CHAPTER 13

Using External Modules: The Python Interface to R

LEARNING GOAL: You can use Python to do statistical analyses with R.

13.1 IN THIS CHAPTER YOU WILL LEARN

- How to run R commands from a Python script
- How to save R output into a Python variable
- How to generate an R object (e.g., a vector) from a Python object (e.g., a tuple)
- How to automatically generate R plots from Python

13.2 STORY: READING NUMBERS FROM A FILE AND CALCULATING THEIR MEAN VALUE USING R WITH PYTHON

13.2.1 Problem Description

Biologists have a constant need to statistically analyze their data and plot them. R (www.r-project.org/) is one of the most frequently used pieces of software for statistical computing and graphical analyses. In many situations, you may find it very useful to be able to call R from a Python script.

For example, if you have to calculate the mean value and the standard deviation of several distributions of numbers, each recorded in a different file, and then you want to automatically create one or more plots, you can delegate many tasks of your calculation to R and use Python to connect them. In this chapter, we assume that you already know how R works. If not, we strongly suggest that you familiarize yourself with the basics of R before reading this chapter.

Python has two modules, RPy and RPy2, to connect with R. RPy2 is a redesigned version of RPy. All examples in this chapter use RPy2, which we recommend to use. The module must be downloaded, installed, and imported into a script or a Python session (see Box 13.1 for RPy2 installation).

BOX 13.1 INSTALLING THE PYTHON INTERFACE TO R

Installing RPy or RPy2 may be the most difficult thing in this entire chapter. In fact, you have to choose a release of RPy or RPy2 that is consistent with the R and Python versions that are installed on your computer.

If `easy_install` is available on your computer, you can just type in a UNIX/Linux shell

```
sudo easy_install rpy2
```

`easy_install` is a Python module that lets you automatically download, build, install, and manage Python packages. To check if the package is available on your computer, go to the command line terminal and type

```
easy_install
```

If you get the warning (or a similar one)

```
error: No urls, filenames, or requirements specified (see— help)
```

instead of

```
easy_install: Command not found.
```

it means that you already have `easy_install`. Otherwise go to https://pypi.python.org/pypi/setuptools.

In the following session, simple R actions are performed, such as creating vectors, creating matrices, reading data from a file, and calculating the mean of a set of numbers. Once you get the philosophy behind the use of R via Python, you will find it easy to access any R function from Python.

13.2.2 Example Python Session

```
Python commands:
import rpy2.robjects as robjects
r = robjects.r
pi = r.pi
x = r.c(1, 2, 3, 4, 5, 6)
y = r.seq(1,10)
m = r.matrix(y, nrow = 5)
n = r.matrix(y, ncol = 5)
f = r("read.table('RandomDistribution.tsv', sep = '\t')")
f_matrix = r.matrix(f, ncol = 7)
mean_first_col = r.mean(f_matrix[0])
```

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

```
Equivalent R commands:
> p = pi
> x = c(1,2,3,4,5)
> y = seq(1,10)
> m = matrix(y, nrow = 5)
> n = matrix(y, ncol = 5)
> f = read.table('RandomDistribution.tsv',sep = '\t')
> f_matrix = matrix(f, ncol = 7)
> mean_first_col = mean(f[,1])
```

Figure 13.1 shows how the file `RandomDistribution.tsv` looks.

6071	103	0.0169659034755	40	0.00658870037885	276	0.0454620326141
6106	109	0.0178512938094	38	0.00622338683262	265	0.0433999344907
6148	93	0.015126870527	65	0.01057254391670	261	0.0424528301887
6119	114	0.018630495179	32	0.00522961268181	239	0.0390586697173
6118	87	0.0142203334423	47	0.00768224910101	287	0.0469107551487
6154	104	0.0168995775106	52	0.00844978875528	277	0.0450113747156
6154	118	0.019174520637	31	0.00503737406565	258	0.0419239519012
6143	94	0.0153019697216	23	0.00374409897444	281	0.0457431222530
6120	120	0.0196078431373	26	0.00424836601307	261	0.0426470588235
6142	108	0.0175838489092	45	0.00732660371215	290	0.0472158905894
6129	107	0.017457986621	36	0.00587371512482	262	0.0427475934084
6117	126	0.0205983325159	37	0.00604871669119	285	0.0465914664051
6171	138	0.0223626640739	40	0.00648193161562	255	0.0413223140496
6121	140	0.0228720797255	25	0.00408429995099	257	0.0419866034962
6090	107	0.0175697865353	39	0.00640394088670	270	0.0443349753695
6123	106	0.0173117752736	45	0.00734933855953	260	0.0424628450106
6139	141	0.0229679100831	53	0.00863332790357	225	0.0366509203453
6122	118	0.0192747468148	38	0.00620712185560	265	0.0432865076772
6084	99	0.0162721893491	33	0.00542406311637	260	0.0427350427350
6094	113	0.0185428290121	21	0.00344601247128	259	0.0425008204792
6139	102	0.0166150838899	27	0.00439811044144	289	0.0470760710213

FIGURE 13.1 Portion of the RandomDistribution.tsv file.

13.3 WHAT DO THE COMMANDS MEAN?

13.3.1 The `robjects` Object of `rpy2` and the `r` Instance

We assume that the module `rpy2.py` is installed on your computer (see Box 13.1) and that you already know how R works. The module to be imported to use the `rpy2` package is `robjects`:

```
import rpy2.robjects as robjects
```

The `r` object of the `rpy2.robjects` module (`robjects.r`) represents the “bridge” from Python to R. In the example, `robjects.r` has been assigned to the variable `r` to avoid writing `robjects.r` every time an R function is used:

```
r = robjects.r
```

13.3.2 Accessing an R Object from Python

At this point, you can start using R in Python. You can access R objects from Python in three ways: (1) accessing an R object as an attribute of the `r` object, using the dot syntax; (2) using the `[]` operator on `r` like you would use a dictionary; and (3) calling `r` like you would do with a function, passing the R objects as arguments. In all cases, the result is an R vector.

Accessing an R Object as an Attribute of the `r` Object, Using the Dot Syntax

In R you can access, for instance, the `pi` object (which in R is a vector of length 1, the value of which is 3.141593) as follows:

```
> pi
[1] 3.141593
```

In Python, you can get `pi` by typing

```
>>> import rpy2.robjects as robjects
>>> r = robjects.r
>>> r.pi
<FloatVector - Python:0x10c096950/R:0x7fd1da546e18>
[3.141593]
```

This makes perfect sense, with `r` being the Python interface to R: R objects are basically attributes of the `r` object, and you can access them by simply using the dot syntax. Notice that if you use the `print` statement, the result will look a bit different:

```
>>> print r.pi
[1] 3.141593
```

And since `r.pi` is a vector of length 1, if you want to get its numerical value, you have to use indexing:

```
>>> r.pi[0]
3.141592653589793
```

Accessing an R Object Using the `[]` *Operator on* `r` *Like You Would Use a Dictionary*

You can think of R object names and their values as `key:value` pairs of a dictionary and retrieve the value of `'pi'` as follows:

```
>>> pi = r['pi']
>>> pi
<FloatVector - Python:0x10f4343b0/R:0x7f8824e47f58>
[3.141593]
>>> pi[0]
3.141592653589793
```

Calling `r` *Like You Would Do with a Function, Passing the R Object as an Argument*

Another way to access the value of an R object is by calling the `r` object as you would do with a function, passing as argument the R object name:

```
>>> pi = r('pi')
>>> pi[0]
3.141592653589793
```

In summary, the `r` object works like an object with attributes through the dot syntax, like a dictionary, and like a function to achieve the same result. Notice that in all these cases, the result is a vector, the value(s) of which can be accessed using the `[]` operator as you do for Python lists or tuples.

Nearly everything in R is a vector or a matrix (a vector of vectors). Therefore, it is important to learn how to manipulate such objects, how to extract their elements, and how to convert R objects into Python objects and vice versa.

13.3.3 Creating Vectors

Similarly to the R `pi` object, R functions for vector building can be called as attributes of `robjects.r` (remember that `robjects.r` has been stored in the `r` variable) using the dot syntax:

```
>>> print r.c(1, 2, 3, 4, 5, 6)
[1] 1 2 3 4 5 6
```

Remember that R vectors can be generated using the `c()` function. As in the case of the `pi` object, you can use two additional ways to get R vectors from the `r` object: interpreting `r` as a dictionary or as a function.

To use the dictionary-like approach, you can translate the R function `c()` into a Python function using the `[]` operator:

```
>>> print r['c'](1, 2, 3, 4, 5, 6)
[1] 1 2 3 4 5 6
```

This means that you can call the arguments of `c()` after you have converted it to the Python function `r['c']`. You can also do it in two steps by assigning the `r['c']` function to a variable first and then calling it as you usually call functions in Python:

```
>>> c = r['c']
>>> print c(1, 2, 3, 4, 5, 6)
[1] 1 2 3 4 5 6
```

If you want to use the `r` object as a function instead, you can do it as follows:

```
>>> print r('c(1,2,3,4,5,6)')
[1] 1 2 3 4 5 6
```

Notice that the argument in the `r` call is converted to a string type (using single quotation marks).

These three approaches work for any R functions, e.g., if you want to generate a vector in Python using the R function `seq()`:

```
>>> y = r.seq(1, 10)            #using the dot syntax
>>> print y
```

```
[1] 1 2 3 4 5 6 7 8 9 10
>>> s = r['seq']                #dictionary-like
>>> print s(1,10)
[1] 1 2 3 4 5 6 7 8 9 10
>>> print r('seq(1,10)')        #function-like
[1] 1 2 3 4 5 6 7 8 9 10
```

Q & A: WHICH OF THE THREE WAYS TO ACCESS R OBJECTS SHOULD I USE?

Our suggestion is this: the simpler the better, but much depends on your preferences. You might even decide to mix the different ways to get R objects in a Python program. For example, `r.pi` looks slightly simpler than `r('pi')`, but you may prefer the latter. In all cases, you have to remember that the result of retrieving an R object in Python is always an R vector; therefore, you have to use indexing to specifically access its elements.

13.3.4 Creating Matrices

You can create a matrix in R as follows:

```
> y = seq(1,10)
> matrix(y, ncol = 5)
      [,1]  [,2]  [,3]  [,4]  [,5]
[1,]     1     3     5     7     9
[2,]     2     4     6     8    10
>
```

In Python, you have to convert both `seq()` and `matrix()` R functions into Python objects using `robjects.r`. You can do it in the same three ways as in Section 13.3.3.

Accessing R Functions as Attributes of `robjects.r` *Using the Dot Syntax*

```
>>> import rpy2.robjects as robjects
>>> r = robjects.r
>>> y = r.seq(1,10)
>>> print r.matrix(y, ncol = 5)
      [,1]  [,2]  [,3]  [,4]  [,5]
[1,]     1     3     5     7     9
[2,]     2     4     6     8    10
```

```
>>> print r.matrix(y, nrow = 5)
       [,1]   [,2]
[1,]      1      6
[2,]      2      7
[3,]      3      8
[4,]      4      9
[5,]      5     10
```

Notice that you can also reassign the function to a variable and then use it:

```
>>> import rpy2.robjects as robjects
>>> r = robjects.r
>>> y = r.seq(1,10)
>>> m = r.matrix
>>> print m(y, nrow = 5)
       [,1]   [,2]
[1,]      1      6
[2,]      2      7
[3,]      3      8
[4,]      4      9
[5,]      5     10
```

Accessing R Functions Using the `[]` *Operator on* `robjects.r` *Like You Would Use a Dictionary*

```
>>> import rpy2.robjects as robjects
>>> r = robjects.r
>>> y = r['seq'](1,10)
>>> print r['matrix'](y, ncol = 5)
       [,1]   [,2]   [,3]   [,4]   [,5]
[1,]      1      3      5      7      9
[2,]      2      4      6      8     10
```

Also in this case you can reassign the function to a variable and then use it:

```
>>> import rpy2.robjects as robjects
>>> r = robjects.r
>>> y = r['seq'](1,10)
>>> m = r['matrix']
>>> print m(y, ncol = 5)
       [,1]   [,2]   [,3]   [,4]   [,5]
[1,]      1      3      5      7      9
[2,]      2      4      6      8     10
```

Calling `robjects.r` Like You Would Do with a Function, Passing the R Functions as Arguments

```
>>> import rpy2.robjects as robjects
>>> r = robjects.r
>>> y = r('seq(1,10)')
>>> print r('matrix('+y.r_repr()+', ncol = 5)')
       [,1]    [,2]    [,3]    [,4]    [,5]
[1,]      1       3       5       7       9
[2,]      2       4       6       8      10
```

Notice that the arguments are passed to the `r` object in the form of strings. This implies that the following commands

```
>>> y = r('seq(1, 10)')
>>> print r('matrix(y, ncol = 5)')
```

will return an error message because `y` is not a string (but a vector of numbers), and in Python you cannot mix different data types. Thus, you first have to convert `y` into a string and then concatenate it to `'matrix()'`. The conversion into a string can be nicely done with the `r_repr()` method, which works on all R objects and returns a string representation that can be directly evaluated as R code:

```
>>> y = r.seq(1, 10)
>>> y.r_repr()
'1:10'
```

This applies in general to any R commands: you can write them as strings and then pass them as arguments to `robjects.r` when you call it. For example, the following R command

```
> f = read.table("RandomDistribution.tsv", sep = "\t")
```

can be written in Python as follows:

```
>>> import rpy2.robjects as robjects
>>> r = robjects.r
>>> f = r("read.table('RandomDistribution.tsv', sep = '\t')")
```

13.3.5 Converting Python Objects into R Objects

The previous examples show how to create R objects in Python using R functions in the form of `robjects.r` attributes, dictionary keys, or function arguments. The content of the resulting objects can be either accessed as you would access Python arrays (through the `[]` operator, e.g., `y[0]`) or reused in R functions (as `y` in `matrix()`). However, in many cases, it will turn out to be very useful to convert a Python object (e.g., a list or a tuple) into an R object, which can be used in R functions. In fact, suppose you read the table in Figure 13.1 from a file and save its content to a Python list of lists (e.g., using `readlines()`). What if you want to calculate with R the mean of the values of the table's first column? To this purpose, you can use the `FloatVector()` method of `robjects` that converts lists or tuples of floating numbers (or of strings containing floating numbers) into an R array of floating numbers.

```
>>> F = open('RandomDistribution.tsv')
>>> lines = F.readlines()
>>> l = []
>>> for line in lines:
...     l.append(float(line.split()[0]))
>>> R_vector = robjects.FloatVector(l)
>>> print r.mean(R_vector)
[1] 6127.931
```

The `StrVector()` and `IntVector()` methods of `robjects` convert Python lists (or tuples) into R arrays (i.e., readable by R functions) of strings and integers, respectively:

```
>>> float_vector = robjects.FloatVector([3.66, 2.16, 7.34])
>>> print float_vector
[1] 3.66 2.16 7.34
>>> float_vector = robjects.FloatVector(['3.66', '2.16', '7.34'])
>>> print float_vector.r_repr()
c(3.66, 2.16, 7.34)
>>> string_vector = robjects.StrVector(['atg', 'aat'])
>>> print string_vector
[1] "atg" "aat"
>>> print string_vector.r_repr()
c("atg", "aat")
>>> int_vector = robjects.IntVector(['1', '2', '3'])
>>> print int_vector
```

```
[1] 1 2 3
>>> int_vector = robjects.IntVector([1, 2, 3])
>>> print int_vector.r_repr()
1:3
```

Finally, it must be pointed out that R vector-like objects can be accessed with the delegator `rx`, which represents the R operator "`[`":

```
>>> print float_vector.rx()
[1] 3.66 2.16 7.34
>>> print string_vector.rx()
[1] "atg" "aat"
>>> print string_vector.rx(1)
[1] "atg"
>>> print int_vector.rx()
[1] 1 2 3
```

13.3.6 How to Deal with Function Arguments That Contain a Dot

If you want to calculate with R the mean of the values listed in the first column of the table in Figure 13.1, you can write the following:

```
> f = read.table("RandomDistribution.tsv", sep = "\t")
> f_matrix = matrix(f, ncol = 7)
> mean_first_col = mean(f[,1])
> mean_first_col
[1] 6127.931
```

This translates into Python as follows:

```
>>> import rpy2.robjects as robjects
>>> r = robjects.r
>>> f = r("read.table('RandomDistribution.tsv', sep = '\t')")
>>> f_matrix = r.matrix(f, ncol = 7)
>>> mean_first_col = r.mean(f_matrix[0])
[1] 6127.931
```

But what if you want to deal with, e.g., missing values in the input table? In R you would simply set the `na.rm` argument of the R `mean()` function to `FALSE`. However, in Python, the dot has a precise function, and its use for a different purpose would cause the program to behave wrongly or break. In other words, everything with R function arguments in Python works fine unless one of the argument names contains a dot (e.g., `na.rm`).

In this case, the standard choice consists of translating the dot into a "_" in the argument name:

```
> f = read.table('RandomDistribution.tsv', sep = '\t')
> m = mean(f[,7], trim = 0, na.rm = FALSE)
```

would become the following in Python:

```
>>> f = r("read.table('RandomDistribution.tsv', sep = '\t')")
>>> r.mean(f[3], trim = 0, na_rm = 'FALSE')
<FloatVector - Python:0x106c82cb0/R:0x7fb41f887c08>
[38.252747]
```

See Example 13.3 for more on this.

13.4 EXAMPLES

Example 13.1 Running a Chi^2 Test

The following script tests if the expression of two genes is correlated or independent. An input file (`Chi-square_input.txt`) is shown in Figure 13.2. The first column contains the sample number, and the second column contains the expression level (H = High, N = Normal) of two genes (GENE1, GENE2) in the samples.

SAMPLE	GENE1	GENE2
1	H	H
2	H	H
3	N	N
4	H	N
5	N	N
6	N	N
7	N	N
8	H	H
9	N	N
10	H	N
11	H	H
12	N	N
13	N	N
14	N	N
15	N	N
16	H	H
17	H	N
18	H	H
19	N	H
20	H	H
21	N	N

FIGURE 13.2 Content of the Chi-square_input.txt file used in Example 13.1. *Note:* The first column contains the sample number, and the second and third columns contain the expression level (H = High, N = Normal) of two genes (GENE1, GENE2) in each sample.

Notice that you can choose a shorter name for the imported module; e.g., you can write

```
import rpy2.robjects as ro
```

In this example, we use a short name for `rpy2.robjects`.

```
R session:
> h = read.table("Chi-square_input.txt",header = TRUE,sep \
    = "\t")
> names(h)
[1] "SAMPLE" "GENE1" "GENE2"
> chisq.test(table(h$GENE1,h$GENE2))

     Pearson's Chi-squared test with Yates' continuity \
         correction

data: table(h$GENE1, h$GENE2)
X-squared = 5.8599, df = 1, p-value = 0.01549

Warning message:
In chisq.test(table(h$GENE1, h$GENE2)) :
Chi-squared approximation may be incorrect

Corresponding Python session:
import rpy2.robjects as ro
r = ro.r
table = r("read.table('Chi-square_input.txt', header=TRUE,\
    sep='\t')")
print r.names(table)
cont_table = r.table(table[1], table[2])
chitest = r['chisq.test']
print chitest(table[1], table[2])
```

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

The result from the Chi-squared test looks like this:

```
     Pearson's Chi-squared test with Yates' continuity \
         correction
...
X-squared = 5.8599, df = 1, p-value = 0.01549
```

Notice that the following code does basically the same:

```
import rpy2.robjects as ro

r = ro.r
table = r("read.table('Chi-square_input.txt', header=TRUE,\
    sep='\t')")
```

```
contingency_table = r.table(table[1], table[2])
chitest = r['chisq.test']
print chitest(contingency_table)
```

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

Example 13.2 Calculating Mean, Standard Deviation, *z*-score, and *p*-value of a Set of Numbers

```
R session:
> f = read.table("RandomDistribution.tsv", sep = "\t")
> m = mean(f[,3], trim = 0, na.rm = FALSE)
> sdev = sd(f[,3], na.rm = FALSE)
> value = 0.01844
> zscore = (m -value)/sdev
> pvalue = pnorm(-abs(zscore))
> pvalue
[1] 0.3841792
```

```
Corresponding Python session:
import rpy2.robjects as ro
r = ro.r
table = r("read.table('RandomDistribution.tsv',sep = '\t')")
m = r.mean(table[2], trim = 0, na_rm = 'FALSE')
sdev = r.sd(table[2], na_rm = 'FALSE')
value = 0.01844
zscore = (m[0] - value) / sdev[0]
print zscore
x = r.abs(zscore)
pvalue = r.pnorm(-x[0])
print pvalue[0]
```

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

Notice that to extract a column from the input file, in Python you have to count from 0. This means that column `f[3]` in R corresponds to column `f[2]` in Python. Moreover, the R objects returned by `robjects.r` are vectors. Therefore, if you want to utilize their value, you have to extract it using the `[]` operator. For example, in this example, the *z*-score cannot be calculated directly using

```
zscore = (m – value) / sdev
```

as you would do in R, because `m` and `sdev` are vectors.

Example 13.3 Creating Plots Interactively

Plots with R may or may not be made interactively. Here, we show how to create R plots interactively using functions such as `plot()` or `hist()`.

```
R session:
plot(rnorm(100), xlab = "x", ylab = "y")

Corresponding Python session:
import rpy2.robjects as ro
r = ro.r
r.plot(r.pnorm(100), xlab = "y", ylab = "y")
```

Another example:

```
R session:
f = read.table("RandomDistribution.tsv", sep = "\t")
plot(f[,2], f[,3], xlab = "x", ylab = "y")
hist(f[,4], xlab = 'x', main = 'Distribution of values')

Corresponding Python session:
import rpy2.robjects as robjects
r = robjects.r
table = r("read.table('RandomDistribution.tsv',sep = '\t')")
r.plot(table[1], table[2], xlab = "x", ylab = "y")
r.hist(table[4], xlab = 'x', main = 'Distribution of values')
```

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

Running this example could be frustrating because the plots will appear and immediately disappear from your screen due to the program execution completion. One trick to keep them on the screen for, say, five seconds each, is to use the `sleep()` method from the time module to suspend the program run for five seconds after the execution of each plot command:

```
import rpy2.robjects as ro
import time

r = ro.r
r.plot(r.rnorm(100), xlab = "y", ylab = "y")
time.sleep(5)

table = r("read.table('RandomDistribution.tsv',sep = '\t')")
```

```
r.plot(table[1], table[2], xlab = "x", ylab = "y")
time.sleep(5)

r.hist(table[4], xlab = 'x', main = 'Distribution of values')
time.sleep(5)
```

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

Example 13.4 Saving Plots to Files

To plot to a file with R, you have to set a graphical device like png or pdf. In Python you need to import importr, a method from the rpy2.robjects.packages module. importr makes it possible to retrieve the grDevices object, the attributes of which are grDevices.png and other devices you may need. After finishing the plot, the graphical device must be closed using the dev.off() R command. Here, we show the same examples as in Example 13.3, but plots are saved to .png files:

```
import rpy2.robjects as ro
from rpy2.robjects.packages import importr
r = ro.r
grdevices = importr('grDevices')
grdevices.png(file = "RandomPlot.png", width = 512, \
    height = 512)
r.plot(r.rnorm(100), ylab = "random")
grdevices.dev_off()
```

RandomPlot.png is shown in Figure 13.3. Here is a second example:

```
import rpy2.robjects as ro
from rpy2.robjects.packages import importr
r = ro.r
table = r("read.table('RandomDistribution.tsv',sep = '\t')")
grdevices = importr('grDevices')
grdevices.png(file = "Plot.png", width = 512, height = 512)
r.plot(table[1], table[2], xlab = "x", ylab = "y")
grdevices.dev_off()
grdevices.png(file = "Histogram.png", width = 512, height = 512)
r.hist(table[4], xlab = 'x', main = 'Distribution of values')
grdevices.dev_off()
```

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

Plot.png and Histogram.png are shown in Figure 13.4.

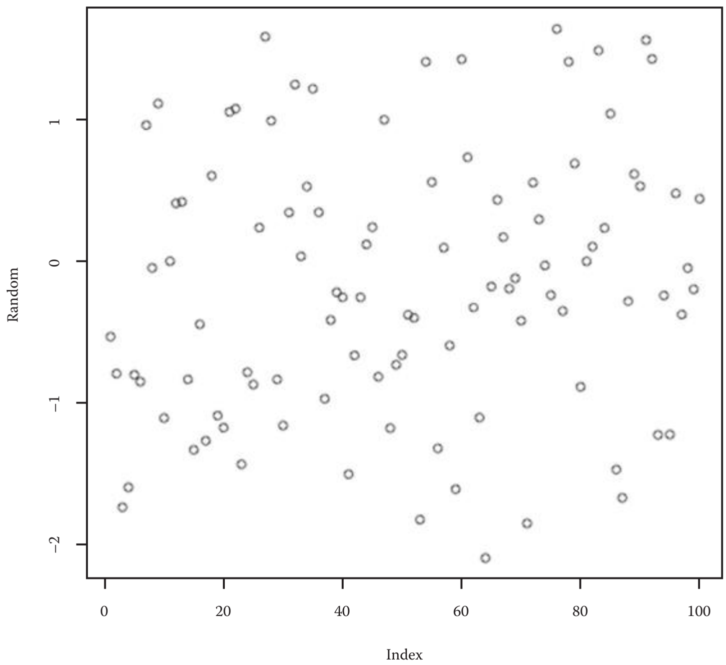

FIGURE 13.3 Random plot generated with `RPy2` tools. *Note:* Plot obtained in the first part of Example 13.4 (`RandomPlot.png`).

13.5 TESTING YOURSELF

Exercise 13.1 Statistical Calculations

Calculate mean, standard deviation, z-score, and p-value of sets of values obtained from your experiments.

Exercise 13.2 Chi-square Test for Smokers and Nonsmokers and Lung Cancer

Carry out a chi-square test to check whether two variables x and y are independent, where x = yes/no (yes if the sample patient was a smoker) and y = yes/no (yes if the sample patient died of lung cancer). You can either retrieve patient samples from the Internet or invent them.

Exercise 13.3 Plot a Histogram and Save It to a .pdf File

Read a list of numbers from a file, use them to plot a histogram using R with Python, and save the plot to a .pdf file.

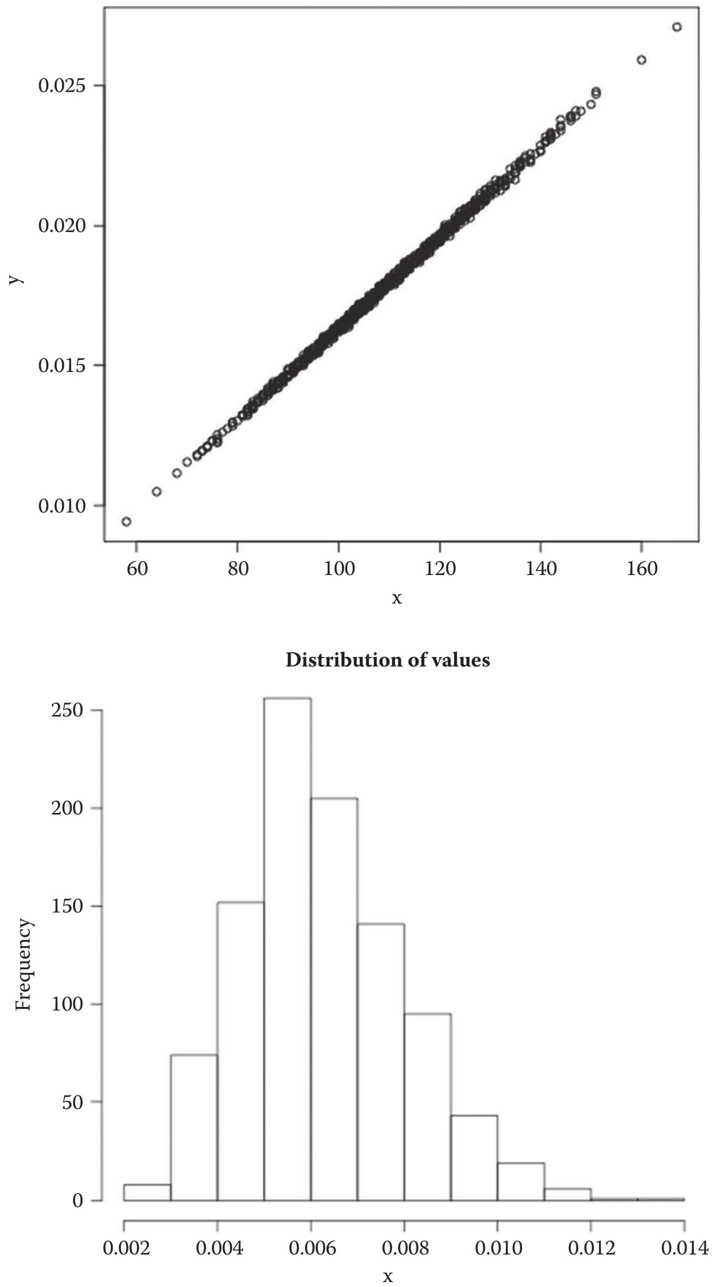

FIGURE 13.4 Plot and histogram generated with `RPy2` tools from data in Figure 13.1. *Note:* Plots obtained in the second part of Example 13.4 (`Plot.png` and `Histogram.png`).

Exercise 13.4 Plot a Boxplot

Plot the boxplot of the second, fourth, and fifth columns of a table of your choice. Color it in orange, set *x* and *y* labels, and the plot title. Do it both interactively and saving the plot to a file.

Hint: `r.boxplot(f[1], f[3], f[5], col = "orange", xlab = "x", main = "Boxplot", ylab = "y")`

Exercise 13.5

Plot a heatmap of two sets of data.