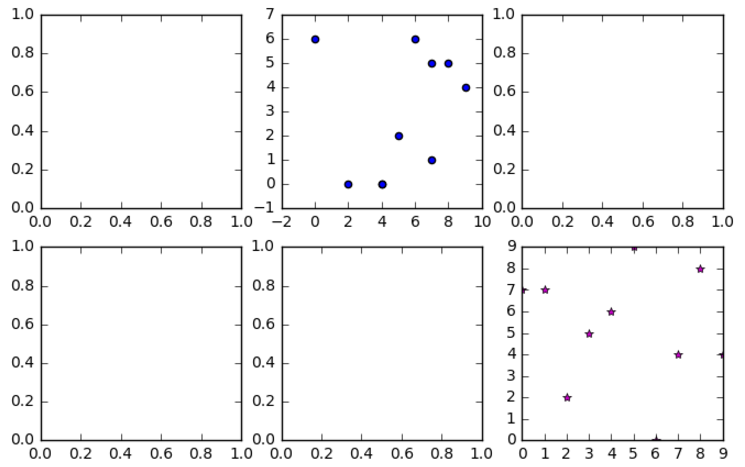
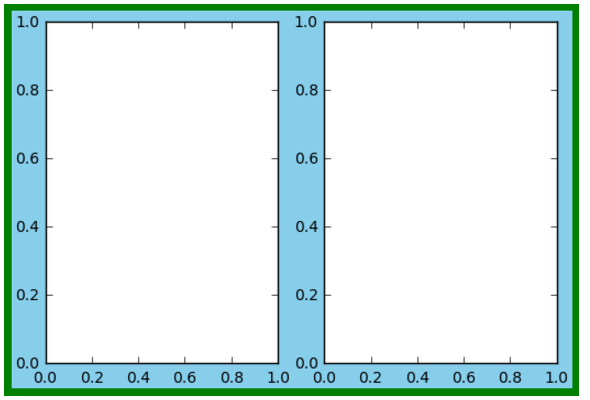
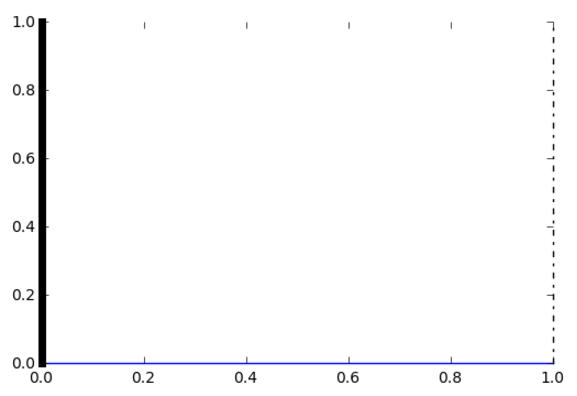
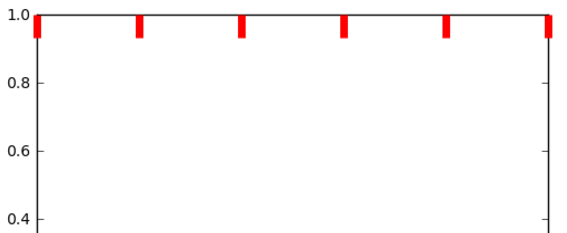
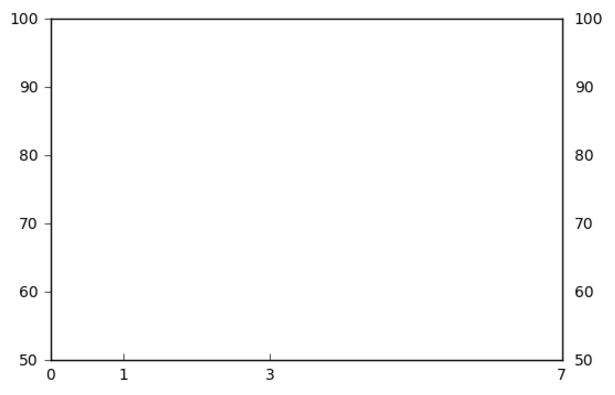
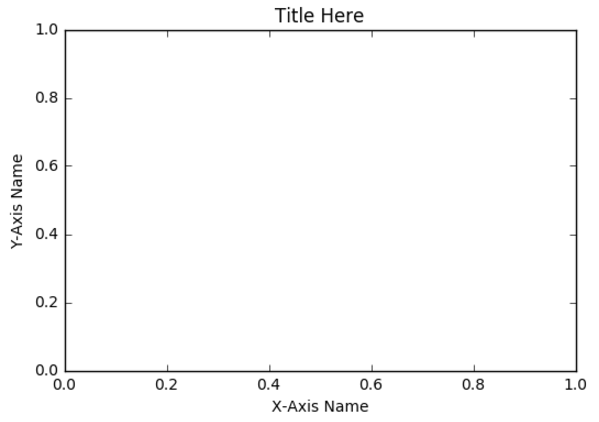
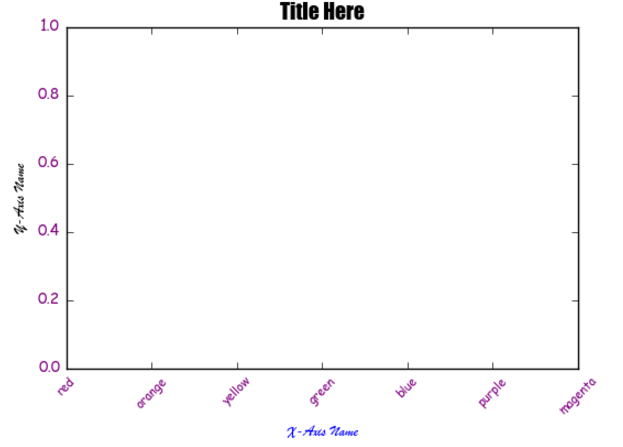
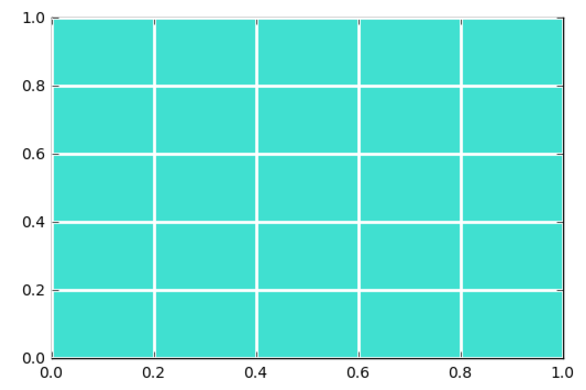
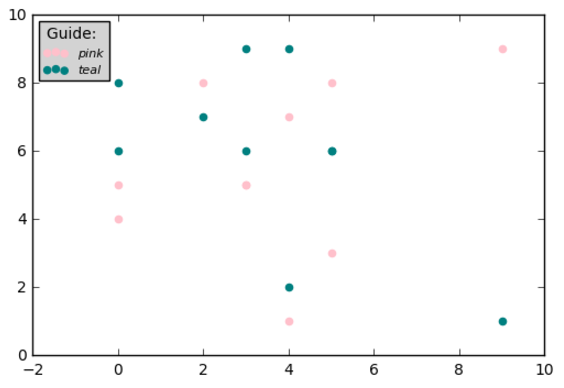
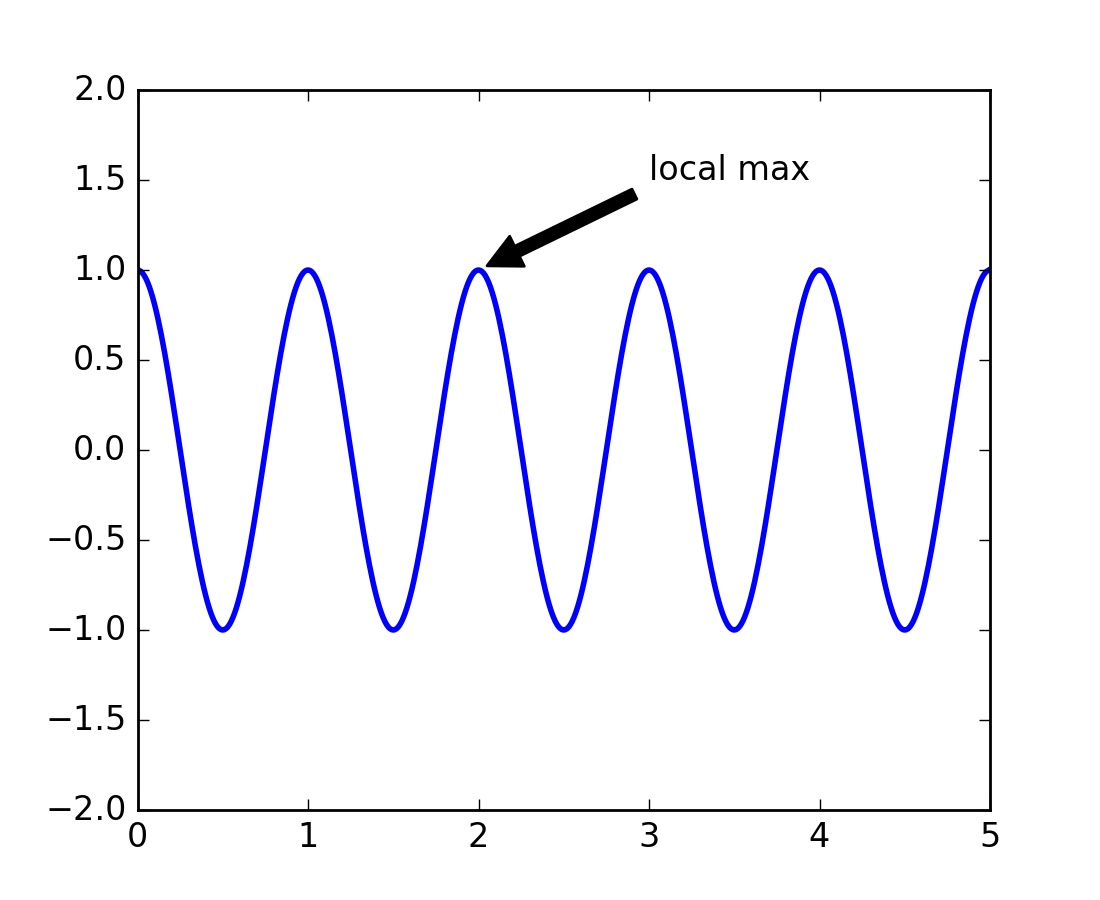
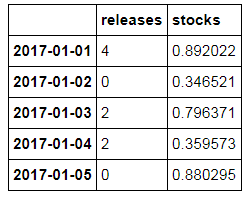
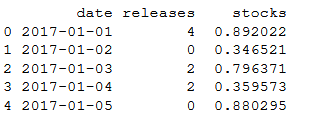
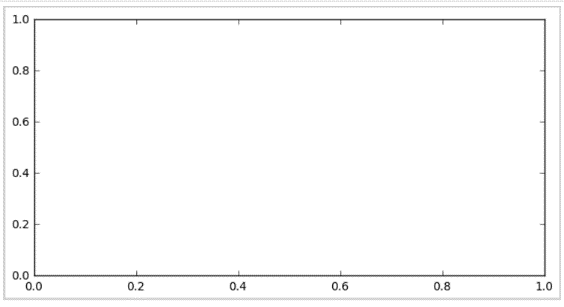
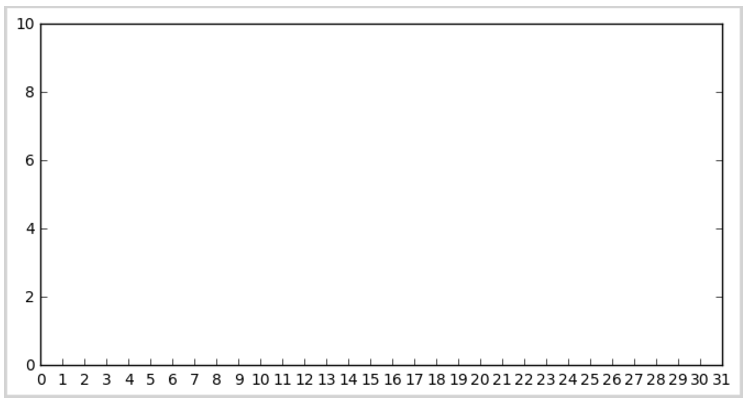
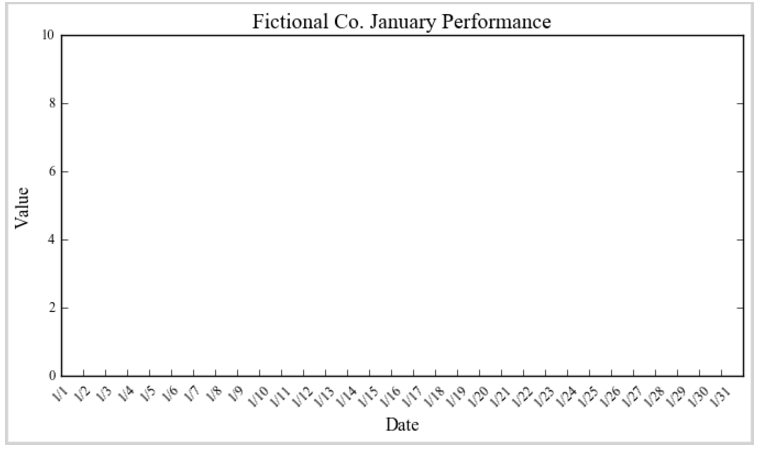
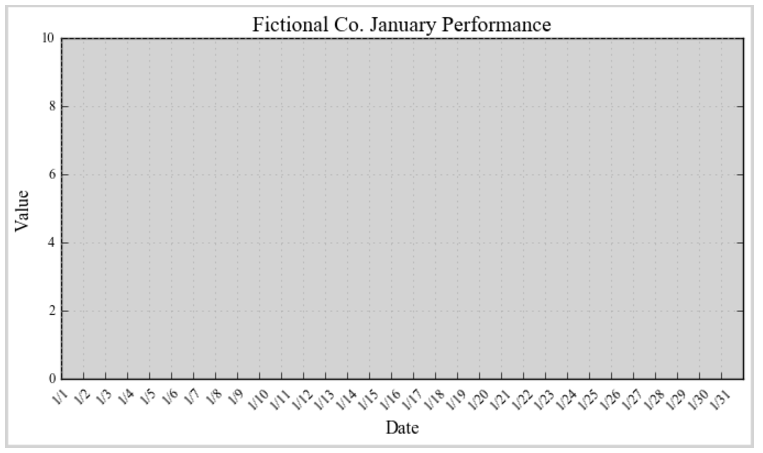
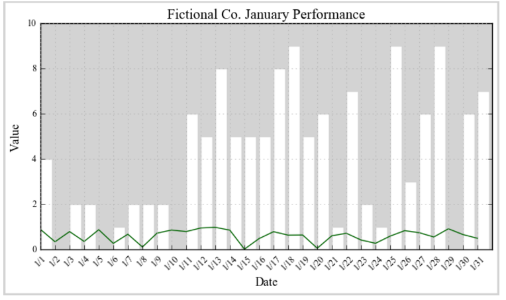
Unit 5-4: Intermediate MatplotLib

* Our Canvas
  + Remember that there are two ways to create the plot space in Matplotlib:
    - 1) Let the library create a default plot for you by going directly to a plotting function (e.g., plt.plot() or plt.scatter()).
    - 2) Create your own plot, consisting of a figure and an axes object, using plt.subplots() and then plot on to that (e.g., ax.plot()).
  + Note that it's possible to tell the .subplots() method to create more than one plot. However, you can add the parameters nrows or ncols to create multiple subplots, which would unpack into multiple ax objects. You need to define these as a tuple of lists — you'll need the same number of lists as rows and the same number of ax objects within each list as columns. You can then apply plotting functions to each of these axes.
  + Here's an example:
    - # Create some random data.
    - x = [random.randrange(10) for n in range(10)]
    - y = [random.randrange(10) for n in range(10)]
    - z = [random.randrange(10) for n in range(10)]
    - # Create plots.
    - fig, ([ax\_topL, ax\_topM, ax\_topR], [ax\_botL, ax\_botM, ax\_botR]) = plt.subplots(nrows=2, ncols=3, figsize=(8,5))
    - # Plot data.
    - ax\_topM.scatter(x, y)
    - ax\_botR.plot(x, 'm\*')
    - 
  + Following Along
    - If you want to follow along in your own IDE, remember that you'll need the following imports and settings:
      * import random
      * import matplotlib.pyplot as plt
      * % matplotlib inline
  + Figure
    - Think of the figure object — usually defined as fig — as the workspace for your plots. To make any changes to the overall output of your plotting, you'll need to adjust the figure's parameters.
    - This includes aspects such as the size of the overall output (figsize), color of the workspace (facecolor — effectively the background behind the charts), or a box around the outside of the whole area (edgecolor, linewidth). You can see all aspects available to change here.
    - Also, if you want to save the visualization, you'll need to output the whole item (fig is the parent of all axes objects, so fig.savefig(filename) will save your workspace and everything in it).
    - This should make sense because the figure does not change when you create multiple subplots — you still only have one fig. The axes objects simply get placed in this space.
    - fig, (ax1, ax2) = plt.subplots(ncols=2, facecolor='skyblue', linewidth=10, edgecolor='green')
  + 
  + How would you create a visualization that is 5 inches wide by 2 inches tall?
    - Fig,ax=plt.subplots(figsize=(2,5))
  + Axes
    - The real fun of Matplotlib customization, however, comes in manipulating the axes. If the figure sets up the workspace, the axes are the canvases in that studio. The plots are the actual images on those canvases, but you've already seen how to [apply plotting functions](https://matplotlib.org/api/axes_api.html#plotting) to the axes (ax.plot(), ax.scatter(), etc.) and know that each of these methods has its own parameters. In this lesson, we'll just be focusing on how to customize the graphs themselves.
    - While there are [tons of capabilities](https://matplotlib.org/api/axes_api.html#plotting) within the axes class, we're going to focus on:
      * Changing the borders around the graph.
      * Changing the placement, spacing, appearance, and range of the tick marks.
      * Adjusting the font and size of axis tick labels.
      * Adding labels, titles, and a legend.
      * Adjusting the chart space with grids and colors.
      * Adding text and annotations.
  + Borders
    - The borders around a Matplotlib chart are called spines, and they are attributes of the ax object. There are four (top, right, bottom, and left), and they can all be turned on or off, as well as customized.
    - Because the spines are just Line2D objects, you can use any of the Line2D methods (the ones that begin with set\_) to customize them.
    - Here are some of the customizations you can apply (all are called using the syntax ax.spines[whichspine].attribute(value)):
      * set\_visible() can be True or False and turns the spine on or off.
      * set\_color() changes the spine's color.
      * set\_position() changes where the spine is placed on the axis (i.e., a coordinate, 'center', or 'zero').
      * set\_linewidth() changes the thickness of the spine.
    - fig, ax = plt.subplots()
    - ax.spines['top'].set\_visible(False)
    - ax.spines['right'].set\_linestyle('dashdot')
    - ax.spines['bottom'].set\_color('blue')
    - ax.spines['left'].set\_linewidth(5)
    - 
  + Ticks
    - You may have noticed that, even when we remove spines in the previous graph, the ticks on that axis remain. That's because these are separate objects. That's good news — it means you can customize them individually! We'll get to labels in a bit; for now we're just concerned with the actual tick marks. There are two ways to customize these: their *appearance* and their *placement*.
    - Appearance
      * Use the ax.tick\_params() method to customize the ticks' appearance. Check out [the documentation](https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.tick_params.html#matplotlib.axes.Axes.tick_params) for all of the available options, but for now, the main ones to remember include:
        + axis can be 'x', 'y', or 'both', and determines the axis to which to apply these changes.
        + which can be 'major', 'minor', or 'both' and specifies which set of ticks to modify.
        + color does the obvious (colors changes the label color, too).
        + length adjusts tick length in points.
        + width adjusts tick thickness.
      * ax.tick\_params(axis='x', which='major', color='red', length=15, width=5)
      * 
      * Reminder: If you are following along on your own, you will need to make sure to set the fig and ax with each execution (Hint: Make sure you use fig, ax = plt.subplots()).
    - Placement
      * You can adjust all sorts of aspects of the ticks' placement.
      * Some of these adjustments use .tick\_params() (Again, use the axis and which keywords to specify where to apply these):
        + Turn ticks on or off with ax.tick\_params(side=on|off) (the labels will also be removed unless you use ax.tick\_params(labelside=on|off)).
        + Place the ticks inside or outside of the axes with ax.tick\_params(direction=in|out|inout).
      * Some have their own methods:
        + Set the location and number of the ticks by passing a list of numbers to ax.set\_xticks(list) or ax.set\_yticks(list).
        + Add autoscaled minor ticks between the major ones with ax.minorticks\_on() (or get rid of them with ax.minorticks\_off()).
        + Set the range limits of the ticks with ax.set\_xlim(start, stop) or ax.set\_ylim(start, stop).
      * ax.tick\_params(axis='y', direction='out') # Place the y axis ticks outside of the axes.
      * ax.tick\_params(top='off') # Turn off the top ticks AND their labels.
      * ax.tick\_params(right='off', labelright='on') # Turn off the right ticks but KEEP their labels.
      * ax.set\_xticks([0, 1, 3, 7]) # Place ticks at specific locations along the x axis.
      * ax.set\_ylim(50, 100) # Set the range of the y axis.
      * 
  + Labels and Titles
    - This is one of the more straightforward adjustments you can make to a chart. Later, we'll discuss how to format fonts, but to add labels and titles to your charts, simply use the following syntax:
      * Title: ax.set\_title('title').
      * X-axis label: ax.set\_xlabel('label').
      * Y-axis label: ax.set\_ylabel('label').
    - ax.set\_title('Title Here')
    - ax.set\_xlabel('X Axis Name')
    - ax.set\_ylabel('Y Axis Name')
    - 
  + Tick Labels
    - Unless you specify otherwise, your tick marks will be labeled with the coordinate of their placement. Bar charts allow you to include labels with the tick\_labels parameter, but what if you wanted to include labels in a different kind of chart? Or customize those labels? The axes class has you taken care of.
    - To add labels to any chart, use ax.set\_xticklabels(list) or ax.set\_yticklabels(list). You can also define the rotation of the labels by passing a number into the rotation keyword within these methods (see example below).
    - There are some specific methods for formatting these labels, but generally you will want to use more general customization functions, most of which are contained in our friend .tick\_params(). Some of the more useful ones include:
      * labelsize in points (int) or as a string (e.g., 'large').
      * labelcolor as a string or Matplotlib color spec.
      * labelbottom, labeltop, labelright, and labelleft can be set to 'on' or 'off' to indicate whether or not to draw the labels.
    - Remember, you use axis ('x', 'y', or 'both') and which ('major', 'minor', or 'both') to specify the labels to which these parameters apply.
    - colors = ['red','orange','yellow','green','blue','purple','magenta']
    - ax.set\_xticks(range(len(colors))) # Arrange the ticks along the x axis.
    - ax.set\_xticklabels(colors, rotation=45) # Add labels to the ticks.
    - ax.tick\_params(axis='x', labelcolor='red', labelsize=8) # Change the color and font size of the labels.
    - 
* Fonts
  + You may want to change the font, size, or color of the text in your chart.
  + You can easily change the **font of the title or axes labels** by including fontname, fontsize, or color keywords when defining them: ax.set\_title('Title', fontname='Arial', fontsize=8, color='blue'). You can see all possible [fonts](http://jonathansoma.com/lede/data-studio/matplotlib/list-all-fonts-available-in-matplotlib-plus-samples/) here.
  + There are several ways to change the text of **tick labels**:
    - 1) When creating the labels with ax.set\_xticklabels(list), you can add parameters for color or size (as discussed on the last slide).
    - 2) You can set these using ax.tick\_params() with the parameters labelcolor or labelsize.
    - 3) If you want to actually change the font of the labels, you'll need to use a for loop to iterate through the ticks and set the parameters for each (see example below). You can use any of the [text methods](https://matplotlib.org/api/text_api.html) that start with 'set', including .set\_fontname(), .set\_fontsize(), and .set\_color().
  + *If you use the third option, because it's a for loop, you could actually change it differently for each tick by iterating through a list!*
    - # Titles
    - ax.set\_title('Title Here', fontname='Impact', fontsize=14) # Set chart title and change font and font size.
    - ax.set\_xlabel('X-Axis Name', fontname='Brush Script MT', color='blue') # Set x label and change font and font color.
    - ax.set\_ylabel('Y-Axis Name', fontname='Brush Script MT') # Set y label and change font.
    - #Tick Labels
    - ax.tick\_params(labelcolor='purple') # Change color of all tick labels.
    - for tick in ax.get\_xticklabels():
    - tick.set\_fontname('Comic Sans MS') # Loop through x-tick labels, changing font of each.
    - for tick in ax.get\_yticklabels():
    - tick.set\_fontname('Comic Sans MS') # Repeat for y-tick labels.
* Chart Background
  + We learned how to change the figure background earlier (which surrounds each chart), but what about modifying the chart space itself? There are two main aspects we can control.
  + First, you can make a grid by using ax.grid(True) and can [customize it further](https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.grid.html#matplotlib.axes.Axes.grid) if you'd like. The grid, like the spines, is made up of Line2D objects.
  + You can then change the color of the plot area using ax.set\_facecolor(color).
  + Note: In earlier versions, this could be set with ax.set\_axis\_bgcolor(), so if .facecolor() isn't working, try bgcolor(). This example uses the older version.
  + These are simple commands, but they can dramatically change the look of your chart.
  + ax.set\_facecolor('turquoise') # If this doesn't work, try ax.set\_facecolor(color).
  + ax.grid(True, color='white', linestyle='-', linewidth=2)
  + 
* Legend
  + When plotting multiple sets of data on the same chart, you'll usually want to include a guide of what is what. Including a legend makes your visualizations easier to understand and interpret. There are a few steps to including this element:
  + 1) Include a label parameter for each plot when you create it (e.g., ax.scatter(x, y, label='name')).
    - An alternative is to define each plot as a variable (e.g., s1 = ax.scatter(x, y)) and then pass a tuple of variables and a tuple of labels to your legend method (ax.legend((s1, s2), ('label1', 'label2'))).
  + 2) After you've made your plots, include ax.legend() to create the legend on your chart. This is [highly customizable](https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.legend.html#matplotlib.axes.Axes.legend), but a few key tools include:
    - loc: This can be a string (like 'upper right' or 'center left') or an integer for exact coordinate placement (bbox\_to\_anchor is another way to choose placement).
    - ncol: The number of columns.
    - prop: Sets font properties (fontsize also exists; does not include color).
    - title: Adds a title.
  + 3) Make customizations to the legend background using the .get\_frame() method and its submethods, the most useful of which is .set\_facecolor().
  + ax.scatter(x, y, color='pink', label='pink')
  + ax.scatter(x, z, color='teal', label='teal')
  + legend = ax.legend(loc='upper left', prop={'size':8, 'style':'italic'}, title='Guide: ')
  + legend.get\_frame().set\_facecolor('lightgrey')
  + 
* Text and Annotation
  + A final way to customize your graphs is by adding text or annotations. This is a bit tricky, so no worries if it takes some time to pick up!
    - You can [add text](https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.text.html#matplotlib.axes.Axes.text) to your chart by simply telling it what to write where, and you can include a dictionary to control the font style (Syntax: ax.text(x, y, string, fontdict={})).
    - [Annotation](https://matplotlib.org/users/annotations_intro.html) takes text one step further by allowing you to name both the point of the annotation and the text you want to describe it. You can also use both text and arrows to annotate a chart.
  + 
* Connecting to Pandas and Seaborn
  + In the last lesson, we learned that Seaborn allows for style customization. But you can also pass customized axes objects into Seaborn to give it a jump start. Additionally, you can pass them into a Pandas plot to customize those graphs as well.
    - Seaborn: As long as you are creating a single Seaborn plot object, you can include an ax parameter and bring in your Pyplot axes object: sns.regplot(x, y, ax=ax) (This will not work for pair plots or other Seaborn objects that create several axes at once).
    - Pandas: This works in Pandas as well. Just add the ax keyword to your plotting function (e.g., df.plot(ax=ax) or df.plot.bar(ax=ax)) to place your plot on your customized axes object.
  + The key with both of these is that you must have an actual axes object. If you just used plt.plot() and haven't figured out the name of the axes object you created, you will not be able to share it with these other libraries
* Plotting Walk Through: The Data
  + Now let's put it all together. We're going to walk through the entire process of creating a visualization.
  + To begin, we need some data. We'll be charting data for both the closing stock prices and the number of press releases of a fictional company over the last month.
  + Remember to set up your libraries:
    - import pandas as pd
    - import matplotlib.pyplot as plt
    - %matplotlib inline
  + We'll assume that the following data have already been cleaned and processed via the first several steps of EDA (remember Lesson 2 from this unit).
    - df = pd.read\_csv('assets/fictionalco.csv', index\_col=0)
    - df.head()
  + 
    - NOTE: I could not find the original file so I recreated the table using the following code:
      * import numpy as np
      * import pandas as pd
      * df = pd.DataFrame(np.array([["2017-01-01",4,.892022], ["2017-01-02",0,.346521], ["2017-01-03",2,.796371], ["2017-01-04",2,.359573], ["2017-01-05", 0, .880295]]), columns=["date","releases","stocks"])
      * df['date']=pd.to\_datetime(df.date)
      * print (df)
      * 
* Plotting Walk Through:
  + The Figure
    - Our first step is to create our overall workspace. We make our figure object when we first create the subplots — at the same time as the axes — by using the code fig, ax = plt.subplots().
    - This is the time to add any customizations to the overall output, such as the figure size or a line around the visual.
    - The figure should automatically display when you create it.
    - fig, ax = plt.subplots(figsize=(8, 4), edgecolor='lightgrey', linewidth=3)
    - 
  + Ticks
    - Now we can format our axes to show the correct placement and appearance of tick marks.
      * # Find number of data points to fit on each axis.
      * n\_x = len(df)
      * n\_y = max(df.releases)
      * #X-Axis
      * ax.set\_xlim(0, n\_x) # Increase the width of x axis to fit all of our data points.
      * ax.set\_xticks(range(n\_x + 1)) # Set tick marks as evenly spaced for the correct number of days.
      * ax.tick\_params(axis='x', top='off') # Turn off the top ticks for a cleaner view,
      * #Y-Axis
      * ax.set\_ylim(0, n\_y+1) # Increase height of y axis to fit all values.
    - 
  + Titles and Labels
    - With the graph set up, we should now add a title, axis labels, and tick labels.
      * #X-Axis
      * days = [(str(d.month) + '/' + str(d.day)) for d in df.index] # Format the strings we want as the x axis labels.
      * ax.set\_xticklabels(days, rotation=45, size=9, fontname='Times New Roman') # Feed our labels into the graph and format them.
      * ax.set\_xlabel('Date', size=12, fontname='Times New Roman') # Set a label for the axis as a whole.
      * #Y-Axis
      * ax.set\_ylabel('Value', size=12, fontname='Times New Roman') # Label the y axis.
      * for tick in ax.get\_yticklabels(): # Because we don't want to reset the labels entirely, we'll add formatting to each one.
      * tick.set\_fontname('Times New Roman')
      * tick.set\_fontsize(9)
      * #Chart
      * ax.set\_title('Fictional Co. January Performance', size=14, fontname='Times New Roman')
      * 
  + The Plot Area
    - Now that everything around the edges is tuned, we want to prepare the space for the chart. In this case, we'll make the background a light gray and add a subtle, dark gray grid on top of that.
    - ax.grid(True, color='darkgrey') # Add a grid with dark gray lines.
    - ax.set\_facecolor('lightgrey') # Set the plot background to light gray.
    - 
  + Plot the Data
    - Your space is now set up! It's time to add the data by using the various Pyplot plotting methods we learned earlier. Remember that each of these allows for customization of the output using parameters within the method, so feel free to adjust as much as you'd like in this step.
    - Remember to add a label parameter when plotting so you can create the legend.
    - ax.bar(range(n\_x), df.releases, color='white', edgecolor='lightgrey', label='Press Releases')
    - ax.plot(df.stocks.values, color='darkgreen', label='Closing Stock Price')
    - 
  + Add a Legend
    - The only thing that's missing now is a quick visual guide to the data in the form of a legend. Because you added label keywords while plotting, you can simply add it by running ax.legend(). At this time, you can add parameters to customize the legend, too.
    - ax.legend(loc='upper right', prop={'size':8, 'style':'italic'})
    - 