CHAPTER 16

Creating Scientific Diagrams

LEARNING GOAL: You can use `matplotlib` to generate high-resolution plots.

16.1 IN THIS CHAPTER YOU WILL LEARN

- How to create a bar plot
- How to create a scatterplot
- How to create a line plot
- How to create a pie chart
- How to draw error bars
- How to plot a histogram

16.2 STORY: NUCLEOTIDE FREQUENCIES IN THE RIBOSOME

In 2009, the Nobel Prize in Chemistry went to Venkatraman Ramakrishnan, Thomas Steitz, and Ada Yonath for their studies on the structure and function of the ribosome. The ribosome is among the biggest known molecular machines, made up of three RNA components (in prokaryotes the 23S, 16S, and 5S rRNA) and many proteins. The RNA consists of the four basic ribonucleotides and a few modified nucleotides that fine-tune the ribosomal function.

TABLE 16.1 Nucleotide Numbers in the 23S Ribosomal Subunit.

Species	A	C	G	U
T. thermophilus	606	759	1024	398
E. coli	762	639	912	591

16.2.1 Problem Description

In Table 16.1, you find the exact number of nucleotides for the 23S subunit of the *Thermus thermophilus* ribosome resolved by Ramakrishnan et al. and the corresponding data for *Escherichia coli*. In this chapter, you will learn how to display these data in a more attractive way.

The following program creates a bar plot using the `matplotlib` library. First, the *y*-values and bar labels are put into list variables. Second, the `figure()` function creates an empty diagram. Third, various `matplotlib` functions like `bar()` draw components of the diagram. Finally, `savefig()` saves the diagram to a `.png` file.

16.2.2 Example Python Session

```
from pylab import figure, title, xlabel, ylabel,\
    xticks, bar, legend, axis, savefig
nucleotides = ["A", "G", "C", "U"]
counts = [
    [606, 1024, 759, 398],
    [762, 912, 639, 591],
    ]
figure()
title('RNA nucleotides in the ribosome')
xlabel('RNA')
ylabel('base count')

x1 = [2.0, 4.0, 6.0, 8.0]
x2 = [x - 0.5 for x in x1]

xticks(x1, nucleotides)

bar(x1, counts[1], width=0.5, color="#cccccc", label="E.coli 23S")
bar(x2, counts[0], width=0.5, color="#808080", label="T. \
    thermophilus 23S")

legend()
axis([1.0, 9.0, 0, 1200])
savefig('barplot.png')
```

Source: Adapted from code published by A.Via/K.Rother under the Python License.

16.3 WHAT DO THE COMMANDS MEAN?

16.3.1 The `matplotlib` Library

The program uses `matplotlib`, a Python library for scientific diagrams. It contains many functions to generate drawings from data, and some of them have a lot of options. Instead of explaining every option exhaustively, we focus on creating standard diagrams from straightforward data such as *x*- and *y*-values. This chapter provides you with ready-made scripts for different kinds of plots that you can then customize.

Q & A: HOW CAN I EXECUTE THE EXAMPLE?

To use `matplotlib`, you need to install it separately from Python. On all systems this can be done by the single terminal command

```
easy_install matplotlib
```

This requires installation of the `easy _ install` program.

On Ubuntu Linux, you also can use

```
sudo apt-get install python-matplotlib
```

For Mac OS X 10.6 or higher, `.dmg` files are provided. On Windows, you need to download and install Scientific Python (http://scipy.org/) first.

The program in Section 16.2.2 creates a file `barplot.png` (see Figure 16.1) in the same directory in which the Python script has been executed.

You can use `matplotlib` to create a diagram in just four steps:

```
from pylab import figure, plot, savefig
xdata = [1, 2, 3, 4]
ydata = [1.25, 2.5, 5.0, 10.0]
figure()
plot(xdata, ydata)
savefig('figure1.png')
```

Source: Adapted from code published by A.Via/K.Rother under the Python License.

First, the library is imported. Second, you start a new figure. Third, you plot some data. Fourth, you save the figure to a `.png` file. This gives you a plot that you can use right away to analyze your data. Below, more options are presented that you can use to create high-quality plots.

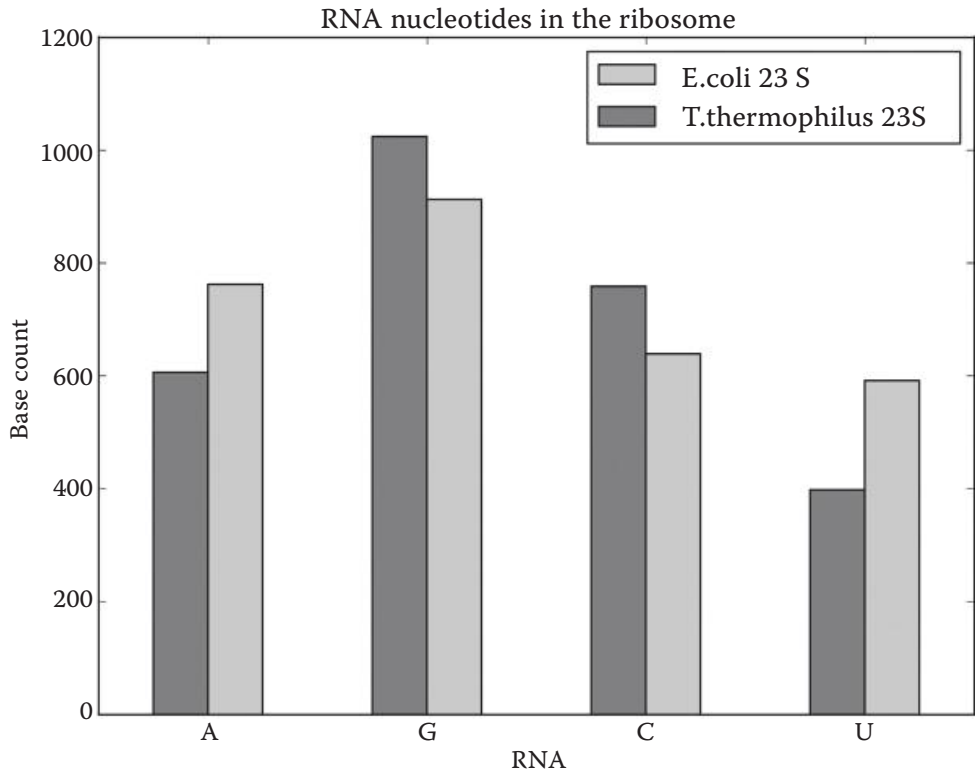

FIGURE 16.1 Bar plot.

Q & A: Can I Create Multiple Figures in the Same Program?

Each time you call the `figure()` function, the background is cleared, and you start creating a new plot. While `matplotlib` is capable of creating multi-panel figures, there are several good reasons to create separate images. First, you can use separate figures for different purposes (posters, presentations, publications) easier than you can use a single multi-panel image. Second, you might want to customize or annotate parts of the figure in a separate program. Finally, single figures are easier to analyze. Creating figures with multiple panels should be left for the stage of manuscript preparation when all your results have been analyzed and confirmed.

16.3.2 Drawing Vertical Bars

The `bar()` function simply draws bars. In the simplest version, it takes as parameters two lists: one for the *x*-axis values and one for the bar heights:

```
bar([1, 2, 3], [20, 50, 100])
```

The first list `[1, 2, 3]` indicates where the bars should start on the *x*-axis; the second list `[20, 50, 100]` indicates how high each bar should be.

Similarly, you can create horizontal bars with the `barh()` function:

```
barh([1, 2, 3], [20, 50, 100])
```

The example program in Section 16.2.2 draws four groups with two bars each (see Figure 16.1). If you want to nicely align the bars horizontally with their labels, you need to carefully prepare the *x*-values. There are two lists of *x*-positions in the program: `x1` is used for the label of the right bar of each group, and `x2` is used for the label of the left bar of each group (shifted by 0.5 to the left with respect to `x1`). The final `bar()` command has three additional parameters for the bar widths, color, and label that is to appear in the legend box:

```
bar(x1, counts[1], width = 0.5, color = "#cccccc", \
    label = "E.coli 23S")
```

16.3.3 Adding Labels to an *x*-Axis and *y*-Axis

Every scientific diagram that has an *x*-axis and *y*-axis needs to have a concise description on each axis, including the unit used. The `xlabel()` and `ylabel()` functions draw a text on the respective axis:

```
xlabel('protein concentration [mM]')
```

It is possible to use mathematical symbols, subscripts, and superscripts in the labels:

```
xlabel('protein concentration [$\muM$]')
```

A full list of symbols can be found at http://en.wikipedia.org/wiki/Help:Formula.

16.3.4 Adding Tick Marks

Equally important as labeling the axes is adding numerical or text marks along the axis. `matplotlib` adds numbers by itself, but sometimes the result is not what you want. The `xticks()` and `yticks()` functions draw customized ticks:

```
xticks(xpos, bases)
```

writes each element of the `bases` list of strings at the positions `xpos` on the *x*-axis. The `yticks` function works similarly.

16.3.5 Adding a Legend Box

The `legend()` function takes the labels of all data sets that were plotted so far and writes them to a legend box in the order of appearance. It does not need any argument:

```
legend()
```

16.3.6 Adding a Figure Title

The `title()` function simply adds text on the top of the diagram, like in the example program in Section 16.2.2:

```
title('RNA bases in the ribosome')
```

16.3.7 Setting the Boundaries of the Diagram

`matplotlib` automatically chooses the extent of the diagram in such a way that all data will be visible. Sometimes, though, this is not what you want. For instance, if you have a large set of data and want to zoom in on the lower 10%, you can use the `axis()` function:

```
axis([lower_x, upper_x, lower_y, upper_y])
```

`axis()` takes a list of four values: the lower and upper boundaries for both the *x*-axis and the *y*-axis. In the bar plot program, `axis()` is used to adjust the margins on the left, right, and top of the canvas to balance the diagram optically:

```
axis([1.0, 9.0, 0, 1200])
```

16.3.8 Exporting an Image File in Low Resolution and High Resolution

The `savefig()` function writes the entire diagram to an image file in `.png` format:

```
savefig('barplot.png')
```

By default, a 600 × 600 pixel image with 100 dpi will be created. You can create higher resolution images by adding a precise dpi value:

```
savefig('barplot.png', dpi = 300)
```

You can also export to `.tif` and `.eps` formats directly:

```
savefig('barplot.tif', dpi = 300)
```

16.4 EXAMPLES

Example 16.1 How to Plot a Function

`matplotlib` can create line plots from *x*/*y* data. The `plot()` function requires two lists of values that have the same length. The following example plots a sine function (see Figure 16.2):

```
from pylab import figure, plot, text, axis, savefig
import math

figure()

xdata = [0.1 * i for i in range(100)]
ydata = [math.sin(j) for j in xdata]

plot(xdata, ydata, 'kd', linewidth=1)
text(4.8, 0, "$y = sin(x)$",\
    horizontalalignment='center', fontsize=20)
axis([0, 3 * math.pi, -1.2, 1.2])

savefig('sinfunc.png')
```

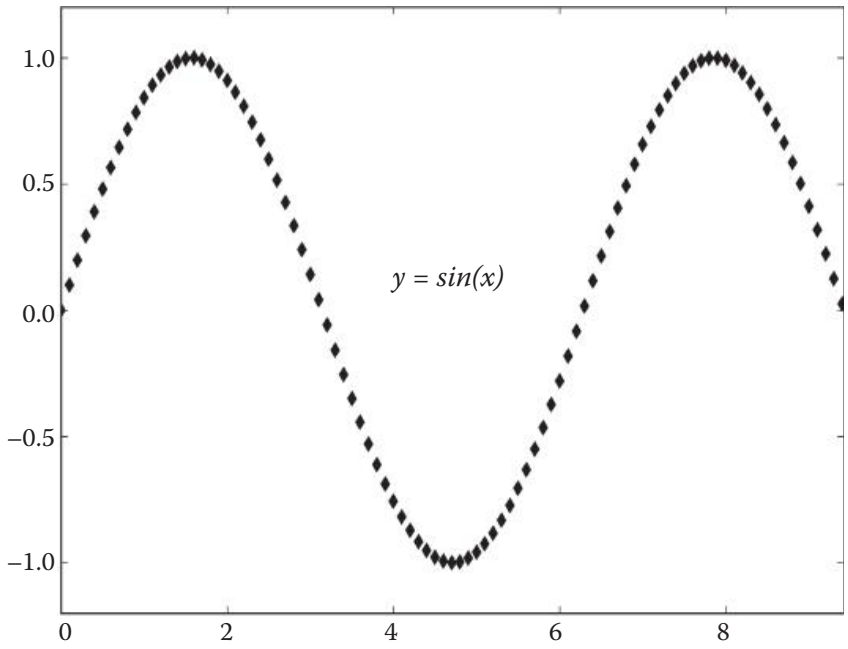

FIGURE 16.2 Plot of a sine function.

First, the program creates a list of equidistant *x*-values using the `range()` function and a list comprehension (explained in Chapter 4). The *y*-values are created by a list comprehension that calculates `sin(x)` for each *x*-value. The third parameter in the plot function, `'kd'`, indicates the color and style of the line. The first character stands for the color (k: black, r: red, g: green, b: blue; others exist). The second character indicates the symbol used for drawing (`o`: circles, `s`: squares, `v` and `^`: triangles, `d`: diamonds, `+`: crosses, `-`: lines, `:`: dotted lines, `.`: dots). The `text()` function adds a text marked by the enclosing $ symbols at a given *x*/*y* position using the mathematical TeX notation (see http://en.wikipedia.org/wiki/Help:Formula). When you limit the *x*-right boundary to `3 * math.pi` in the `axis()` function, the `y = sin(x)` function begins and ends at the same height (for purely aesthetic reasons).

Q & A: When Using the `plot()` Function, I See Two Symbols in the Legend Box. How Can I Get Just One?

For a line plot or scatterplot, if you add the `legend()` function to the code, you will see the symbol used for drawing (a diamond in the example) twice in the legend box. It is possible to fix this within `matplotlib`, but it is complicated. We recommend leaving the legend out or as it is and edit the legend box manually in a graphics program when all your figures are finished.

Example 16.2 Drawing a Pie Chart

A pie chart like the one in Figure 16.3 can be drawn with the `pie()` function:

```
from pylab import figure, title, pie, savefig
nucleotides = 'G', 'C', 'A', 'U'
count = [1024, 759, 606, 398]
explode = [0.0, 0.0, 0.05, 0.05]

colors = ["#f0f0f0", "#dddddd", "#bbbbbb", "#999999"]
```

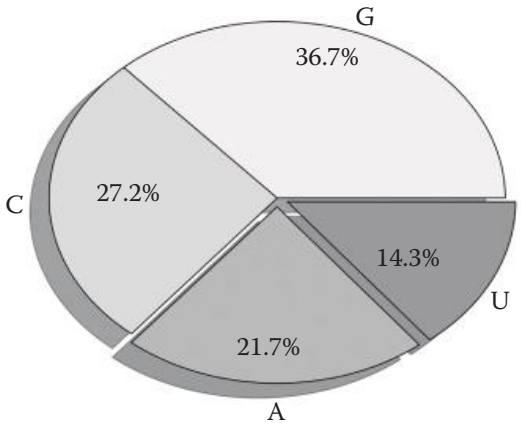

FIGURE 16.3 Pie chart: Bases in 23S RNA from *T. thermophilus*.

```
def get_percent(value):
 '''Formats float values in pie slices to percent.'''
   return "%4.1f%%" % (value)

figure(1)
title('nucleotides in 23S RNA from T.thermophilus')
pie(count, explode=explode, labels=nucleotides, \
    shadow=True, colors=colors, autopct=get_percent)
savefig('piechart.png', dpi=150)
```

Source: Adapted from code published by A.Via/K.Rother under the Python License.

If you want to emphasize that the amount of G and C in the *T. thermophilus* ribosome is more than half of the overall nucleotides, you can use a pie chart. The `pie()` function in `matplotlib` works in a similar way as `bar()` or `plot()`: you supply the numbers, labels, and colors as lists of items. By the values in the `explode` list, you can move pie slices out from the center (a 0.0 means it attaches to the other slices). In addition to the labels, you can also write text into each slice. The `autopct` parameter is a function (the function name is used like a variable here, this is why no parentheses are needed) that gets the size fraction of a slice (from 0.0 to 1.0) and returns a string. The `get_percent` function converts the number to a string and adds a percent symbol (see Chapter 3 for string formatting).

Q & A: Why Are There Four Percent Symbols in the `get_ percent` Function?

Each of the four percent symbols in `get_percent()` is necessary, but they all have a different meaning:

```
return "%4.1f%%" % (value)
```

In string formatting, a single percent character is interpreted as the start of a formatting character (`%s` for a string, `%i` for an integer, `%f` for a float). The first symbol `%4.1f` is the placeholder for a float inside the string. The double percent character (`%%`) is the placeholder for a normal percent symbol, because a single percent would be interpreted as another formatting character. Finally, the fourth percent symbol connects the formatting string to the tuple with values to be inserted.

Example 16.3 Adding Error Bars

Error bars can be added to both scatterplots and bar plots (see Figure 16.4). In both cases, you need a third list of numbers for the size of the error bars. A scatterplot with error bars is created by the `errorbar()` function (which works very similar to `plot()`), whereas for a bar plot the parameters `yerr` and `ecolor` can be added to the `bar()` function.

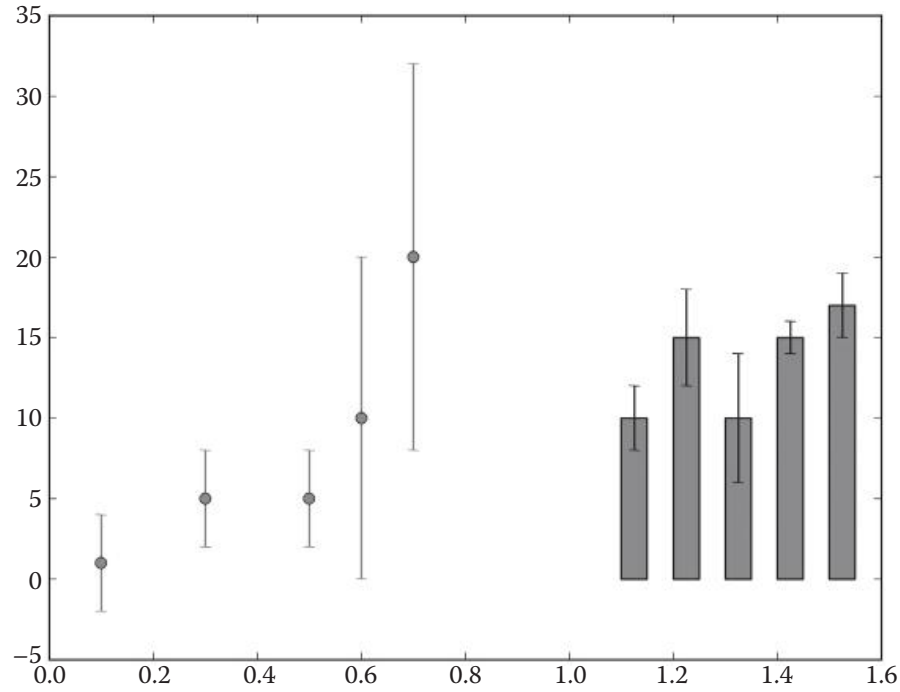

FIGURE 16.4 Error bars in scatterplots and bar plots.

```
from pylab import *
figure()
from pylab import figure, errorbar, bar, savefig
figure()

# scatterplot with error bars
x1 = [0.1, 0.3, 0.5, 0.6, 0.7]
y1 = [1, 5, 5, 10, 20]
err1 = [3, 3, 3, 10, 12]
errorbar(x1, y1, err1 , fmt='ro')

# barplot with error bars
x2 = [1.1, 1.2, 1.3, 1.4, 1.5]
y2 = [10, 15, 10, 15, 17]
err2 = (2, 3, 4, 1, 2)
width = 0.05
bar(x2, y2, width, color='r', yerr=err2, ecolor="black")

savefig('errorbars.png')
```

Source: Adapted from code published by A.Via/K.Rother under the Python License.

Example 16.4 Drawing a Histogram

The following code takes a list of integers and plots their absolute frequencies in five bins (see Figure 16.5):

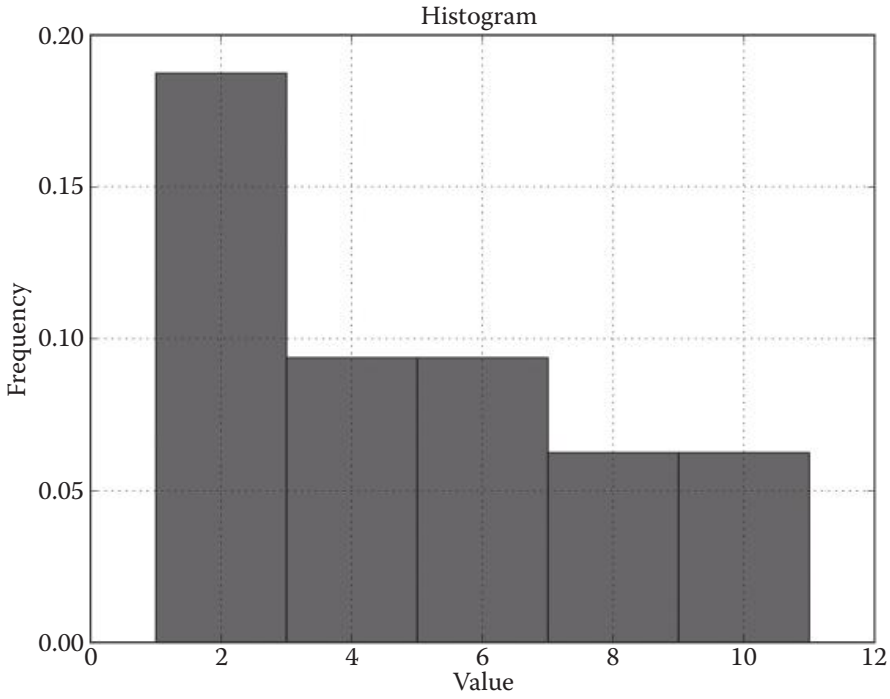

FIGURE 16.5 Histogram.

```
from pylab import figure, title, xlabel, ylabel, \
    hist, axis, grid, savefig
data = [1, 1, 9, 1, 3, 5, 8, 2, 1, 5, 11, 8, 3, 4, 2, 5]
n_bins = 5

figure()
num, bins, patches = hist(data, n_bins, normed=1.0, \
    histtype='bar', facecolor='green', alpha=0.75)
title('Histogram')
xlabel('value')
ylabel('frequency')
axis()
grid(True)
savefig('histogram.png')
```

Source: Adapted from code published by A.Via/K.Rother under the Python License.

The `hist()` function creates a bar plot but groups the data into the given number of bins first. The `grid()` function switches on a grid in the background of the diagram.

Q & A: WHERE CAN I FIND HELP TO DRAW FURTHER DIAGRAM TYPES?

The quickest method to find your way around `matplotlib` is to borrow from working examples. You can check the `matplotlib` gallery web page (http://matplotlib.org/gallery.html) for an example diagram that is the closest to what you want to do. Then copy the source code for the example and start manipulating it (see Exercise 16.4).

16.5 TESTING YOURSELF

Exercise 16.1 Create a Bar Plot

Display the neuron lengths from Chapter 3 as a plot with horizontal bars.

Exercise 16.2 Create a Scatterplot

Plot the number of bases in *T. thermophilus* versus the number of bases in *E. coli* from Table 16.1.

Exercise 16.3 Draw a Histogram

Read a FASTA file with multiple sequences. Plot a histogram of the sequence lengths.

Exercise 16.4 Use the `matplotlib` Gallery

Find out how to create a box-and-whisker plot in the `matplotlib` gallery. Copy and paste the example to a Python file and execute it.

Exercise 16.5 Create a Series of Plots

Write a program that parses a multiple sequence FASTA file, counts the frequencies of the 20 amino acids in each sequence, and creates a separate pie chart for each sequence, showing the top five frequencies and summarizing the rest as "other."