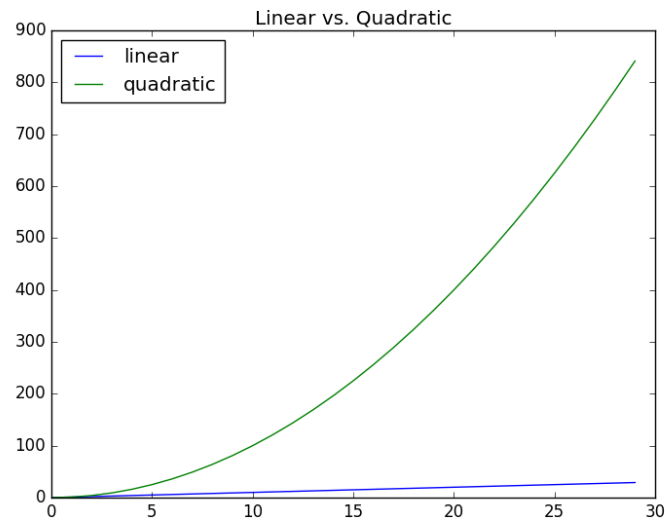
label each plot

can specify a location

```
plt.figure('cube exp')
plt.clf()
plt.plot(mySamples, myCubic, label = 'cubic')
plt.plot(mySamples, myExponential, label = 'exponential')
plt.legend()
plt.title('Cubic vs. Exponential')
```

can use best location

ADDING MORE DOCUMENTATION



CONTROLLING DISPLAY PARAMETERS

- now suppose we want to control details of the displays themselves
- examples:
 - changing color or style of data sets
 - changing width of lines or displays
 - using subplots

CHANGING DATA DISPLAY

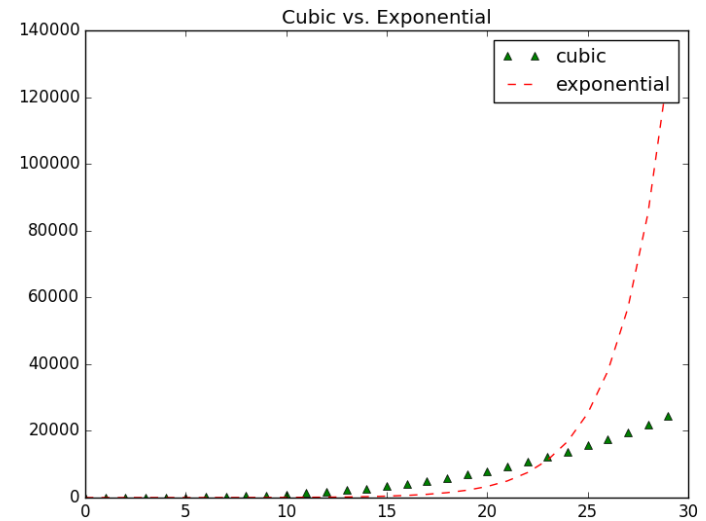
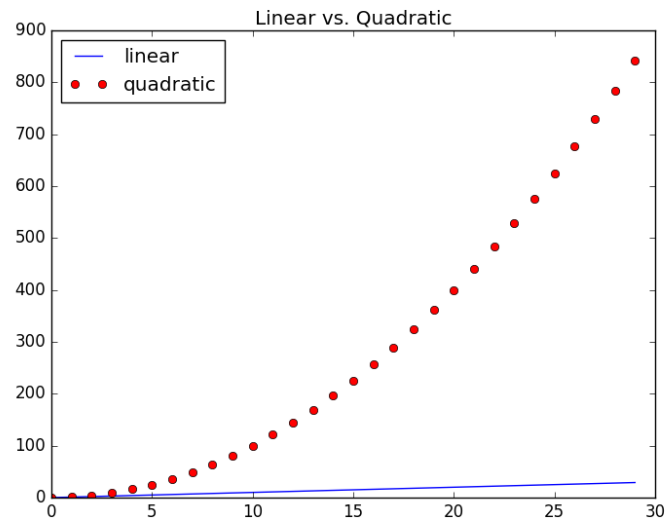
```
plt.figure('lin quad')
plt.clf()
plt.plot(mySamples, myLinear, 'b-', label = 'linear')
plt.plot(mySamples, myQuadratic, 'ro', label = 'quadratic')
plt.legend(loc = 'upper left')
plt.title('Linear vs. Quadratic')
```

string specifies color and style


```
plt.figure('cube exp')
plt.clf()
plt.plot(mySamples, myCubic, 'g^', label = 'cubic')
plt.plot(mySamples, myExponential, 'r--', label = 'exponential')
plt.legend()
plt.title('Cubic vs. Exponential')
```

see documentation for choices of color and style

CHANGING DATA DISPLAY



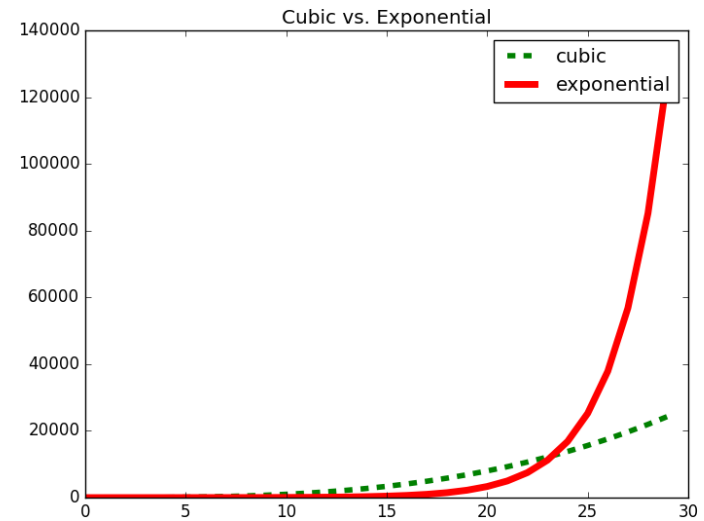
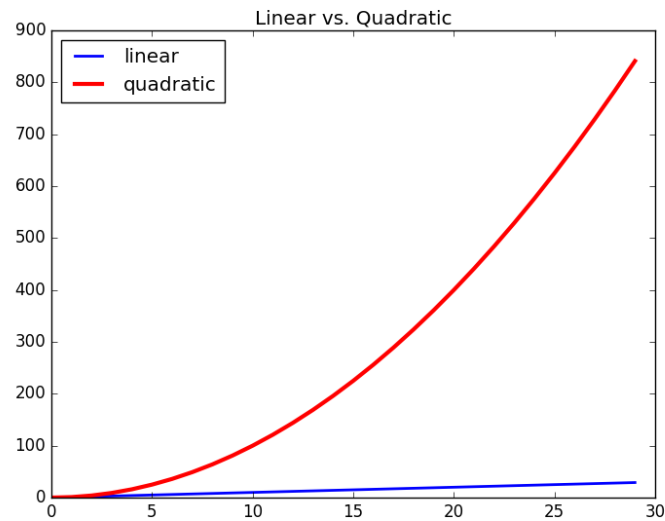
CHANGING DATA DISPLAY

```
plt.figure('lin quad')
plt.clf()
plt.plot(mySamples, myLinear, 'b-', label = 'linear', linewidth = 2.0)
plt.plot(mySamples, myQuadratic, 'r', label = 'quadratic', linewidth = 3.0)
plt.legend(loc = 'upper left')
plt.title('Linear vs. Quadratic')
```

*keyword can
change size of
parameter*

```
plt.figure('cube exp')
plt.clf()
plt.plot(mySamples, myCubic, 'g--', label = 'cubic', linewidth = 4.0)
plt.plot(mySamples, myExponential, 'r', label = 'exponential', linewidth = 5.0)
plt.legend()
plt.title('Cubic vs. Exponential')
```

CHANGING DATA DISPLAY



USING SUBPLOTS

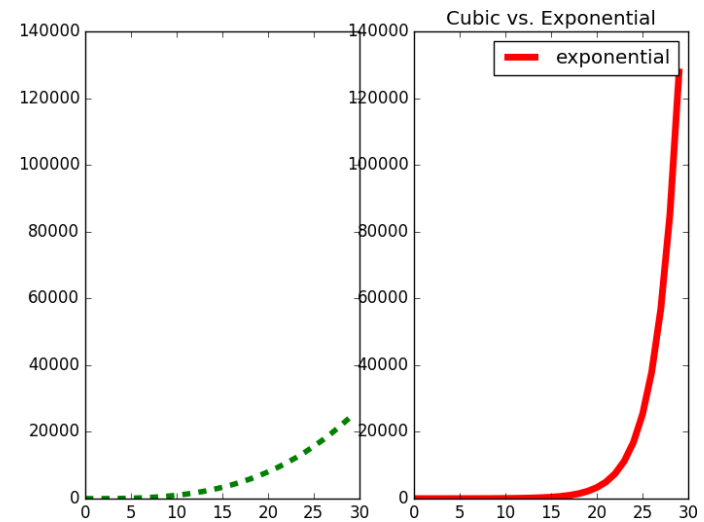
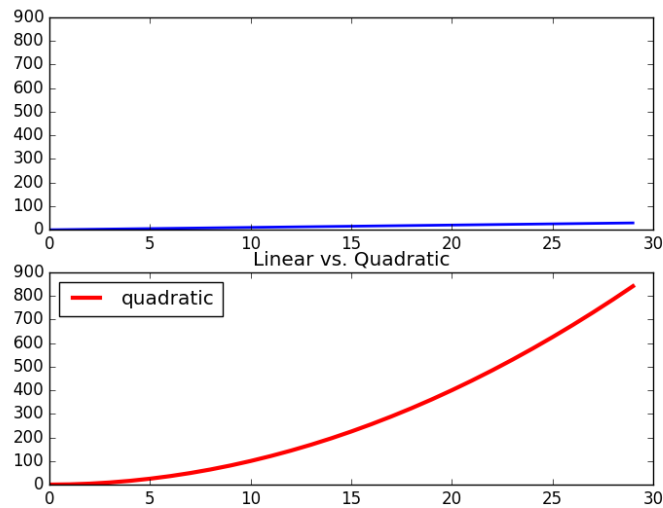
```
plt.figure('lin quad')
plt.clf()
plt.subplot(211)
plt.ylim(0,900)
plt.plot(mySamples, myLinear, 'b-', label = 'linear', linewidth = 2.0)
plt.subplot(212)
plt.ylim(0,900)
plt.plot(mySamples, myQuadratic, 'r', label = 'quadratic', linewidth = 3.0)
plt.legend(loc = 'upper left')
plt.title('Linear vs. Quadratic')
```

*arguments are
number of rows &
cols; and which
location to use*

*set limit within
each subplot*

```
plt.figure('cube exp')
plt.clf()
plt.subplot(121)
plt.ylim(0, 140000)
plt.plot(mySamples, myCubic, 'g--', label = 'cubic', linewidth = 4.0)
plt.subplot(122)
plt.ylim(0, 140000)
plt.plot(mySamples, myExponential, 'r', label = 'exponential', linewidth = 5.0)
plt.legend()
plt.title('Cubic vs. Exponential')
```

USING SUBPLOTS



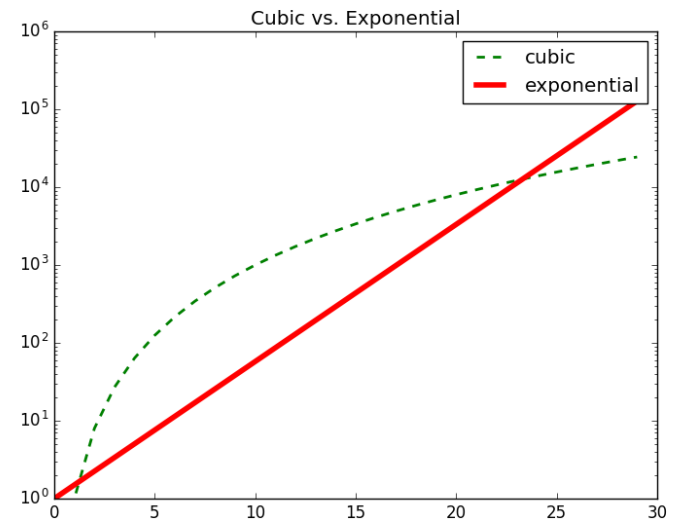
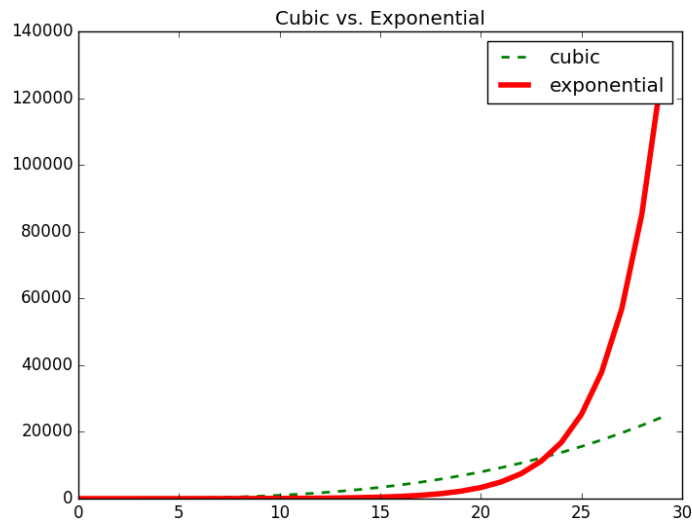
CHANGING SCALES

```
plt.figure('cube exp log')
plt.clf()
plt.plot(mySamples, myCubic, 'g--', label = 'cubic', linewidth = 2.0)
plt.plot(mySamples, myExponential, 'r', label = 'exponential', linewidth = 4.0)
plt.yscale('log')
plt.legend()
plt.title('Cubic vs. Exponential')
```

*argument specifies
type of scaling*

```
plt.figure('cube exp linear')
plt.clf()
plt.plot(mySamples, myCubic, 'g--', label = 'cubic', linewidth = 2.0)
plt.plot(mySamples, myExponential, 'r', label = 'exponential', linewidth = 4.0)
plt.legend()
plt.title('Cubic vs. Exponential')
```

CHANGING SCALES



AN EXAMPLE

- want to explore how ability to visualize results can help guide computation
- simple example
 - planning for retirement
 - intend to save an amount m each month
 - expect to earn a percentage r of income on investments each month
 - want to explore how big a retirement fund will be compounded by time ready to retire

AN EXAMPLE: compound interest

```
def retire(monthly, rate, terms):  
    savings = [0]  
    base = [0]  
    mRate = rate/12  
    for i in range(terms):  
        base += [i]  
        savings += [savings[-1]*(1 + mRate) + monthly]  
    return base, savings
```

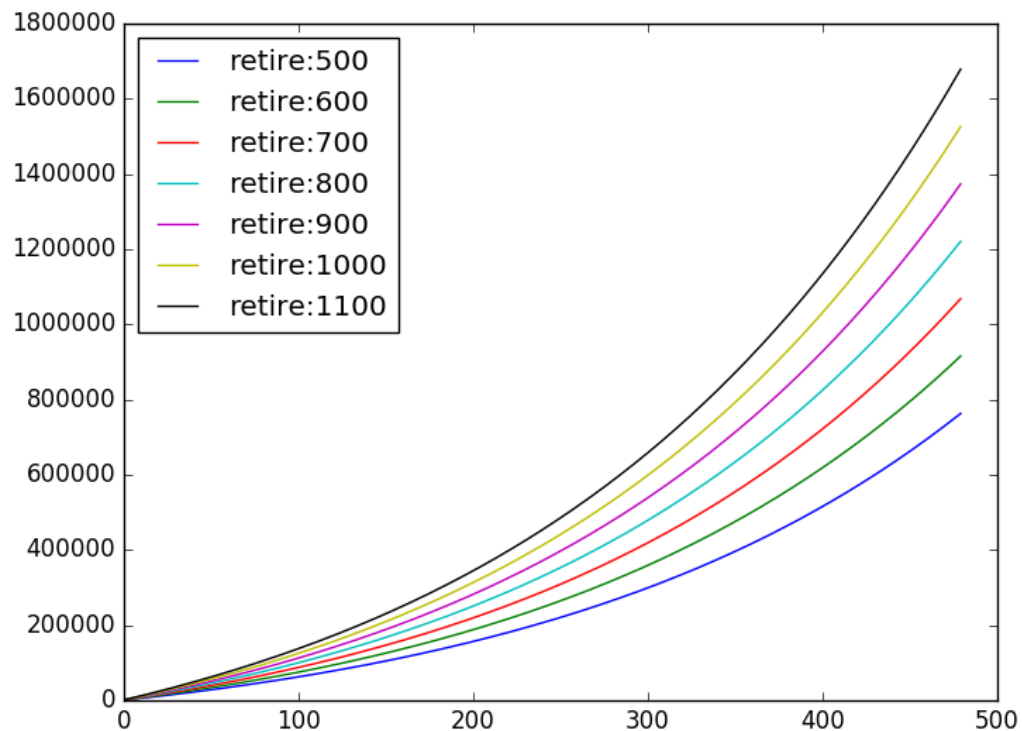
DISPLAYING RESULTS vs. MONTH

```
def displayRetireWMonthlies(monthlies, rate, terms):  
    plt.figure('retireMonth')  
    plt.clf()  
    for monthly in monthlies:  
        xvals, yvals = retire(monthly, rate, terms)  
        plt.plot(xvals, yvals,  
                 label = 'retire:' + str(monthly))  
        plt.legend(loc = 'upper left')  
  
displayRetireWMonthlies([500, 600, 700, 800, 900,  
                        1000, 1100], .05, 40* 12)
```

*clear frame so
can reuse*

*put
informative
label on each*

DISPLAYING RESULTS vs. MONTH



ANALYSIS vs. CONTRIBUTION

- can see impact of increasing monthly contribution
 - ranges from about 750K to 1.67M, as monthly savings ranges from \$500 to \$1100
- what is effect of rate of growth of investments?

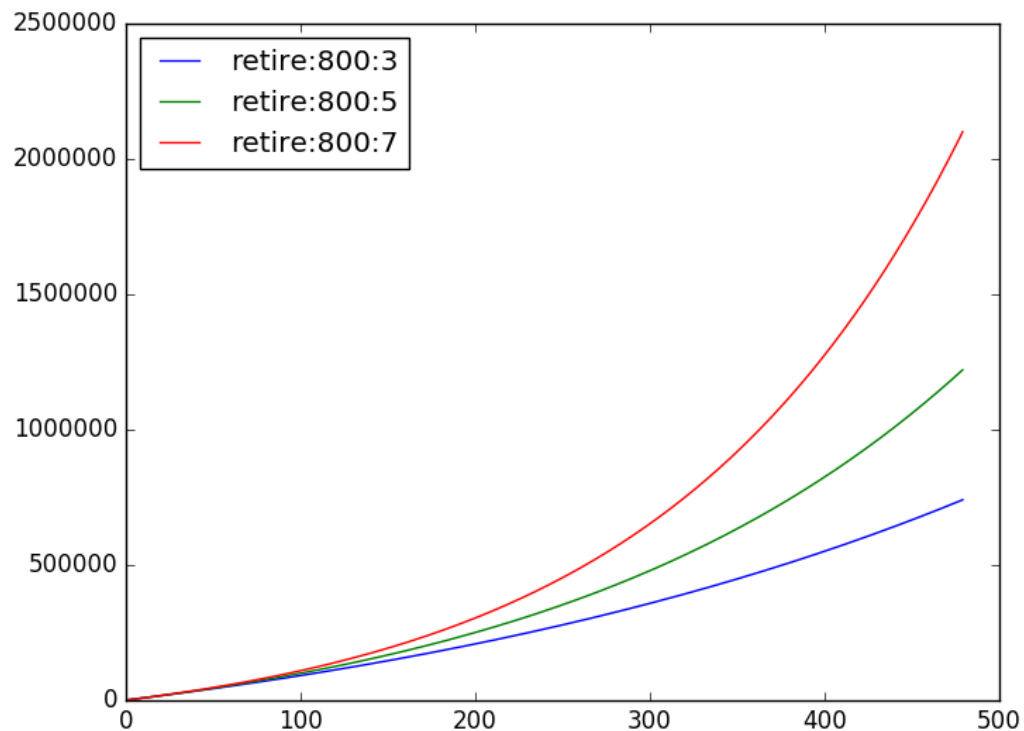
DISPLAYING RESULTS vs. RATE

```
def displayRetireWRates(month, rates, terms):  
    plt.figure('retireRate')  
    plt.clf()  
    for rate in rates:  
        xvals, yvals = retire(month, rate, terms)  
        plt.plot(xvals, yvals,  
                 label = 'retire:' + str(month) + ':' + \  
                        str(int(rate*100)))  
    plt.legend(loc = 'upper left')
```

```
displayRetireWRates(800, [.03, .05, .07], 40*12)
```

put
informative
label on each

DISPLAYING RESULTS vs. RATE



ANALYSIS vs. RATE

- can also see impact of increasing expected rate of return on investments
 - ranges from about 600K to 2.1M, as rate goes from 3% to 7%
- what if we look at both effects together?

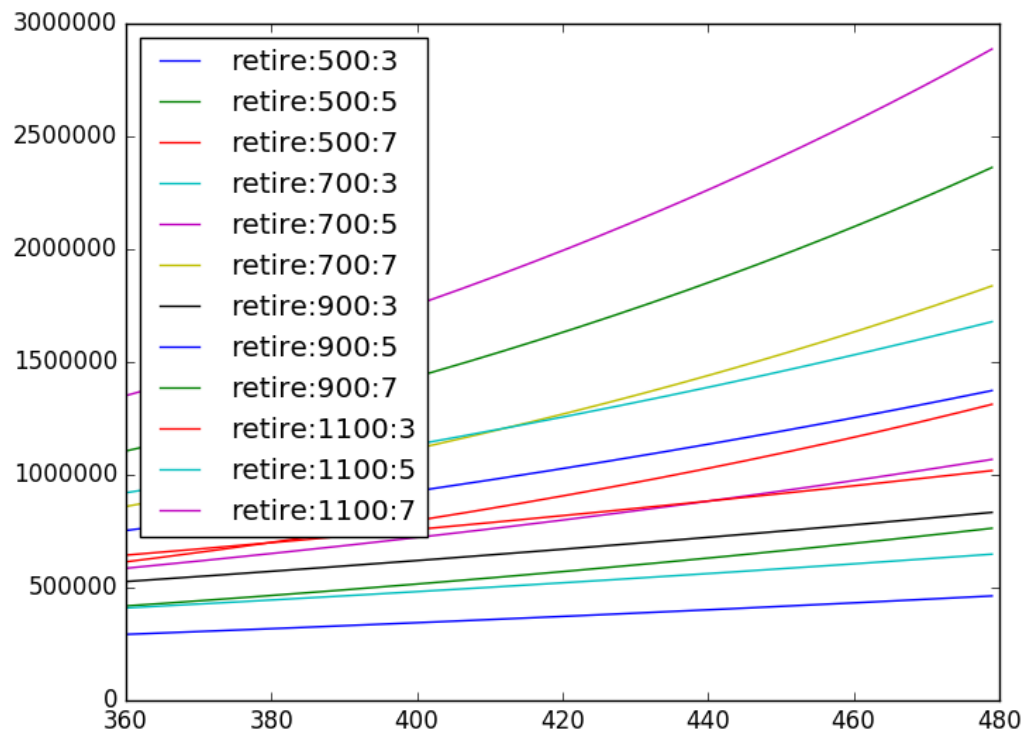
DISPLAYING RESULTS vs. BOTH

```
def displayRetireWMonthsAndRates(monthlies, rates, terms):  
    plt.figure('retireBoth')  
    plt.clf()  
    plt.xlim(30*12, 40*12)  
    for monthly in monthlies:  
        for rate in rates:  
            xvals, yvals = retire(monthly, rate, terms)  
            plt.plot(xvals, yvals,  
                    label = 'retire:' + str(monthly) + ':' \  
                        + str(int(rate*100)))  
    plt.legend(loc = 'upper left')  
  
displayRetireWMonthsAndRates([500, 700, 900, 1100],  
                             [.03, .05, .07],  
                             40*12)
```

*focus on last
stages of fund*

*put
informative
label on each*

DISPLAYING RESULTS vs. BOTH



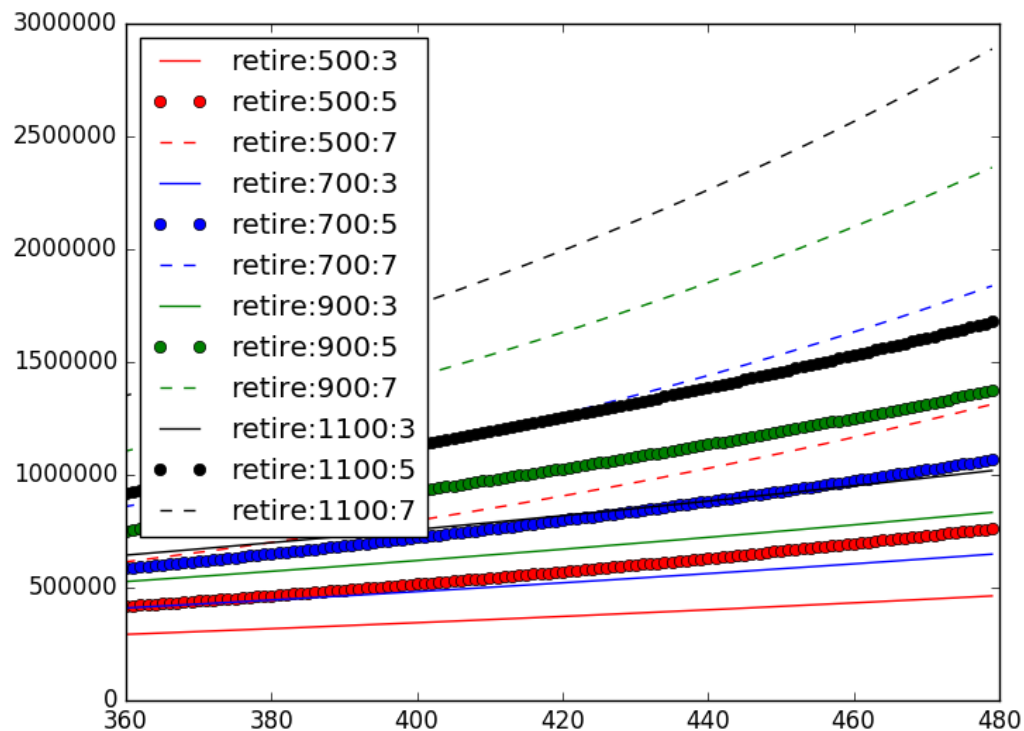
DISPLAYING RESULTS vs. BOTH

- hard to distinguish because of overlap of many graphs
- could just analyze separately
- but can also try to visually separate effects

DISPLAYING RESULTS vs. BOTH

```
def displayRetireWMonthsAndRates(monthlies, rates, terms):  
    plt.figure('retireBoth')  
    plt.clf()  
    plt.xlim(30*12, 40*12)  
    monthLabels = ['r', 'b', 'g', 'k']  
    rateLabels = ['-', 'o', '-']  
    for i in range(len(monthlies)): create sets of labels  
        monthly = monthlies[i]  
        monthLabel = monthLabels[i%len(monthLabels)] pick new label for each month choice  
        for j in range(len(rates)): pick new label for each rate choice  
            rate = rates[j]  
            rateLabel = rateLabels[j%len(rateLabels)] create label for plot  
            xvals, yvals = retire(monthly, rate, terms)  
            plt.plot(xvals, yvals,  
                    monthLabel+rateLabel,  
                    label = 'retire:' + str(monthly) + ':' \   
                        + str(int(rate*100)))  
    plt.legend(loc = 'upper left')  
  
displayRetireWMonthsAndRates([500, 700, 900, 1100], [.03, .05, .07],  
                             40*12)
```

DISPLAYING RESULTS vs. BOTH



DISPLAYING RESULTS vs. BOTH

- now easier to see grouping of plots
 - color encodes monthly contribute
 - format (solid, circle, dashed) encodes growth rate of investments
- interaction with plotting routines and computations allows us to explore data
 - change display range to zero in on particular areas of interest
 - change sets of values and visualize effect – then guides new choice of values to explore
 - change display parameters to highlight clustering of plots by parameter