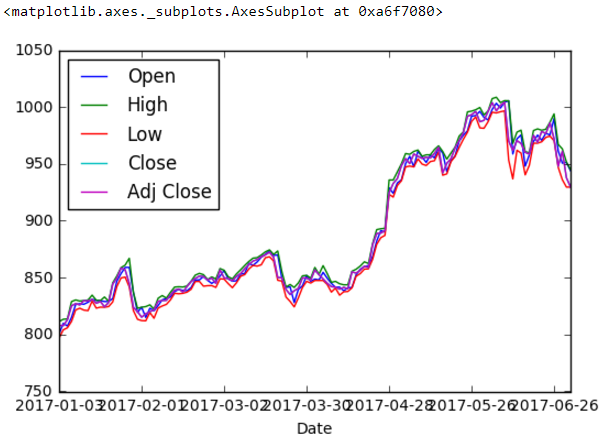
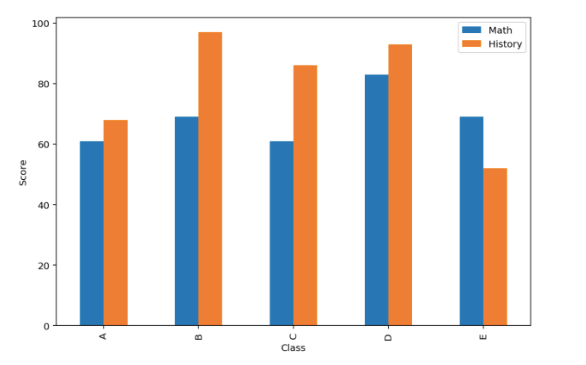
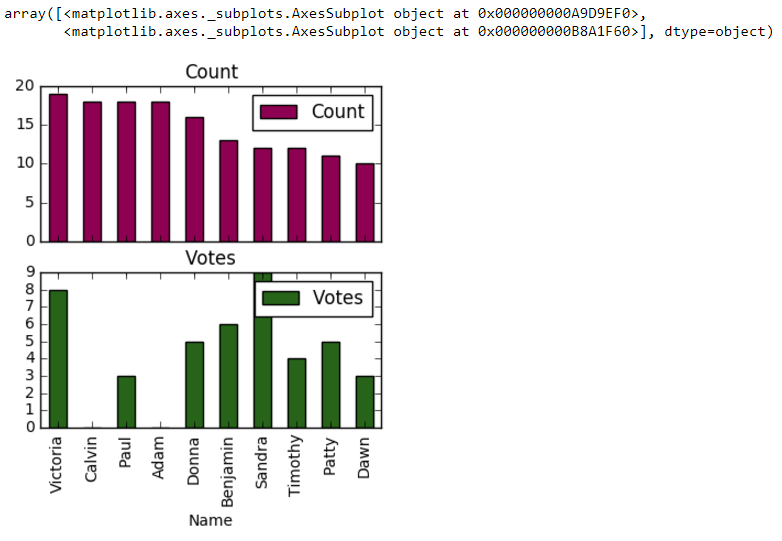
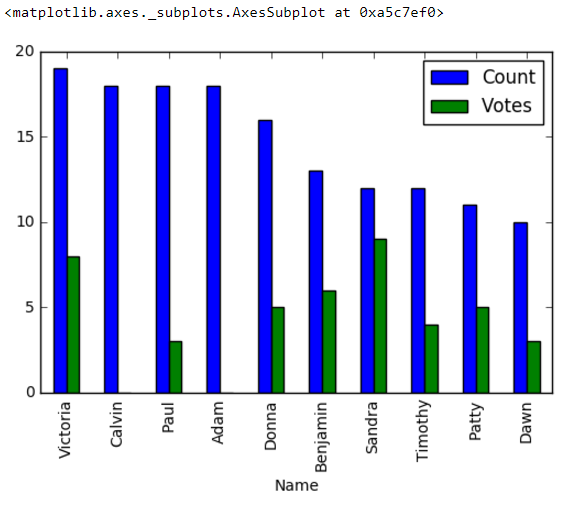
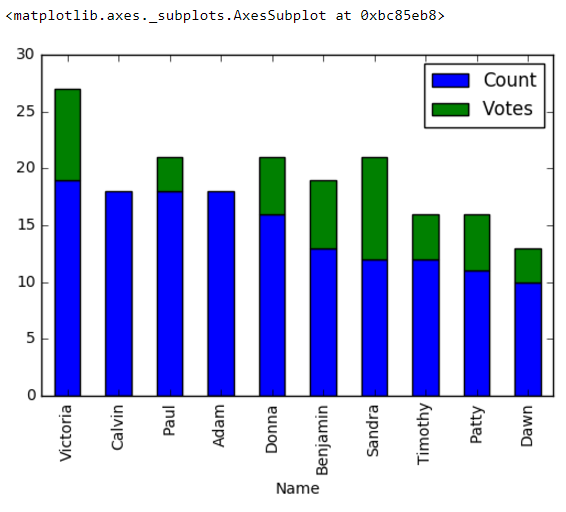
Unit 5-2: Plotting with Pandas

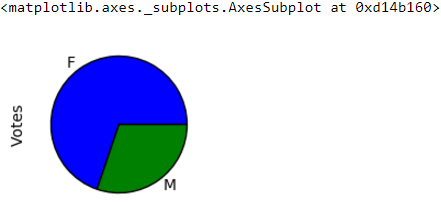
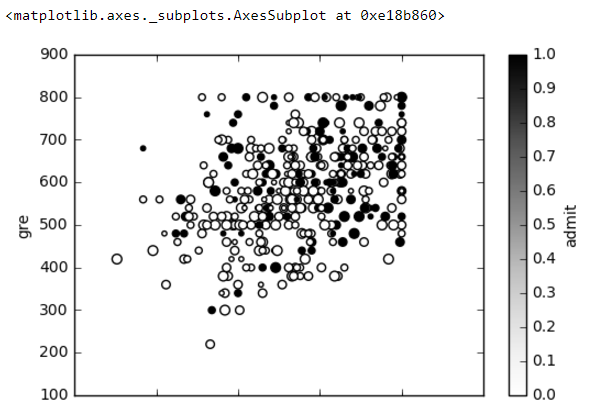
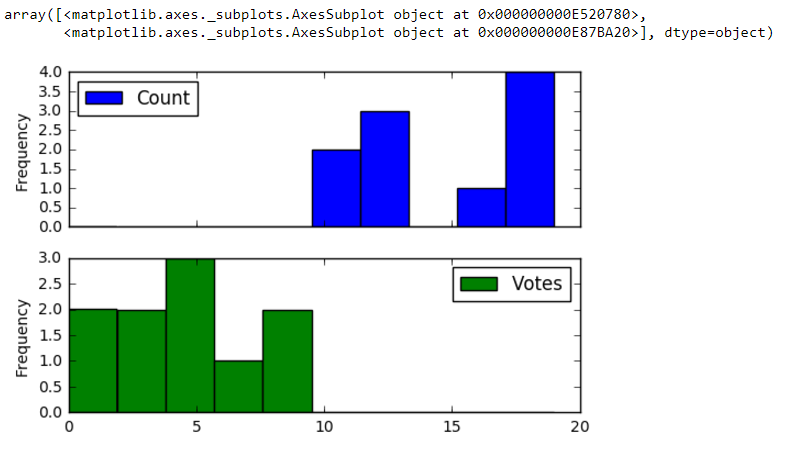
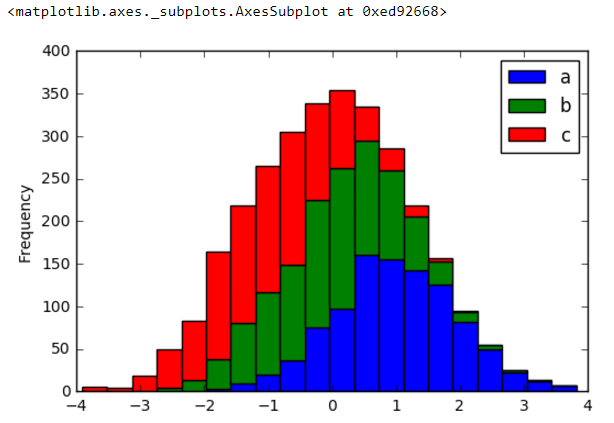
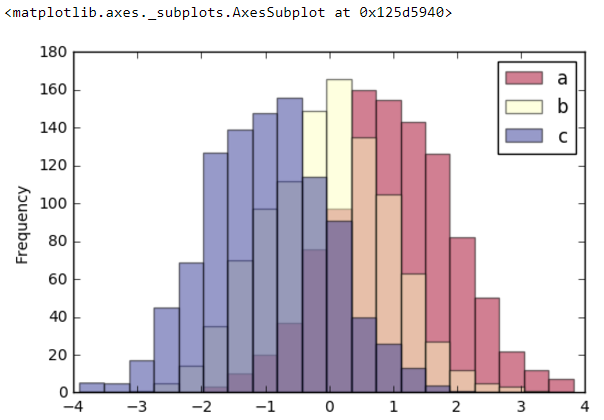
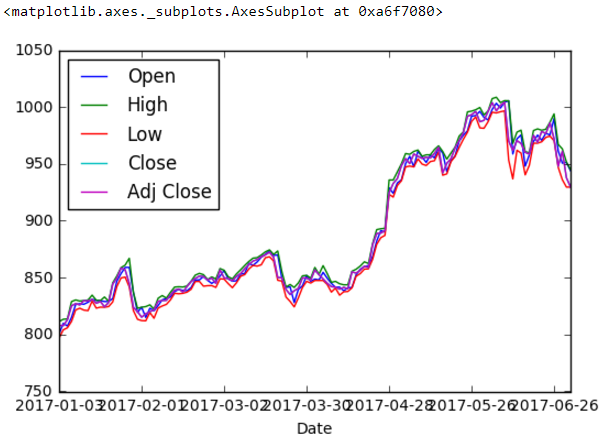
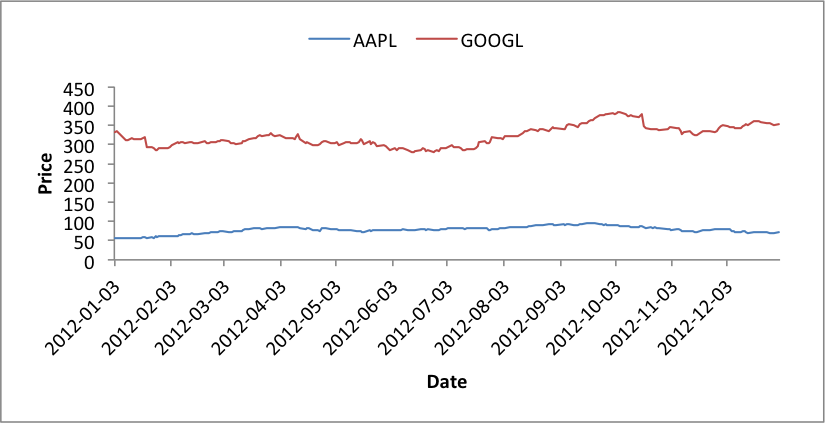
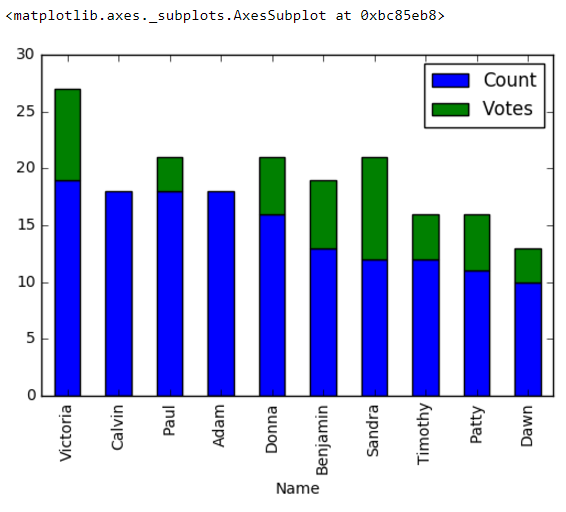
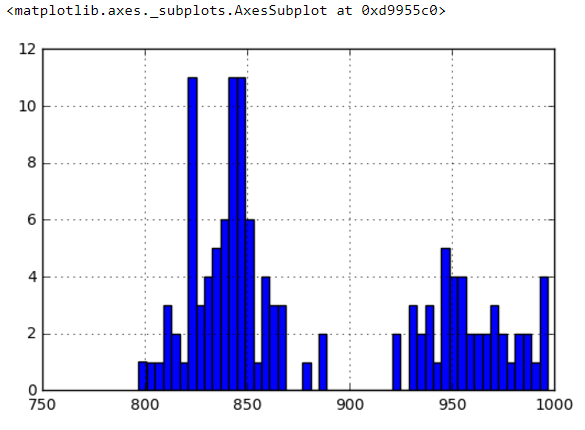
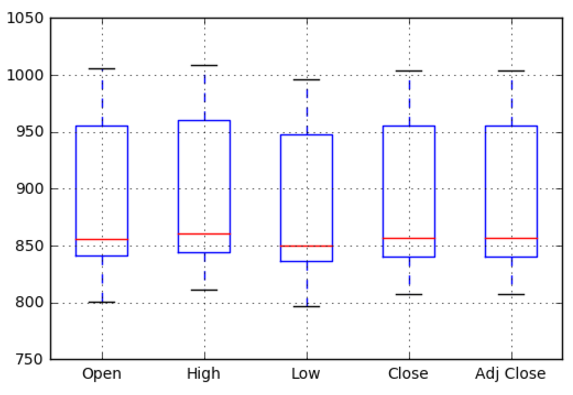
* Introduction: Plotting in Pandas
  + The majority of data manipulation and exploration in this course will be done in Pandas. Lucky for us, the library is decked out with visualization capabilities.
  + It's also possible to use Matplotlib and pull in data from Pandas, but applying plotting functions directly to your DataFrame is often the simplest way to go.
  + All of Pandas' plotting functions are wrappers around Matplotlib functions (meaning that they simply point to the same script), so they're identical in their processes and output. Note that, whenever we want to use Pandas to visualize data, we download Matplotlib.
* Import Pandas
  + For our coding purposes, we'll use the conventional importing process:
    - import pandas as pd
    - import matplotlib.pyplot as plt
    - % matplotlib inline
* Default Chart: df.plot()
  + Is your data set in Pandas? Great, then you're ready to start plotting! The default plot type in Pandas is a line graph, but, because this chart generally has limited uses, you'll likely want to change that setting. The syntax for the most basic type of chart is: df.plot(x='columnname', y='columnname').
  + You can use this default if your data have a datetime index (meaning that all of the rows show data for a specific date), or if the index itself tracks a steadily increasing variable. If your index itself is not appropriate to use as the x axis, you can still use this default chart, but you'll need to specify which columns to plot against one another by passing x and y parameters to the .plot() function.
  + Here's an example of a DataFrame with a datetime index, plotted using only the default command:
  + 
* Chart Types and The kind Parameter
  + In general, you'll want to specify the chart type you're planning to produce from your data. Do this by passing a kind keyword into the .plot() method:
    - df.plot(kind='yourchoice')
  + Pandas makes the following charts available:
    - bar or barh for bar plots.
    - hist for histograms.
    - box for box plots.
    - kde or density for density plots.
    - area for area plots.
    - scatter for scatterplots.
    - hexbin for hexagonal bin plots.
    - pie for pie plots.
* Simple Plotting With .plot(kind='')
  + In Pandas, if you specify a chart kind but not columns, you'll get Pandas' interpretation of which columns are worth plotting (usually the numeric ones). It's much more effective if you tell it which columns you want to plot.
  + For example, to make a bar chart, you need to include the following parameters:
    - kind='charttype' (For example, bar to make a bar chart — you can turn the bars horizontal using barh.)
    - x='columnname' (Your x (horizontal) axis.)
    - y='columnname' (Your y (vertical) axis.)
  + Example: kind='bar'
  + Here's an example that plots the math and history scores for 5 different classes. We've set the class names along the x axis and have plotted the bar chart to show the scores for each subject.
  + 
* Simple Plotting: Additional Parameters
  + Besides 'kind', x, and y, there are several other useful parameters in the Pandas .plot() function.

|  |  |  |  |
| --- | --- | --- | --- |
| **Keyword** | **Default Value** | **Possible Values** | **Result** |
| ax | None | Name of Matplotlib object | Connects a plot to a Matplotlib plot object for additional customization. |
| subplots | FALSE | True, False | Splits each column's plot into its own distinct visualization. |
| figsize | None | Tuple (width, height) in inches | Changes the size of the resulting plot. |
| title | None | String (or list if you're using subplots) | Adds a title to the graph. |
| grid | None | True, False | Turns the grid on or off. |
| legend | TRUE | True, False | Adds a legend to the plot. |
| style | None | List or dictionary | [Applies a Matplotlib style to the plot or subplots.](https://matplotlib.org/users/style_sheets.html) |
| xlim and ylim | None | 2-tuple | Min and max values for the given axis. |
| rot | None | int | Rotation for axis labels. |
| fontsize | None | int | Font size for axis ticks. |
| colormap | None | String | [Applies a Matplotlib colormap.](https://matplotlib.org/examples/color/colormaps_reference.html) |

* + Other parameters to explore are layout, sharex/sharey, use\_index, logx/logy/log log, xticks/yticks, table, xerr/yerr, stacked, secondary\_y, and sort\_columns. Check out their documentation [here](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html#pandas.DataFrame.plot).
  + 
* Better Plotting: Embedded Methods
  + Rather than using 'kind' as an argument within .plot(), you'll have more control over your charts by taking the .plot() method one level deeper and using methods specific to different chart types, such as .plot.pie() or .plot.bar(). Whereas .plot() can only handle 'kind', x, y, and the other broad parameters we listed earlier, each of these methods allows for control over the exact aspects of the type chart that would be important.
  + When using these functions on either a DataFrame or Series, you can add keywords/parameters from any of the following sources:
  + 1) The overall df.plot() function (e.g., x, y, figsize, etc.). 2) The narrower function's arguments. 3) The matching Matplotlib function.
  + We'll go into the detailed exploration of chart types in the following slides.
* Bar Charts
  + Overview: Bar charts are simple to create in Pandas — their syntax is almost identical to the default. In any version of possible syntax, you can use bar for vertical columns and barh for horizontal ones. Remember that your x must be categorical and your y must be numerical.
  + Unique method parameters: In reality, only an x column is required so that you know the categories into which you're splitting the data. Using y allows you to specify one or more columns of which to measure height.
  + Correct usage: Bar charts illustrate some summary statistic (e.g., count, sum, average, etc.) for categorical data.
  + Syntax: df.plot(kind='bar', x='col', y='col') or df.plot.bar(x='col', y='col').
  + Examples:





* Pie Charts
  + Overview: Pie charts are often simple, especially because they usually only show the values in a single column. It's common to use pie charts after .group by() statements in Pandas, as you're often displaying the distribution within groups.
  + Unique method parameters: Like bar charts, pie charts' unique method doesn't require any additional parameters. You can use subplots=True to create pie charts for each column rather than just a single one.
  + Correct usage: Pie charts should only be used to display a percentage or relative proportions of categorical data.
  + Syntax: df.plot.pie(y='column'), df.column.plot.pie(), or df.plot(kind='pie', y='column') — all will produce identical graphs.
  + 
* Scatterplots
  + Overview: Scatterplots can be some of the most visually appealing charts and are among the few that allow you to visualize a third and fourth dimension of data fairly easily by using color and size.
  + Unique method parameters: Scatterplots require both x and y parameters and allow us to customize the size (s=) and color (c=) of the data points. 'c' should be the column you'd like represented by color variation, and you can add 'colormap=' to set a colormap scheme. 's' can either be a single integer, to make the data points a specific size, or a column's values (although you must use the full df['column']), to represent yet another dimension of the data.
  + Correct usage: Scatterplots explore the relationship between two variables and work best when those variables have continuous, numerical values.
  + Syntax: df.plot(kind='scatter', x='col', y='col') (which will not allow you to customize size or color) or df.plot.scatter(x='column', y='column', c='column', s=numeric).
  + Example: The scatterplot below shows the relationship between the scores students received on the GRE, a standardized test that is often used to determine acceptance into graduate programs, and admittance.
  + 
* Histograms
  + Overview: Histograms add another level of sophistication by having their own method, which can be called directly on a DataFrame, as opposed to only within the .plot() method (although they have that, too).
  + Unique method parameters:
    - df.plot.hist(): by='col' allows you to specify the column you want to group by, although this isn't often useful. It's more effective to use bins=int, which allows you to declare the number of bins by which you want to divide your data.
    - df.hist(): This is slightly different in that by splits the histogram into subplots by the defined column and 'column' is used in place of x. 'bins' is also available, as well as several customization options.
  + Correct usage: Histograms are effective for examining the distribution of your data set as it divides into bins along a continuous, numerical scale.
  + Syntax: df.plot.hist() or df.plot(kind='hist') (which are identical in output), or df.hist() (which splits variables into subplots and can be further customized).
  + Example, which shows the difference in the two plotting methods:
  + 
* Layering Charts
  + We include this trick here because it's most frequently used for histograms, but it's also good to know for other charts. When trying to include two histograms on the same plot, you'll need to prevent one from hiding the other. There are two common ways to go about this:
  + 1) Stacking: As we saw with bar charts, the stacked keyword allows bars to appear on top of one another. This is best for combining variables to see how they are distributed in aggregate.
  + 
  + 3) 2) Transparency: The alpha parameter, which can be set at any float between 0 and 1, defines the opaqueness of the plots. If you define alpha to allow for transparency, it's possible to see the histograms through one another. This is best for comparing distributions of individual variables.
  + 
* Line Graphs
  + Overview: Line graphs are the default chart type in Pandas, which we discussed earlier. Calling the .plot() function without passing 'kind' or narrowing the method will return a line graph.
  + Unique method parameters: None, only the default parameters apply.
  + Correct usage: Line graphs illustrate the life or trends of a continuous variable along an ordered, continuous variable (such as time).
  + Syntax: df.plot(data).
  + Example:
* 
* Exploratory Data Analysis in Pandas
  + One of the key uses of visualization in Pandas is EDA — exploring your data set when you first begin working with it. Before you visualize, remember to do the following:
    - 1) Check out the .head() and .tail() of the data to get a sense of what they look like.
    - 2) Check for nulls and decide how to handle them (Drop? Fill? With what?).
    - 3) Check the data types of the columns using .dtypes() and adjust as necessary.
    - 4) Remove any columns you're not interested in including and make sure the names of remaining columns are clear.
    - 5) Get the summary statistics of the data using .describe().
  + Once you've completed those steps, try visualizing the data for more insights. Examples follow on the next slides.
  + Note: The following visualizations ran a line from Matplotlib, allowing a particular style to be set — you will learn how to do this in later lessons.
* Check For Trends
  + If (and only if!) your data are sequential — i.e., your index shows a meaningful progression of a variable, or the data are collected over time — look for trends by creating a line graph. Remember that Pandas will automatically graph any numerical columns unless you tell it otherwise.
  + df.plot()
  + 
* Check the Distribution
  + If your data are broken into distinct categories, use a bar chart to examine the distribution of data among the categories.
  + df['column'].plot(kind='bar')
  + 
  + If your data instead fall along a continuous scale, use a histogram to examine the distribution.
  + df['column'].hist(bins=50)
  + 
  + You may want to run both to illustrate different columns, but remember to either call them on specific columns (as in these examples), or, if calling on the entire DataFrame, be clear in your x and y parameters.
  + You can also learn about the distribution — and identify outliers — by creating a box plot. A library called Seaborn creates more elaborate box plots, but you can get a basic idea from Pandas' version. It'll help you see outliers and assess the mean, spread, and skew of the data.
  + df.boxplot()
  + 
* Check Relationships
  + Later we'll learn about more advanced charts for comparing variables to one another (e.g., heat maps and pair plots). For now, let's explore the classic scatterplot. You can run a loop to compare all columns or only plot the ones you're most interested in.
  + df.plot(kind='scatter', x='col1', y='col2')
  + 