Lab 14

Data Visualization

Lab Objective: *Use data visualizations to explore data and communicate effectively to others.*

The purpose of visualization is insight, not pictures. —Card, Mackinlay, & Shneiderman (after Hamming)

The ability to take data—to be able to understand it, to process it, to extract value from it, to visualize it, to communicate it—that's going to be a hugely important skill in the next decades,... because now we really do have essentially free and ubiquitous data. So the complimentary scarce factor is the ability to understand that data and extract value from it. —Hal Varian (Google's Chief Economist)

What is Data Visualization?

Data visualizations (or graphs) are used to understand and explore data as well as communicate results to others. Often data is more easily interpreted or understood in graphical format than as a list of numbers or data points.

Types of Visualizations

Here are some of the most useful types of data visualizations. We explain each of these in more detail below.

1. A *bar chart* is generally the preferred method for displaying a small (discrete) collection of one-dimensional data points. Each discrete data point is represented as a bar whose length is determined by the value of the data (see Figure 14.1).
2. A *histogram* is a special type of bar graph where the length of each bar corresponds to the number of data points in a certain range (sometimes called a *bin*). Histograms are useful for revealing statistical distributions (see Figure 14.11).
3. A *line plot* plots (x, y)-pairs as points and connects them with a curve. You should use a line plot for graphing continuous functions or when there is a natural order to your two-dimensional data—for example when the data is a time series (a sequence of values over time).

4. A *scatter plot* plots (x, y) tuples as discrete points. This should be used instead of a line plot when the data is two-dimensional but has no natural order.

   A scatter plot can reveal correlation (or lack thereof) between x and y.

5. Three-dimensional data can be displayed on a two-dimensional page with a *contour plot*. This draws lines in the plane where the third value is constant—like a topographical map. Filling in the contours with successive colors gives a *heat map* (see Figure 14.8).

   Three-dimensional data can also be plotted as a surface in 3-space, but it is often difficult to find a view where none of the important features of the data are obscured. A contour plot or a heat map can help avoid this problem and is often easier for the viewer to decode.

Bar Charts

Bar charts are best for relatively small sets of discrete, one-dimensional data. They are usually best presented with the bars running horizontally because when the bars are vertical the labels tend to run together. The other possible solution to that problem—making the text vertical—is harder to read (see Figure 14.1).

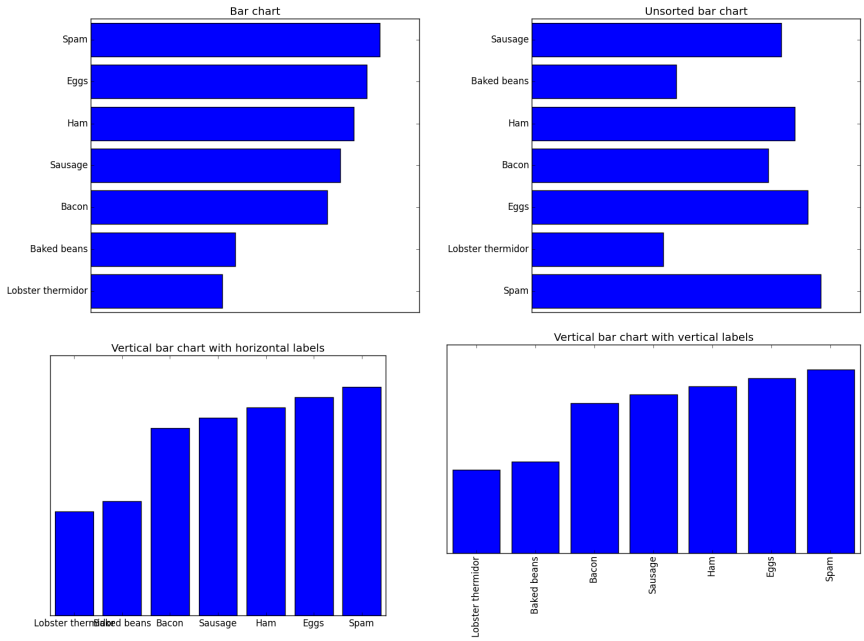

Figure 14.1: Bar Charts. Note how the good example in the upper left is easier to use for comparison than the upper right, and easier to read than the bottom two charts. Also, the labels in the bottom left are overlapping and the labels in the bottom right are vertical, making it hard to read.

You can create a bar chart in matplotlib with the commands `plt.bar()` and `plt.barh()`.

```
# This code created the top right bar chart.
labels = 'Spam', 'Eggs', 'Ham', 'Sausage', 'Bacon', 'Baked beans','Lobster ↩
    thermidor'
```

```
val = [10, 11, 18, 19, 20, 21, 22]    # the bar lengths
pos = np.arange(7)+.5    # the bar centers on the y axis
plt.barh(pos,val, align='center')
plt.yticks(pos, labels[::-1])
plt.show()
```

The data in a bar chart should generally be sorted in some natural way, based on the intended use. If the chart will be used to compare the value of different data points, we usually sort by size; if the chart will be used for looking up specific values, the data could be sorted alphabetically or in some other logical order that facilitates finding specific values.

As a general rule, bars should always start at zero because people are conditioned to expect that. Failing to do this can be confusing or misleading.

Problem 1. The following table provides average heights of men and women in 17 countries. Order the data and plot a horizontal bar chart.

Country	Average Male Height	Average Female Height
Austria	179.2 cm (5 ft 7.5 in)	167.6 cm (5 ft 6 in)
Bolivia	160.0 cm (5 ft 3 in)	142.2 cm (4 ft 8 in)
England	177.8 cm (5 ft 10 in)	164.5 cm (5 ft 5 in)
Finland	178.9 cm (5 ft 10.5 in)	165.3 cm (5 ft 5 in)
Germany	178 cm (5 ft 10 in)	165 cm (5 ft 5 in)
Hungary	176 cm (5 ft 9.5 in)	164 cm (5 ft 4.5 in)
Japan	172.5 cm (5 ft 7.5 in)	158 cm (5 ft 2 in)
North Korea	165.6 cm (5 ft 5 in)	154.9 cm (5 ft 1 in)
South Korea	170.8 cm (5 ft 7 in)	157.4 cm (5 ft 2 in)
Montenegro	183.2 cm (6 ft 0 in)	168.4 cm (5 ft 6.5 in)
Norway	182.4 cm (6 ft 0 in)	168 cm (5 ft 6 in)
Peru	164 cm (5ft 4.5 in)	151 cm (4 ft 11.5 in)
Sri Lanka	163.6 cm (5 ft 4.5 in)	151.4 cm (4 ft 11.5 in)
Switzerland	175.4 cm (5 ft 9 in)	164 cm (5 ft 4.5 in)
Turkey	174 cm (5 ft 8.5 in)	158.9 cm (5 ft 2.5 in)
U.S.	176.1 cm (5 ft 9.5 in)	162.1 cm (5 ft 4 in)
Vietnam	165.7 cm (5 ft 5 in)	155.2 cm (5 ft 1 in)

Histograms

A histogram depicts a distribution of numerical data. To construct a histogram of several one-dimensional data points, we divide the data into "bins" based on what range they lie in, and then we plot a bar whose height corresponds to the number of points in the bin. Histograms can give a clear view of the way data is distributed, but sometimes changing the size and position of the bins can change the picture substantially. You can create a histogram in matplotlib with the command `plt.hist()`.

When plotting histograms, it can sometimes be useful to include reference lines. A nice way to do this without cluttering the figure is to make the reference lines white. You can do this in matplotlib with the command `plt.grid(True, color='w',linestyle='-')`.

```
from numpy.random import randn
from matplotlib import pyplot as plt
```

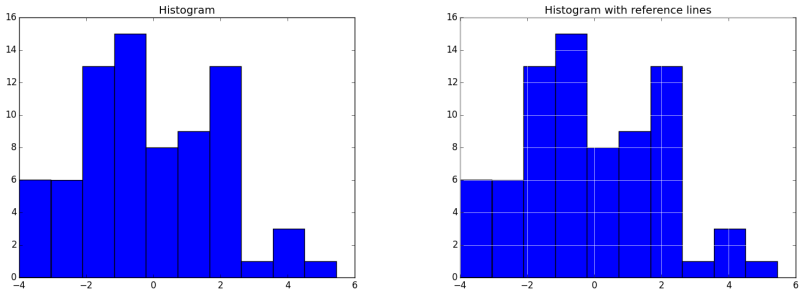

Figure 14.2: Histogram and Histogram with Reference Lines

```
data = randn(75) + randn(75) + randn(75) + randn(75)
plt.hist(data)
plt.grid(True, color = 'w', linestyle = '-')

plt.title("histogram")
plt.show()
```

Problem 2. Using 1000 random numbers from a normal distribution,

```
from numpy.random import normal
normal_numbers = normal(size=1000)
```

plot a histogram with white reference lines. Do this for 20 bins, 10 bins, 5 bins, and 3 bins.

Line Plots

Line plots should not be used for data that is one-dimensional or is two-dimensional with no natural ordering or progression because it gives the false impression that there is an order or progression when none exists (see Figure 14.4).

Recall that you can create a line plot in matplotlib with the command `plt.plot()`.

When plotting line graphs in matplotlib, the default line width is usually too thin. It will help your reader (or you) if you set the line width to 2 or more with the argument `lw=2` (See Figure 14.3).

Problem 3. Using 1000 data points, plot the equation

$$y = x^2 \sin(x),$$
$$x \in [0, 100].$$

It should look like Figure 14.5.

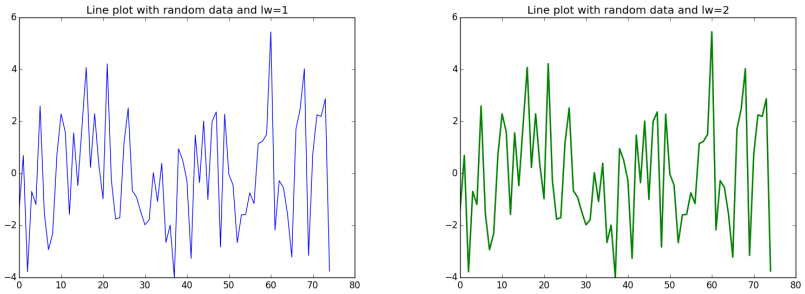

Figure 14.3: Line Plot. The left plot is plotted with `lw=1` whereas the right plot is plotted with `lw=2`. Notice that the right plot is better because it is easier to see the line.

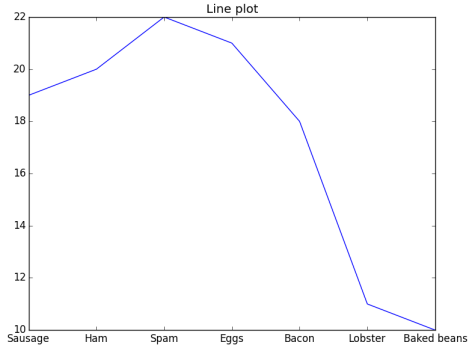

Figure 14.4: Line plot. Notice that this graph appears to show that the x-values are ordered. However, in reality, they are unordered. This is a bad practice and misrepresents the data.

Scatter Plots

Scatter plots graph (x, y) tuples and should be used instead of line plots when the data is two-dimensional but has no natural order.

You can create a scatter plot in matplotlib with the command `plt.scatter()`.

Figure 14.6 displays two scatter plots, the first appearing to have no strong correlation and the second a strong correlation. However, the same data is being plotted and the only difference is the scale and window size. Manipulating these can change your interpretation and should be done with careful consideration.

Problem 4. Plot the average heights of men (y) against women (x) using the data from Problem 1. Zoom in and out to see what difference that makes.

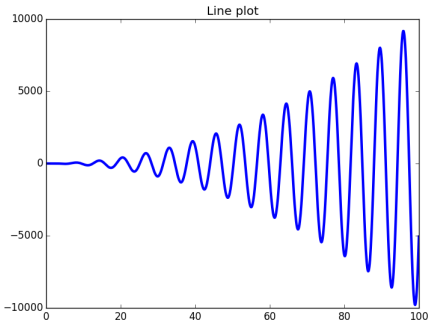

Figure 14.5: Solution to Problem 3

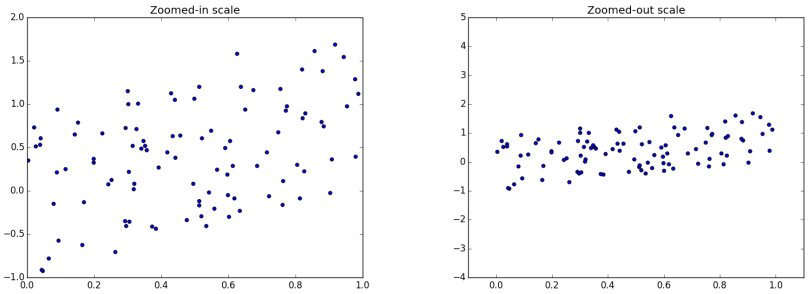

Figure 14.6: Scatter Plots. The data in both plots are the same, but note how in the first we see little correlation compared to the second. This is solely an artifact of the scale of the plot. Keep this in mind when you make scatter plots.

Contour Plots and Heat Maps

Three-dimensional data can be displayed on a two-dimensional page with a *contour plot*. This draws lines in the plane where the third value is constant—like a topographical map.

You can create a contour plot in matplotlib with `plt.contour()`.

Filling in the contours with successive colors gives a *heat map*. Note that the default color map for matplotlib is the *rainbow color map*. This is a poor choice for a heat map or colored contour map because people do not naturally interpret one color to be greater than another. To solve this problem in matplotlib, use the argument `cmap='NAME_OF_COLORMAP'` where `NAME_OF_COLORMAP` is one of matplotlib's sequential or diverging colormaps like `afmhot`.

```
# This code corresponds to the figure on the top right of the contour maps.
import numpy as np
from matplotlib import pyplot as plt

n=400
xran = np.linspace(-1.5,1.5,n)
yran = np.linspace(-1.5,1.5,n)
X, Y = np.meshgrid(xran,yran)
F = Y**2 - X**3 + X**2
plt.contourf(X, Y, F, [-2,-1,0.0001,1,2,3,4,5] ,cmap=plt.get_cmap('afmhot'))
```

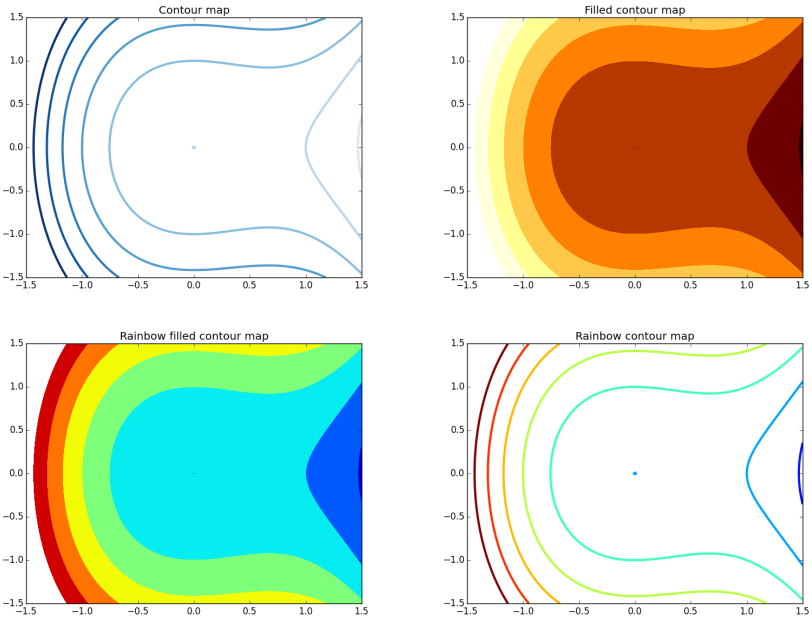

Figure 14.7: Contour Maps. The top left is a good contour map and the top right is a good heat map, and the bottom two are inferior because of bad color choices. The top right corresponds to the code below.

Two sequential colormaps are `afmhot` and `cool`. Two diverging colormaps are `seismic` and `bwr`. These colormaps along with their color scheme can be found on `http://matplotlib.org/examples/color/colormaps_reference.html`.

Problem 5. Using 400 intervals, plot a filled contour map of

$$z = \sin(x) + \sin(y),$$
$$x \in [0, 12\pi], y \in [0, 12\pi].$$

Be sure to use an appropriate color map.

You can also create a pseudocolor plot in matplotlib with the `plt.pcolormesh()` command.

Finally, three-dimensional data can also be plotted as a surface in 3-space, but it is often difficult to find a view where none of the important features of the data are obscured. A heatmap can help avoid this problem and is often easier for the viewer to decode (see Problem 7).

Exploring Data with Visualizations

When you visualize data you may notice trends that are not apparent from the numbers. The following problem is a famous example of this phenomenon.

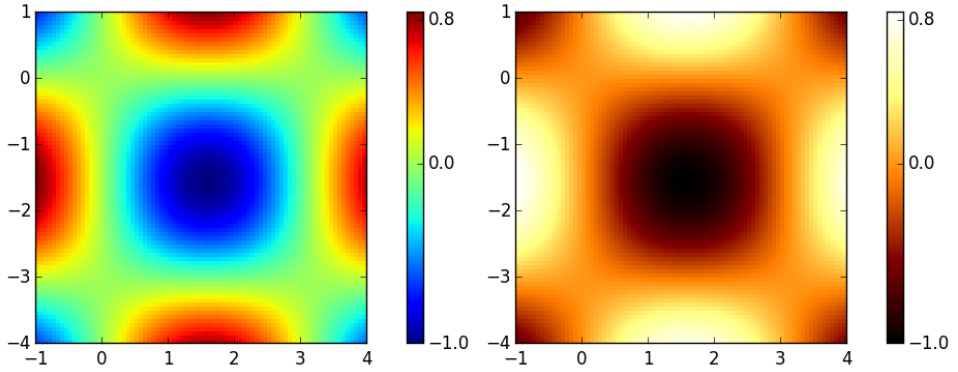

Figure 14.8: Both of these pseudocolor plots depict the function $z = \sin(x)\sin(y)$ on the domain $[-1, 4] \times [-4, 1]$. The plot on the left uses a rainbow gradient, which is pretty to look at, but whose choice of colors has no meaningful relationship to the data. The plot on the right uses the color map called `afmhot`, which has the important benefit that the lighter colors are naturally associated with higher values, while the darker colors are naturally associated with lower values.

Problem 6. The data sets I-IV in Table 14.1 are known as Anscombe's quartet. Each data set has identical statistical properties. In each case,

- The mean of x is 9 and the mean of y is 7.5.
- The variance of x is 11 and the variance of y is 4.127.
- The correlation between x and y is .816.
- The linear regression line is $y = 3 + 5x$.

Plot each data set. What do you notice?

I		II		III		IV	
x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

Table 14.1: These four sets of data are known as Anscombe's quartet.

As Problem 6 demonstrates, a picture can quickly reveal properties of the data that are difficult to see in a list of numbers.

The Iterative Process of Exploratory Visualization

Understanding a data set through visualization is often an iterative process. We start with an initial visualization—usually a broad overview. Examining the result leads to observations that provoke questions. We then filter or transform the data and adjust the visualization based on those questions and observations, and then repeat the process until we converge on a useful result.

Some of the questions that you should be asking in this iterative process include:

1. Is the data correct and reliable?
2. Are there any clear patterns, trends, or irregularities?
3. Would a different type of visualization give more information?
4. Is something unusual or interesting going on outside my view?
5. Would filtering the data to look at a particular subset give a more informative picture?
6. Would a transformation of the data give a more informative picture?

Let's consider each of these questions in more detail.

1. Data Integrity

Visualization is a powerful way to identify problems with your data set. Ask yourself, whether the results look like what you expect. Are they the right magnitude? Are there missing values?

2. Trends, Patterns, and Irregularities

Some patterns you may look for:

1. Are there trends in time (up, down, flat, cyclical)?
2. Does the data look random, cyclical, or patterned?
3. Are there clear correlations between the x- and y-values?
4. Is the trend linear or curved?
5. Do lines intersect, or are they parallel?
6. Is the data symmetric or skewed?
7. What is the variance (wide vs narrow)?
8. Are there clusters and gaps?
9. Is the data dense or sparse?

3. Using Different Types of Visualization

Data can be visualized using different types of visualization and is relatively easy to transform.

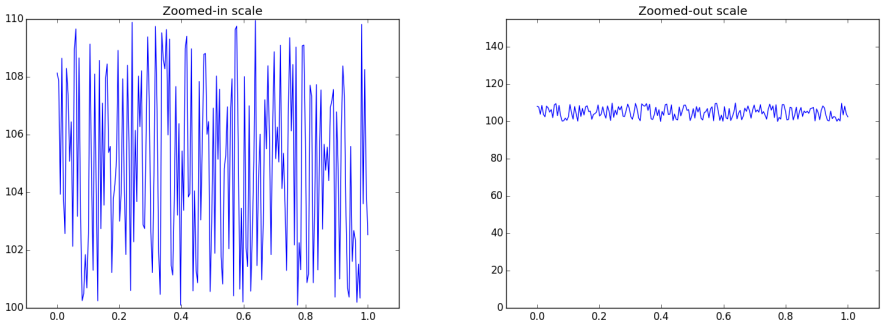

Figure 14.9: Scale issues. Notice that y-values on the left graph look random. However, it is clear to see on the right graph that the y-values aren't as sporadic.

Problem 7. This is an example of a surface that obscures your view:

```
x = np.linspace(-2*np.pi,2*np.pi,num=400)
y = np.linspace(-2*np.pi,2*np.pi,num=400)
X, Y = np.meshgrid(x,y)
Z = np.exp(np.cos(np.sqrt(X**2 + Y**2)))
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X,Y,Z)
plt.show()
```

Change this surface to a heatmap or a contour plot. Return a string of the benefits of each type of visualization. See the beginning of the lab for how to make a contour plot.

4. Changing the View

By choosing to plot only a portion of your data, you may be able to focus your attention on the most relevant data. On the other hand, you may also miss some interesting behavior in the data you didn't graph. Figures 14.6 and 14.9 are two such examples.

Another issue to consider is scale. If a number changes from 50,000 to 55,000 over some period, that increase can appear small or large, depending on where we start the y-axis and how long we stretch out the x-axis (see Figure **??**).

You can modify the scale of an existing plot using `plt.yscale()` or `plt.xscale()`.

Problem 8. Return to your plot of

$$y = x^2 \sin(x)$$
$$x \in [0, 100]$$

from Problem 3 and plot it a few more times adjusting the scale by setting the

y limits to be $\pm 10^k$ for $k = 0, 1, 2, 3, 4$ by using the command `plt.ylim(lower_limit,upper_limit)`.

5. Filtering the Data

Sometimes important information about a subset of the data can be obscured by considering the whole data set. Some information is difficult to notice when all the data is combined into one plot. But when the data is divided into subsets, more information can be gathered. For example, when the height of males and females are plotted for multiple countries, there may appear to be a large variance. But when males' heights and females' heights are isolated, more information about the variance can be deduced.

6. Transforming the Data

Often a transformation of the data can reveal useful information that was previously hidden.

For example, data that grows very rapidly or that is tightly clustered may be better visualized by plotting the logarithm of the data, instead of the original data, A log-lin plot (also called a log plot) is constructed by by taking the logarithm of the y-values of your data set.

This is easily done in matplotlib with the command `plt.semilogy(x,y)`. The syntax is the same as `plt.plot()`.

On the left side of Figure 14.10, $x^{14}+1$ and $x^{13}+1$ look almost identical. Graphic them on a log-lin plot (as on the right side of Figure 14.10) makes their differences apparent. Beware, however, that you should not label the transformed axis with transformed labels, because the reader ultimately needs the untransformed values. So a log-lin plot could have y-axis with $10^1, 10^2, 10^3, 10^4$ etc. all equidistant. And not with values $1, 2, 3, 4$ equidistant.

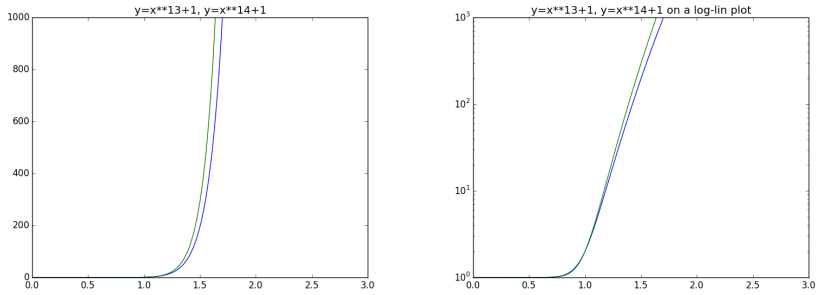

Figure 14.10: Plotting exponential graphs on a log-lin plot

Similarly, if the data grows very slowly, you may find that a lin-log plot (given by by taking the logarithm of the x-values) gives a more useful view. This is easily done in matplotlib with the command `plt.semilogx(x,y)`.

Transformations can also be useful for histograms. As an example, let us look at the cost of health care claims. Our data set is fabricated, but designed to closely resemble real-life data. An initial plotting of the data produces the histogram on the left in Figure 14.11. From this picture, it is hard to see the nature of the data—the plot almost looks like a single bar. We may think that all health care claims are cheap. However, when

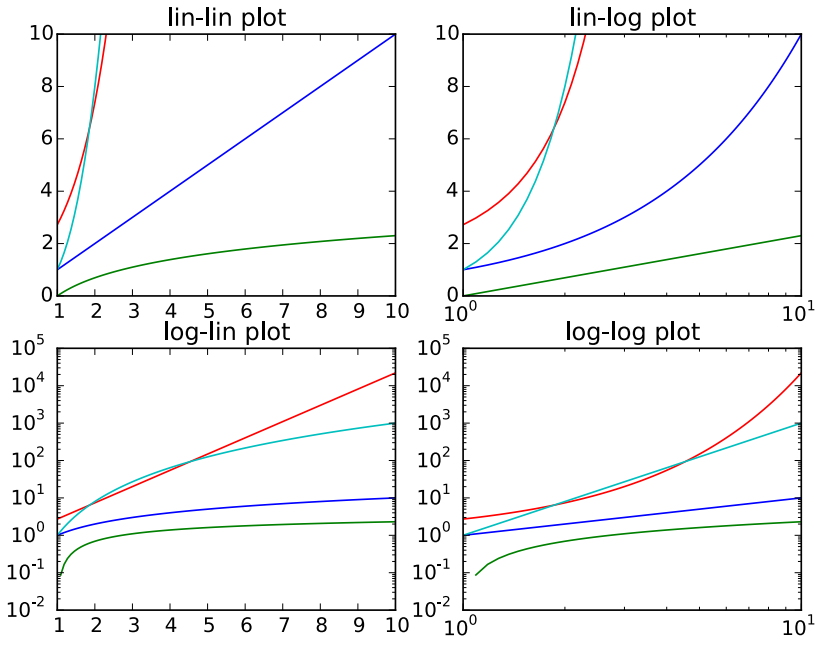

Figure 14.11: Here are some functions plotted on various logarithmic and linear scales. The dark blue line is $y = x$, the green line is $y = \log(x)$, the red line is $y = e^x$, and the light blue line is $y = x^3$. We have used different ranges on the y-axes to highlight appropriate parts of the graphs.

we take the logarithm of the claims prices, we get the histogram on the right in Figure 14.11. Graphed on this scale, the data has a bell-shaped distribution similar to a normal distribution. Moreover, we see that there are some very expensive claims being submitted, though they are few.

As a general rule, taking the log of the y-values is most useful when the range of the y-values is orders of magnitude larger than the range of the x-values. Our health care data was an example of this. Similarly, taking the log of the x-values is most useful when the range of the x-values is much bigger than the range of the y-values. In any case, you can try applying a log scale to one or both of your axes as a way to explore your data.

Be warned that many different functions can look almost linear on log plots. Thus, if you graph your data on a log-log plot and you see a line, you cannot conclude that your data must follow a polynomial. In general, you need more information.

Communicating Data with Visualizations

As the saying has it, "a picture is worth a thousand words." Certainly, a data visualization is far more valuable than a thousand data points to a colleague who wants to understand the results of your analysis. Data visualizations are critical for communication in both business and research.

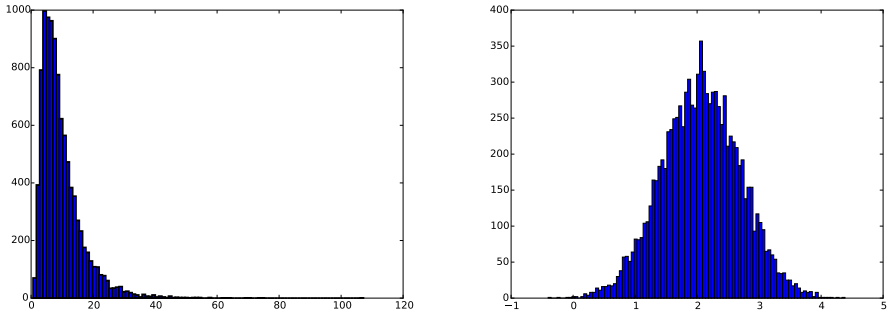

Figure 14.12: Two histograms of the same data are plotted above. The plot at left is a lin-lin scale, whereas the plot at right is a log-lin scale (log of the y-values). In this case, changing scales revealed important information about the data.

Choosing the Right Visualization

At this point, you have already used visualization to explore your data and draw conclusions, and you are ready to tell your results to someone else. The first step in creating a graphic to do this is to choose the right type of visualization. As we saw in the section "Types of Visualizations," there are many possibilities, each with different strengths. You should carefully consider which one best suits your data set and communication needs. Your goal should always be communicating as clearly, effectively, and accurately as possible.

Visualizations to Avoid: Pie Charts, Radar Charts, and Stacked Bar Charts

There are also several popular types of visualizations that you should generally **not** use. These may look pretty, but they tend to obscure information or cause confusion because of natural difficulties humans have in interpreting them.

For example, people have much more difficulty judging changes and variation in area than they do in length. As a result, charts that depend on distinguishing variation in area will be less effective and easily misunderstood. One of the most common charts that has this problem is the pie chart. Figure 14.13 indicates the difficulty in judging the difference in area between the lobster and bean slices.

As bad as the pie chart may be, there is a way to make things worse—make the chart three-dimensional. It seems to be very popular for people to convert a visualization that is naturally 2 or 1 dimensional into one that is artificially three dimensional, but this inevitably makes things more confusing. For example, we already have difficulty distinguishing the variation in area in a pie chart, and when we add a third dimension, it becomes even harder to distinguish variation in volume.

Other visualization types that tend to be confusing and ineffective include the stacked bar and the radar graph. The radar graph is just generally confusing. The main problem with stacking bars is that the second bar is not based at zero, so it becomes difficult to compare the different parts, as in Figure 14.14

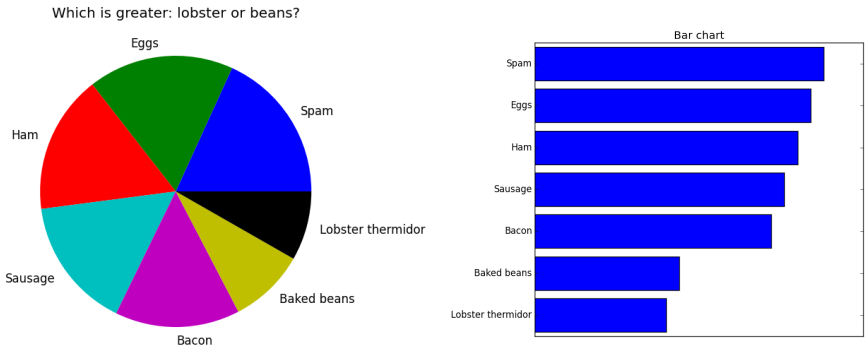

Figure 14.13: A pie chart on the left that has been improved by a transformation into a bar chart on the right.

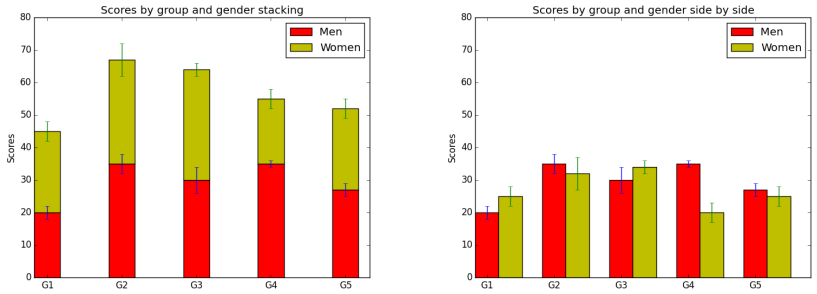

Figure 14.14: Stacked Bar Chart. Note how much more difficult it is to compare the relative values when they are stacked (on the left) instead of adjacent (on the right).

Simplify

A sentence should contain no unnecessary words, a paragraph no unnecessary sentences for the same reason that a drawing should contain no unnecessary lines and a machine no unnecessary parts. —Strunk and White

Once you have chosen the visualization best suited to your data, you should design the details so that all elements contribute to the communication of data.

Data visualization guru, Edward Tufte, offers two principles for simplifying graphics: (1) erase ink that does not communicate data, and (2) erase ink that communicates data redundantly. ([**tufte2001**] pp.96-100). According to these principles, decorative backgrounds, fancy lettering, and cute graphics in the corner of your plots should all be deleted.

Other opportunities for simplification are harder to notice and implement. As an example, let us examine the plot on the right of Figure 14.11. This plot was created with the following code.

```
import numpy as np
import scipy as sp
from matplotlib import pyplot as plt
```

```
m = 2.07
s = 0.63
num_samples = 10000
samples = []

for i in range(num_samples):
    samples.append(sp.random.normal(m, s))

sp_samples = sp.array(samples)

# Plot the histogram
plt.hist(sp_samples, 100)
```

For purposes of this example, the only important part of the above code is the line `plt.hist(sp_samples, 100)` that plots the histogram. What ink in this plot can we erase because it communicates no data?

First, let's get rid of the vertical black lines that separate the bars of the histograph and the black outline. These are meaningless for our application and only distract. We can do this by modifying our call to `plt.hist()` as follows.

```
plt.hist(sp_samples, 100, edgecolor='none')
```

Next, let us turn off the top and right lines that box in the graph. To do this, we need to access the "axis" object associated with the figure. We can access the "axis" object with the command `plt.gca()` (get current axis).

```
# Get current axis instance
axis = plt.gca()

# Hide top and right spines
axis.spines['right'].set_visible(False)
axis.spines['top'].set_visible(False)
```

These commands only turn off the sides of the box, not the tick marks. To turn off the tick marks we run the following commands.

```
# Only show bottom and left tick marks
axis.yaxis.set_ticks_position('left')
axis.xaxis.set_ticks_position('bottom')
```

Finally, we do not need so many tick marks on the x- and y-axes. We adjust those along with specifying the range on each axis.

```
# Fix x- and y-ranges
plt.xlim(-1,5)
plt.ylim(0, 350)

# Use fewer axis ticks
plt.xticks(np.arange(0, 5, 2))
plt.yticks(np.arange(0, 351, 100))
```

The final graph is shown in Figure 14.15. Note how much cleaner this looks than the original.

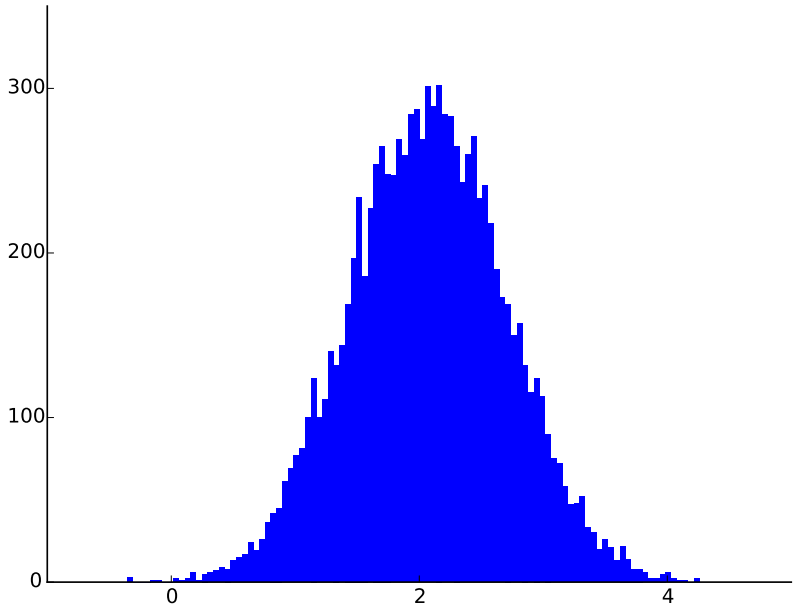

Figure 14.15: A simplified version of the histogram in Figure 14.11.

Problem 9. Choose a graph you previously created that you think could use simplification. Apply at least 3 of the following to simplify your plot:

1. Turn off some of the spines.
2. Fix the x- or y-range to better fit your data.
3. Change the number of x- or y-ticks.
4. Choose a better color scheme.

Simplify your image in any other ways you can think of.

Small Multiples

Often we have too much information to display cleanly in one chart. One indicator of this is if the reader must often look back and forth between a legend and the chart to decode the images. A very useful way to deal with this problem is the method of *small multiples*, an idea made famous by Edward Tufte.

The idea is to separate the data into many small graphics, placed near each other in a way that allows us to easily compare them.

For example, one might wish to compare the first six Chebyshev polynomials. It would be natural to plot them together on the same graph, as in Figure 14.16. But this graph is too cluttered and one must constantly refer back to the legend to identify which colored

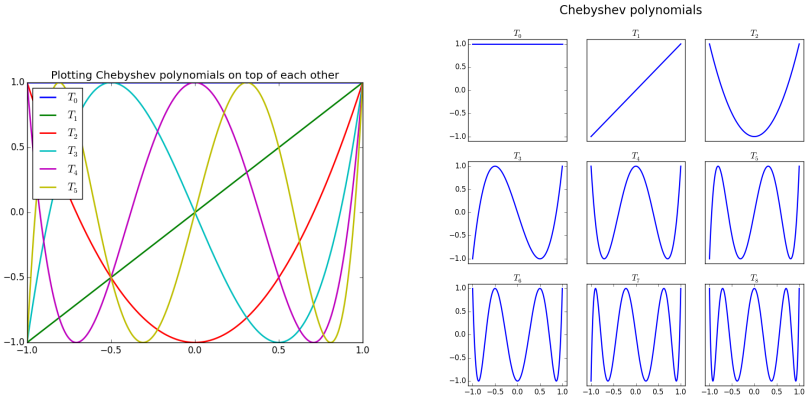

Figure 14.16: Chebyshev polynomials together on the left and separately on the right. Notice that the legend is no longer needed for the right graph and it is easier to compare between Chebyshev polynomials because the graph isn't cluttered.

line is which.

The small multiple approach is to plot each polynomial on its own set of axes, adjacent to each other and aligned in such a way that you can easily compare the different plots as in Figure14.16

Note how even with 9 plots instead of 6, it is much easier to identify the different polynomials and to compare them.

The main tool you need to make small multiples is the command `ax = plt.subplot()`. This allows you to set up multiple plots, and access them one at a time.

```
import numpy as np
from matplotlib import pyplot as plt
from numpy.polynomial import Chebyshev as T

def Chebyshev_subplots():
    fig = plt.figure(dpi=100)
    fig.set_size_inches(10,10)
    fig.suptitle('Chebyshev Polynomials', fontsize=20)

    x = np.linspace(-1,1,500)

    for i in range(9):
        ax = plt.subplot(3,3,i+1)
        ax.plot(x, T.basis(i)(x), lw=2, label="$T_%d$"%i)
        plt.xlim(-1.1,1.1)
        plt.ylim(-1.1,1.1)
        ax.set_title('$T_%d$'%i)
        if i%3:
            # remove the inner y-ticks and labels.
            ax.set_yticklabels([])
            ax.yaxis.set_ticks_position('none')
        if i<6:
            # remove the inner x-ticks and labels.
            ax.xaxis.set_ticks_position('none')
            ax.set_xticklabels([])

    plt.show()
```

Problem 10. The $n+1$ Bernstein basis polynomials of degree n are defined as

$$b_{v,n} = \binom{n}{v} x^v (1-x)^{n-v}, v = 0, 1, ..., n.$$

Plot the Bernstein basis polynomials for v, n in [0,3] as small multiples and compare that to the cluttered version of plotting them on top of each other.