Chapter 1:

An Introduction to R

Often, there is much consternation among students as to why they have to learn *yet another statistical package*. Can't those dad-gum professors make up their minds? Are they trying to drive us crazy? Are they getting kick-backs from the statistical software salesmen?

The answer, more than likely, is that the specific professor truly believes that *this* statistical package is the best available (by whatever measure he or she uses). I am no different. I truly believe that, overall, R is the best statistical package available for the following four reasons:

- It is free (both free pizza and free speech).
- It is flexible.
- It is powerful.
- It matches how science *should* be done.

Let us take these four reasons in the above order and explain what I mean by each.

free

First, it is free. The "R Project for Statistical Computing" is a not-for-profit foundation created for the sole purpose of creating a piece of software that encourages scientific innovations in any field that uses statistics.[1] The cost of the software is zero. That means students (and professors and businesses) do not have to pay fees (license fees or purchase fees) to use the software. As such, as budgets tighten, expect R to become the statistical environment of choice for universities and for businesses. Furthermore, because you are able to load it on a USB drive, you do not have to wander around searching for a computer sporting R like you do with other statistical packages. It is also free in the sense that you are free to modify the code to make it better. This latter part is what allows R to become better, stronger, and faster as time passes.

flexible

powerful

Second and third, it is flexible and powerful. The base distribution of R contains only those parts of R that are universally useful. Thus, it is small and fast to download and begin. However, there are assorted packages for just about any statistical analysis. And, for those methods that are not currently supported by R, you are free to create your own functions and packages for everyone to use. Furthermore, it is a scripting language. As

[1] The URL for the R Project is http://www.r-project.org/. The URL for downloading R is http://cran.r-project.org/.

OS	File link
Linux	your specific distribution
MacOS X	Files: (latest version)
Windows	**base**, and then the **Download** link at the top

Table 1.1: *A list of the appropriate links for downloading* `R` *from the website.*

such, you can program it to do the same tasks repeatedly (with slight modifications) so that robust analyses can be done.

Finally, it matches how science *should* be done. `R` offers a definite separation between the data and the analysis. It also offers a way of keeping track of your analysis as you *do* the analysis. The former allows you to keep your original dataset unmolested. The latter allows your analysis to be checked and replicated. Both of these are important hallmarks of doing science. **science** **replicability**

It is for these reasons that I use the `R` statistical environment. It is also for these reasons that I prefer to teach using it. I am not saying it will cure world hunger, but it will help you learn the right way to do science better than some other statistical programs — plus, it is free.

1.1: Installing R on your computer

As with almost any other program, installing `R` requires two steps: downloading it and installing it. If you wish to install `R` to a USB drive, that is also an option.[2]

1.1.1 Step 1: Download

The `R` program can be downloaded via the Internet. Using your web browser, go to `http://cran.r-project.org/`. Once there, click on the type of computer operating system (OS) you have: Linux, MacOS, or Windows. On the next page, you will click on the appropriate link (depending on your OS) and download it to your desktop or some place just as convenient (see Table 1.1). The specifics are up to your browser and your computer operating system.

[2] One would want to do this if one uses several computers and cannot be certain that `R` will be installed. The steps are the same as for a normal installation, with one change: Instead of selecting `C://R` as the destination folder, you will select the `R` folder on your USB drive.

1.1.2 Step 2: Install

Once the file is on your computer, run the file (an installer) and answer the questions it asks. Usually the default selections are appropriate. The only thing you may wish to change is the destination folder. If you want to save R to your USB drive so you can bring R with you, you will have to select *that* as your destination folder.

After the installer finishes, R is installed on your computer.[3]

1.2: A quick, sample session

As an example, let us do a quick, sample session that checks to make sure R is properly installed on your computer and which does no serious statistical analysis. In this session, you will start R, open a new script window, set the working directory, type in an R script, execute the script, then save the script to your current working directory.

Before you start this session, you should create a directory for your project, called Chapter1, a place where your analysis will take place. In reality, you *should* have a different directory for each project. This is appropriate, as it keeps your projects separate.

1.2.1 Step 1: Start R

If R is installed on your machine, find the icon and double-click on it. If it is only on your USB drive, you will need to find the program. Likely, it is located at USB:\\R-#.##.#\bin\ (where the #s are digits naming your version of R, and USB is the drive letter of your USB drive). Double-click Rgui.exe.[4] Your screen should look something like Figure 1.1.

USB

The R window has one sub-window right now — the 'R Console' window. The Console window is where all the analysis really gets done. All commands you type must eventually find their way to the Console window before R will actually execute them. However, the Console window should *not* be where you do your analysis. It is bad science to do your analysis in the

console window

[3] Actually, if you install it to your USB drive, it is not installed on the computer, *per se*. You will have to double-click the R/bin/Rgui.exe file each time you wish to run R from the USB drive. Doing it this way will mean you should become best friends with the setwd() function.

[4] If you installed a different version of R, then the name of the top folder will reflect that version. Also note that the specifics are for Windows™ installations. MacOS™ will use similar — though not exact — processes.

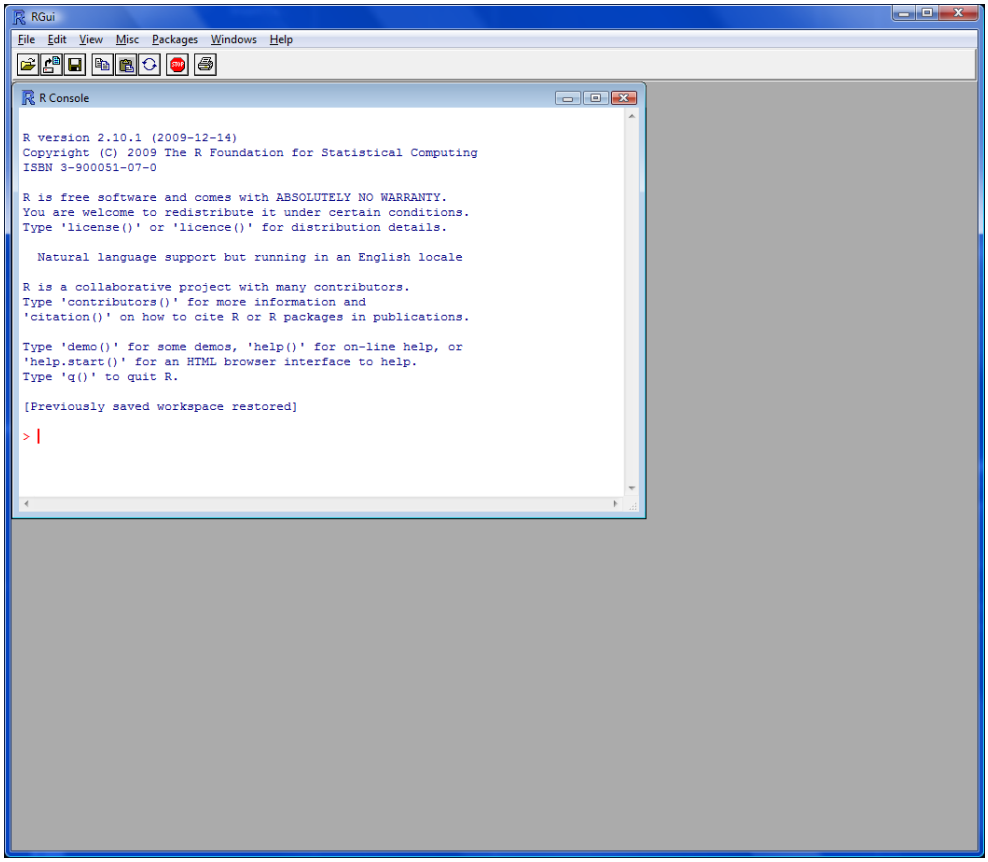

Figure 1.1: *The opening screen for* `R` *in Windows™. This is for* `R` *version 2.10.1, which was released on December 14, 2009 (see the top line of the Console window. Your version will differ.*

`Console` window, since the commands you type there are lost once you press ENTER. This means *replication* of your findings will be extremely difficult and time consuming. It is proper to type your analysis in a separate script window and send them to the `Console` window to be executed. Actually, 'proper' is not entirely correct here, '*the only acceptable manner*' is much more accurate.

replication

While there are several third-party text editors which allow you to type your analysis and send it to the `Console` window, `R` provides a text

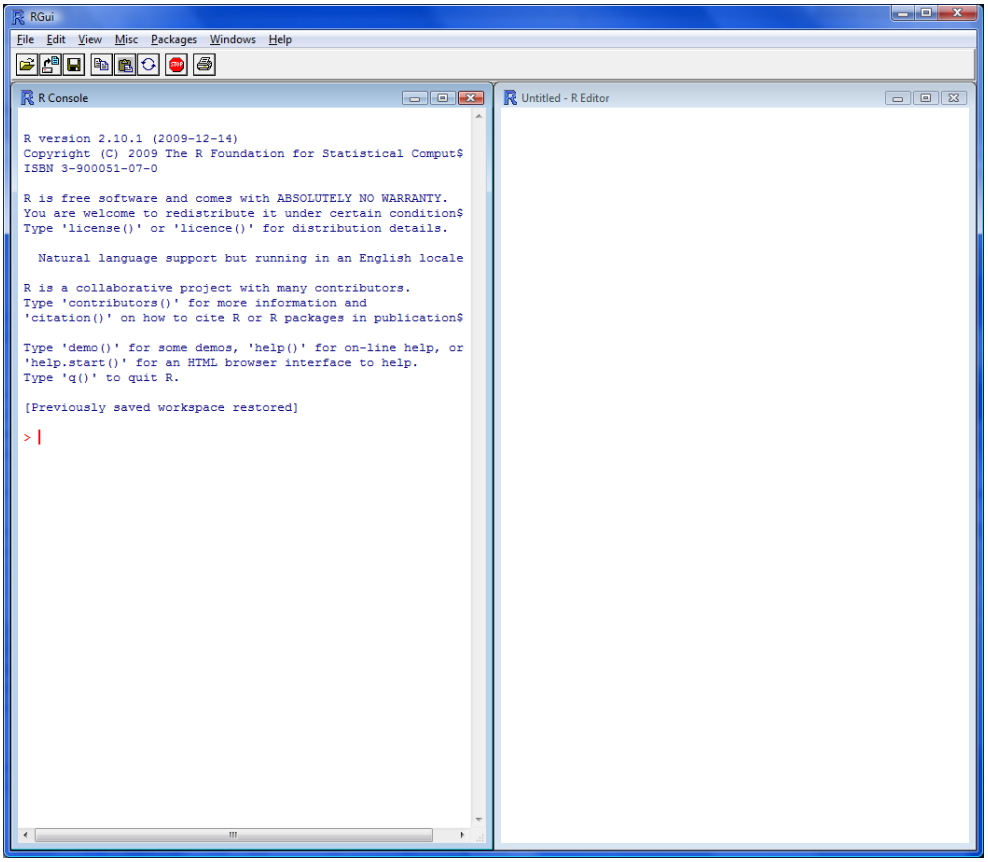

Figure 1.2: *The* `R` *Window after tiling the two sub-windows (*`Console`*, left, and* `Script`*, right). This screen capture was done in Windows™.*

editor that is more than sufficient.[5] The primary advantage to the built-in text editor is that it is easy to send lines of code in the `Script` window to the `Console` window to be executed — just type `Ctrl+r` when your cursor is on the appropriate line. Or, highlight the part you want to execute and type `Ctrl` and `r`.[6]

Ctrl + r

[5] Throughout this book, I will tell you what I do and what I use. I use the text editor that comes with `R`. It does everything I need it to do. If it comes up short for you, then investigate some of the third-party options, such as eMacs, R Studio, or Revolution.

[6] This key combination is the method to use in Windows™. In MacOS™, you will use `Command+Enter`.

1.2.2 Step 2: Start a new script

If you are using a third-party text editor, follow the directions provided by that vendor (which may be just copy-paste). If you are using the R text editor, open a new script: "File | New script...". You now have a second sub-window in the R window. This new window is titled "Untitled - R Editor". Tile the two windows so you can see both at the same time: "Windows | Tile Vertically". At this point, your R window should look similar to Figure 1.2 (Windows™ only).

tiling

1.2.3 Step 3: Type in the script

You have been taught to show your work from the first math class you took back in kindergarten. Continuing a long and storied tradition, you should type your script into the Script window, run it from there, and *save it* for future reference and evidence.

Type the simple script below into the Script window. The script does the following three things: creates a univariate dataset, analyzes the dataset, and plots the data. The first goal is accomplished with a single line. The second by as many lines as depth of analysis you wish to perform. The last by two lines—one for each of the two manners of graphing univariate data.

script window

The code is as follows (make sure you type it in correctly):

```
# Basic statistics
#   script1.R

getwd()
setwd("F:/RFS/Chapter1/")

set.seed(370)
x <- runif(50000, min=10, max=20)

head(x)
tail(x)

length(x)

mean(x)
median(x)

var(x)
sd(x)
IQR(x)

min(x)
max(x)
quantile(x)

mean(x, trim=0.05)
```

```
27  quantile(x, 1:100/100)
28
29  png("01boxplot.png",height=4,width=4,units="in",res=600)
30  par(cex=0.8,cex.lab=0.8,cex.axis=0.8)
31  boxplot(x)
32  dev.off()
33
34  png("01histogram.png",height=4,width=4,units="in",res=600)
35  par(cex=0.8,cex.lab=0.8,cex.axis=0.8)
36  hist(x)
37  dev.off()
```

octothorpe

What does this script do? The first two lines are comments. The comment character is the hash symbol, `#`. Anything on the line following the `#` is ignored by `R`. Commenting your script is a very good idea; it makes the script more understandable to everyone. It also allows *you* to return to the script, understanding your logic. It may help to think of the comments as the outline for your analysis.

working directory

The fourth line determines your working directory. [7] If you save this script by pressing `Ctrl+s`, it will be saved in that directory. If you specify a datafile or save a picture, it will be saved in that directory, unless you specify a path to the new directory. It is good practice to set, as your working directory, your current project directory. That will help ensure all scripts, data files, and pictures are easily available. The fifth line sets the working directory. The one parameter is the path to the directory. That path can be relative (to the current working directory) or absolute. Line 5 provides an absolute path to the directory.[8]

It is good practice, if you are working on more than one computer, or are collaborating with other researchers, to have `setwd()` lines for each computer, then comment out the lines you do not need with `#` symbols.

random numbers

seed

The seventh line sets the random number seed, which guarantees your dataset will be the same as mine. No computer is able to produce random numbers; they produce 'pseudo-random' numbers. The algorithm to produce the numbers varies from system to system, but they all are based on a number called the *seed*. Changing the value of the seed will change the series of random numbers produced. If a seed is not specifically set, then the computer will usually use the current time as the seed. This is useful if you want your trials randomized. It is not if you need to replicate the results. Thus,

[7] In general, if a Windows™ command uses `Ctrl`, the corresponding MacOS™ command will use `Command`. There are a few exceptions to this, however.

[8] We know it is an absolute path since it starts with the drive letter.

setting a seed allows the results in this book to correspond to the results you get, allowing you to check your work.

The next line creates a variable named 'x' and puts 50,000 uniform random numbers, ranging from 10 to 20, in the variable x.[9] In the language of probability, x is a vector of 50000 realizations of a random variable X, such that

data

$$X \overset{\text{iid}}{\sim} \mathcal{U}(10, 20)$$

The 'r' in runif indicates its random aspect; the 'unif,' the uniform distribution. A few other common distributions include norm (Normal or Gaussian distribution), t (Student's t-distribution), and exp (Exponential distribution). Each of the other random number generator distributions have different options, see Appendices A and B, and well as the R Help for specifics.[10]

The next line displays the first six values of x — called the head. The tail is the last six values. Had I wanted to display the first 15 values, I would run `head(x, 15)`. Doing things like helps check that your script is giving you what you expect. I expect this vector to contain 50,000 values between 10 and 20. Scanning the vector tells me that I appear to be getting something like that. The ability to examine the data is one reason why I saved it to a variable. Another reason is that I can now perform an analysis on the same data multiple times.

head and tail

Line 13 displays how many elements are in x — a.k.a. the sample size. In statistical notation, the sample size is usually represented by the variable n. As such, you may frequently see in programs a line such as:

```
n <- length(x)
```

The next several lines find the following information about the data stored in x: mean $\overline{x}$, median $\widetilde{x}$, variance s^2, standard deviation s, interquartile range

statistics

[9] The assignment operator in R is not the equals sign, =, it is the left-arrow <–, a less-than sign and a hyphen. With that said, I have yet to run into problems using an equals sign in R. This book will adhere to using the standard <–.

[10] This is as good a place as any to introduce the R Help and R Search functions. If you know the actual command or statement, but you forget its specifics, type a question mark followed by the command in quotation marks(for example, `?"rnorm"`). However, if you do not know the actual command, but you know a word close to it or what it does, type two questions marks followed by the word or words in quotation marks (for example, `??"random number"`).

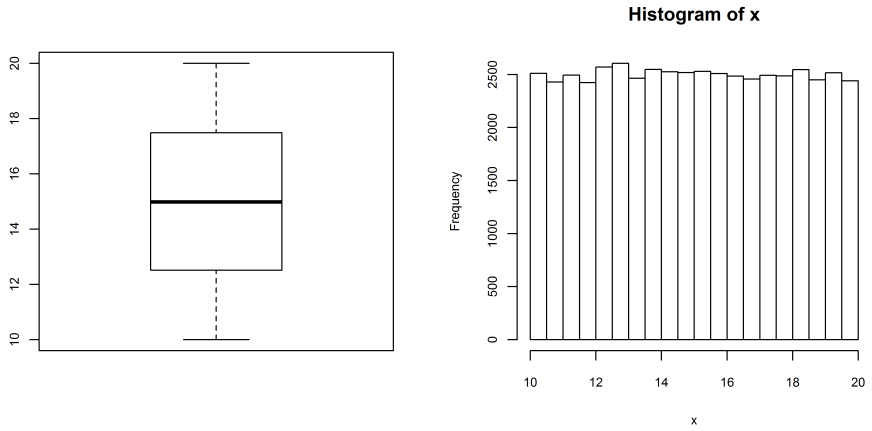

Figure 1.3: *The results of initial graphical analysis of the random uniform dataset. Left Panel: Box-and-whiskers plot. Right Panel: Histogram.*

(IQR), minimum value, maximum value, the five-number summary (quartiles 0 through 4), the 5%-trimmed mean, and all 100 percentiles.[11]

Note the following subtle points. First, I named the data vector `x`, which is lower case. The vector contains 50,000 realizations of a random variable; i.e., the vector is data. Second, the statistics calculated on `x` are lower case, too. They are realizations of random variables, themselves.

data

Lines 29–32 plot the data as a box-and-whiskers plot (Figure 1.3, left), which displays the median (heavy bar in center) and Quartiles 1 and 3 (the ends of the central box). It also displays bars at either the minimum and maximum values or, if there are outliers (signified by dots on the box-and-whiskers plot), at the data value just inside the fences.[12] There are no outliers in this dataset, so the upper bar is the maximum value in the dataset and the lower bar is the minimum value in the dataset. Your box-and-whiskers plot should look exactly like that in Figure 1.3, Left Panel. Note that lines 29

box plot

[11] Where quartiles divide the dataset into 4 equal quarters (hence 'quartile'), percentiles divide the dataset into 100 equal parts. 'Equal' in this sense is number of elements. Note, however, that equal is not truly equal unless the number of elements in a dataset has certain properties; it means 'approximately equals.'

[12] The inner fences are the boundary between "acceptable" values and "outliers" in the data. The lower inner fence is $Q_1 - 1.5 \times IQR$; the upper, $Q_3 + 1.5 \times IQR$.

and 32 work together to save this graphic to your computer in your working directory as the file `01boxplot.png`. **saving graphics**

The histogram (lines 34–37) separates the data into bins of equal width (as a default) and plots the frequency of data in each bin. Your histogram should look like Figure 1.3, Right Panel. As the data comes from a Uniform distribution, we expect the histogram to be flat. All bumpiness is due to the inherent randomness of sampling and the small sample size.[13] **histogram**

1.2.4 Step 4: Save the script

If this were an analysis you used for your research, you would definitely want to save it. Saving the script is rather straight-forward: `File | Save`. If you do not see the `Save` option, then the active sub-window is not the script. Click on the script window and retry the save procedure.[14] **save script**

Save this script as `s1intro.R` in the `Chapter1` folder.

1.3: A second example

Let us have another example. In this example, let us analyze data imported from the Internet. In the previous example, we created our vector of data using the `runif()` function. Here, we will use the `read.csv()` function to import the data. **import**

> ***Note***: `R` is able to load many different data types. One of the most common data formats is the comma-separated variable format (`.csv`). Another is the tab-delimited format (`.txt`). Data stored in either of these two formats are more accessible than data stored in proprietary formats, since any statistical program (and any word processing program) can open them. Natively, `R` cannot open data stored in Stata's™`dat` format, Excel's™ `.xls` or `xlsx` formats, or any other proprietary format. However, loading the `foreign` package with `library(foreign)` makes available commands allowing you to import data in most all file formats.

[13] Technically, the histogram is an approximation of the probability density function (pdf). The pdf is a 'formula' for the distribution (it gives the likelihood of a specified value). Different distributions have different pdfs and (therefore) different expected histograms. As it is an approximation, the histogram will not *be* the pdf; it will only *approximate* it. Better approximations occur with larger sample sizes.

[14] In Windows™, you can use `Ctrl + s`; in MacOS™, `Command + s`.

For this example, let us demonstrate more capabilities by using a remote datafile. Its address is

```
http://rfs.kvasaheim.com/data/positioningtubes.csv
```

The data consist of a series of length measurements of positioning tubes manufactured by a leading company. Looking at the data, you notice that the first row is the word"length." This is the name of the variable.

Here is the script.

```
###  Filename: s2intro.R
###  Purpose:  simple analysis of the lengths
###    of positioning tubes produced by Company XY

### Preamble

setwd("F:/RFS/Chapter1/")
#setwd("E:/CompanyXY/")
#setwd("C:/Consulting/CompanyXY/")


# Access an external function
source("http://rfs.kvasaheim.com/Rfctns/means.R")

# Load and attach data
tube <- read.csv("http://rfs.kvasaheim.com/data/positioningtubes.csv")
attach(tube)

# What is the variable's name?
names(tube)


### Begin analysis

# Central Tendency
mean(length)
means(length,type="geometric")
means(length,type="harmonic")
median(length)

# Spread
sd(length)
var(length)
IQR(length)


### Create the graphics
boxplot(length, main="Boxplot of Positioning Tube Length",
        ylab="Positioning Tube Length [cm]")

hist(length, main="Histogram of Positioning Tube Length [cm]",
        xlab="Positioning Tube Length")
```

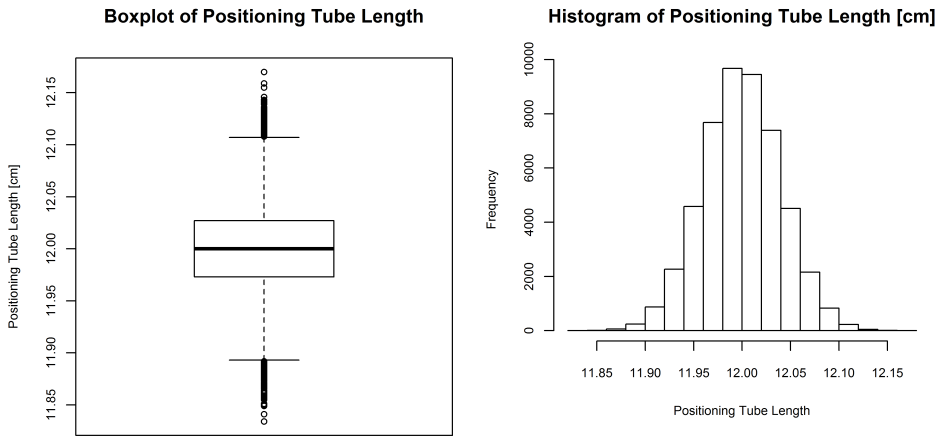

Figure 1.4: *The results of initial graphical analysis of the* `positioningtube` *dataset. Left Panel: Box-and-whiskers plot. Right Panel: Histogram. Note the difference in histogram shapes between this dataset and the previous. Also note how that difference is reflected in the box-and-whiskers plot.*

Note that the first few lines are comments giving information about the file, the analysis, and the data. After that, we get and set the working directory. Since I use this file on three computers, I have three different `setwd()` commands, one for each computer, with two commented out.

Now, I know I will need to calculate the geometric mean of the data. `R` does not have such a function in the base package, so I will import the function from the Internet. The command is `source()`, with the filename being the argument.

Next, we load the data using the `read.csv()` function. Alternatively, we could have used our web browser to download the datafile to our working directory and imported it from there. With this, the entire dataset is stored in our variable `length`.

How did I know the variable was `length`? The line `names(tube)` lists the variable names in the dataframe. Be aware: The dataframe is a variable that contains variables. You can think of `tube` as a container for all variables in the dataset.

The analysis proceeds, starting at line 23. We calculate measures of central tendency and measures of spread. Finally, we produce a box-and-whiskers plot and a histogram: Figure 1.4. Note how these look different

than the box-and-whiskers plot and histogram from the last example. Also note that these two were displayed to your computer screen and not saved to yout working directory. If you ran the entire script in one fell swoop, you may not have seen the box-and-whiskers plot before the histogram replaced it.

> ***Note***: In the previous example, I showed the default box-and-whiskers plot and histogram. In this example, I utilize some of the available options to make the graphics more expressive. Check to see what each option does; I leave it as an exercise for you to make the graphics look the way you prefer.

1.4: A third example

Let us perform a more in-depth analysis, which will hint at some of R's latent power. In this example, we will test one aspect of the appropriateness of the two-sample t-test on data that does *not* meet its Normality assumption. You may wish to skim through Chapter 6 to know what a two-sample t-test is and why we use it.

Monte Carlo

Monte Carlo experiments are a group of simulation algorithms relying on repeated sampling to estimate statistics. Such experiments are appropriate when the equations are too difficult to derive and when computing power is cheap — such as today. To perform Monte Carlo experiments, one should understand the theoretical process underlying the data.

Monte Carlo experiments generally consist of three steps:

1. First, random samples need to be taken.
2. Second the test statistic needs to be calculated.
3. Finally, these two steps are repeated a large number of times.

For this example, let us test the applicability of the two-sample t-test to two Exponentially-distributed populations. Lifetimes are frequently distributed as Exponential random variables. Thus, should we want to compare the time until recidivism for two groups of released prisoners, a control group and a treated group, we will be comparing two Exponential random variables.

time until

We know (or will know) that the t-test assumes both populations are distributed Normally. We also know that the Exponential distribution is about as non-Normal as a distribution can be. Thus, we would expect that the t-test would not be applicable. However, the Central Limit Theorem (see Appendix C) tells us that distribution of the sample means converges to the Standard Normal distribution, which is all that the t-test really needs. The only question is: How quickly? Here, we attempt to determine if $n = 30$ is large enough.

One requirement for a test to be appropriate for the problem at hand is for the p-values to be Uniformly distributed. If this is not true, the test is not a fantastic test, and perhaps something else should be used in its stead. **p-values**

Thus, this Monte Carlo experiment will determine if the p-values of a t-test comparing two Exponentially-distributed random variables are Uniformly distributed (between 0 and 1). $\mathcal{U}(0,1)$

```
###  Filename: script2.R
###  Purpose:  application of Monte Carlo to
###    determining if one should use the t-test
###    when both populations are Exponentially
###    distributed and the sample size is n=30.


### Preamble

setwd("F:/RFS/Chapter1/")
set.seed(370)


### The experiment

# Initialize variables
p <- numeric()     ## Vector of p-values
n <- 30            ## Sample size
B <- 1000000       ## Number of iterations

# The loop
for(i in 1:B) {
  x    <- rexp(n, rate=2)       ## Generate X
  y    <- rexp(n, rate=2)       ## Generate Y
  p[i] <- t.test(x,y)$p.value   ## Get p-value
}

# The analysis
hist(p, main="Histogram of p-values", xlab="p-value")
abline(h=B/20, col=4, lty=2)

ks.test(p, punif)
```

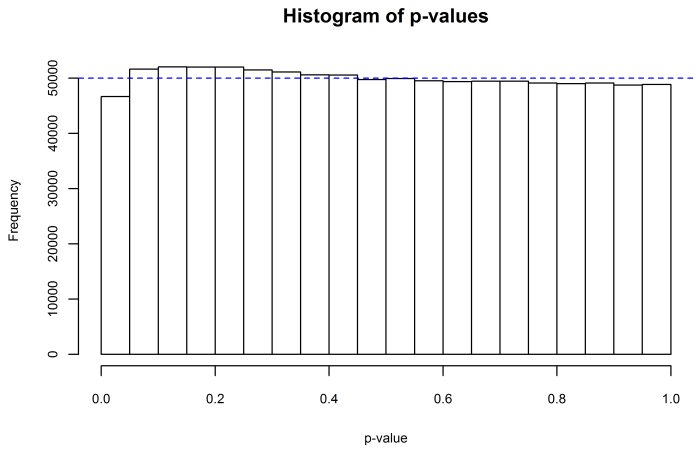

Figure 1.5: *Histogram of p-values from the Monte Carlo experiment. Note that the first bar is lower than one would expect under the uniformity assumption. As such, we can conclude that the test is not appropriate (although it is not as bad as we expected).*

This script starts out the same way as a usual script: Comments describe the script, the working directory is set, and the random-number seed is set. The next block of lines initializes three important variables.

The variable `p` is created to hold all of the calculated p-values from the test. The variable `n` is our sample size of 30. The variable `B` is the number of times we perform the experiment (the number of *trials*). The variable `B` should be as large as possible, remembering that computers are slow. Larger values of `B` produce approximations with better precision. The rule of thumb is that to increase the certainty by one digit, you will have to increase the number of trials by a *factor of* 100. Thus, `B <- 1000000` should give proportion estimates accurate to 3 digits.

trials

The next block is the looping block (Lines 22–26). Inside the loop (lines 23–25) is the experiment. We loop through each experiment `B` times. Each experiment consists of taking two samples of size `n` from an Exponential distribution with rate $\lambda = 2$ (which corresponds to a mean of $\frac{1}{2}$). A two-sample t-test is performed on those two samples, and the calculated p-value is stored in the variable `p` (at position `i`). The loop section is the actual Monte Carlo experiment. Everything else just allows us to use the experiment.

Now that we carried out the experiment, we plot a histogram of the results, plot a horizontal line on the graph, and determine if the histogram is

Uniform. Figure 1.5 is this plot. Note that the histogram is vaguely uniform — except for the first bar, which is much smaller than one would expect. Because of this, I would conclude that the p-values are *not* Uniformly distributed; that is, the test is not appropriate in this case.

If we are not comfortable with testing graphically, we can use the Kolmogorov-Smirnov test. The Kolmogorov-Smirnov test determines if two distributions are significantly different. The null hypothesis is that the two distributions are the same (or that the sample came from the specified distribution). According to the Kolmogorov-Smirnov test, the p-values are not Uniformly distributed ($D = 0.0080; p \ll 0.0001$). As such, we can conclude that we need a sample size larger than $n = 30$ to use the t-test on this data.

K-S Test

It is interesting that the histogram appears to be vaguely Uniform except for the shorter-than-expected first bar. As an extension, you may wish to increase the number of trials to see if that dip goes away.

Do not forget to save this script in your `Chapter1` folder.

1.5: Conclusion

In this chapter, you have learned several advantages to using the `R` statistical environment. You also learned how to download and install `R` on your computer (or USB drive); how to type in two simple, yet informative, scripts; and how to use help and search functions to locate information about statements, commands, or functions in `R`.

1.6: End of Chapter Materials

1.6.1 R Functions

In this chapter, we were introduced to several R functions that will be useful in the future. These are listed here.

Statistics:

IQR(v) This returns the Inter-Quartile Range of the provided vector of values. The IQR is defined as the third quartile less the first quartile: $Q_3 - Q_1$.

ks.test(x,y) This performs a Kolmogorov-Smirnov test, which determines if the two provided samples (`x`,`y`) come from the same distribution. This test is often used to determine if a sample is Normally distributed, in which case `x` will be the data and `y` will be `pnorm`.

length(v) This function returns the number of elements in the vector `v`, usually a vector of data.

max(v) This returns the largest values in the provided vector of values.

mean(v) This returns the arithmetic mean of the provided vector of values. An optional parameter `trim=` allows you to calculate the trimmed mean.

means(v, type) This returns the mean of the provided vector of values. The type of mean returned is specified with the `type=` parameter. Possible values include `"arithmetic"` (the default), `"geometric"`, and `"harmonic"`.

This function must be imported from the Internet using the `source()` function. The parameter `trim=` is also supported (see `mean(v)`, above).

median(v) This returns the median of the provided vector of values.

min(v) This returns the smallest value in the provided vector of values.

quantile(v) This, by default, returns the following values from the provided vector of values: Q_0, Q_1, Q_2, Q_3, and Q_4. The options parameter `probs` allows you to select the percentiles you wish calculated. Thus, the command `quantile(v, c(0.04,0.67,0.99))` returns the 4th, 67th, and 99th percentile in the provided variable v.

sd(v) This returns the sample standard deviation of the provided vector of values. It equals the positive square root of the sample variance.

t.test(·) This function preforms a t-test of the provided data. The four types of t-tests can be specified as

`t.test(x, mu= )`	1-sample t-test
`t.test(x,y)`	2-sample t-test, unequal variances
`t.test(x,y, var.equal=TRUE)`	2-sample t-test, equal variances
`t.test(x,y, paired=TRUE)`	2-sample, paired t-test

var(v) This returns the sample variance of the provided vector of values.

Probability:

set.seed(x) This command specifies the starting random number seed.

rexp(n) This command produces n random numbers drawn from an Exponential distribution, $\mathcal{E}xp(\lambda)$. An optional parameter allows you to specify the `rate` as something other than the default of 1.

runif(n) This command produces n random numbers drawn from a Uniform distribution, $\mathcal{U}(0,1)$. Optional parameters include `min` and `max`, which specify a minimum and a maximum value.

> ***Note***: Appendices A and B cover several important probability distributions.

Graphics:

abline(·) This produces lines on an active plot. Options include producing a horizontal line (`h=`), a vertical line (`v=`), or an oblique line with an intercept (`a=`) and a slope (`b=`).

boxplot(v) This produces a standard box-and-whiskers plot of the provided vector of values. By default, the fences are located at $Q_3 + 1.5IQR$ and $Q_1 - 1.5IQR$. Outliers (those values beyond the fences) are identified with circles on the plot. You can alter both of these features.

hist(v) This produces a histogram of the provided vector of values. Optional parameters allow you to specify the `breaks`, as well as specify that you wish a probability (relative frequency) histogram in lieu of the default frequency histogram.

Mathematics:

+ This is the addition character in `R`.

- This is the subtraction character in `R`.

***** This is the multiplication character in `R`.

/ This is the division character in `R`.

∧ This is the "raise to the power of" character in `R`.

Programming:

getwd() This command tells you the current working directory—the default folder in which the script will be saved and in which the data files will be read.

setwd(p) This command defines the current working directory using path `p`, which can be either absolute or relative to the current working directory. Thus, `setwd(F://RFS/Chapter1/)`, which is an absolute path (the drive letter is specified), will make *that* folder the current working directory, while `setwd(../../RFS/Chapter1/)`, a relative path, will set as the current working directory the folder `Chapter1`, a subfolder of folder `RFS`, which is two levels above your current working directory.

A **Windows™ note**: Windows™ paths usually use the backslash (`\`), which is non-standard. To use Windows paths, either change the backslashes to slashes, or double all or the backslashes. Thus, the following are both acceptable Windows paths:

```
F:/RFS/Chapter1/
F:\\\\RFS\\Chapter1\\
```

I usually use the first form, as it looks better to me, is easier to type, and is more standard across operating systems.

source(p) This command allows you to run external scripts. These external scripts may be functions located locally (on your machine) or remotely (on the Internet).

1.6.2 Exercises and Extensions

This section offers suggestions on things you can practice from just the information in this chapter. Completing these extension exercises requires judicious use of the help and search functions in R, as well as some trial and error — both things I use quite frequently.

For each of the following problems, please save the associated R script in the chapter folder as `ext0x.R`, where `x` is the problem number.

Summary:

1. What do the following R functions do?
   a) `mean(x)`
   b) `sd(x)`
   c) `var(x)`
   d) `median(x)`
   e) `quantile(x,0.25)`
   f) `IQR(x)`
   g) `abline(h=4)`
2. Define replicability. Why is it important in science?
3. What is `csv` format? Give one advantage to saving data in `csv` format.
4. What is a Monte Carlo experiment?

Graphics:

5. Extending the box-and-whiskers plot graphic above (Figure 1.4, left), make the following alterations: Change the y-axis label to "Pipe Length [cm]", the box-and-whiskers plot title to "Box-and-Whiskers Plot of Pipe Lengths", and the orientation of the axis labels to horizontal. [You may want to check the help files on `boxplot()`, `plot()`, and `par()`.]
6. Extending the histogram graphic above (Figure 1.4, right), make the following alterations: Change the x-axis label to "Pipe Length [cm]", the histogram title to "Histogram of Pipe Lengths", and the orientation of the axis labels to horizontal. [You may want to check the help files on `hist()`, `plot()`, and `par()`.]

Monte Carlo:

7. Start a new session in `R`, open a new script, and create a random dataset (of size 100) from the Gaussian distribution, with mean 4 and standard deviation 15 (use `set.seed(370)`). [The Gaussian distribution is also known as the Normal distribution.] Find the mean, median, and variance of the data. Save this script in your chapter folder as `ext01.R`. Explain why the sample standard deviation did not equal 15 and the sample mean did not equal 4.

8. Start a new session in `R`, open a new script, write a script that creates a dataset of size one million ($n = 1000000$), with each element in the dataset being a random number between 0 and 6 (set the seed to 370). [Unless stated otherwise, we assume that 'random' numbers are from a Uniform distribution.] Calculate the mean, median, and third quartile (Q_3) of the data. Save this script in your chapter folder as `ext02.R`.

9. Extending the previous extension, simulate rolling a fair six-sided die by making all of those random numbers integers. [You may want to look up the following functions: `floor()`, `ceiling()`, `round()`, and `sample()` to determine which of them is appropriate and what other changes you may need to make.] Again, set the seed to 370 and calculate the mean, variance, and inter-quartile range. Save this script in your chapter folder as `ext03.R`.

10. Modify `s3MonteCarlo.R` to test if a sample size of $n = 500$ is sufficiently large so that the t-test is appropriate. Save this script in your chapter folder as `ext05.R`.

11. Modify `s3MonteCarlo.R` to test if a sample size of $n = 30$ is sufficiently large so that the t-test is applicable for testing the equality of means between two Gamma distributions. Use a shape parameter of $a = 5$ and a rate parameter of $b = 2$. [Check the `R` help files for `rgamma()`.] Save this script in your chapter folder as `ext06.R`.

    The Gamma distribution is a generalization of the Exponential distribution. As such, it also is good for modelling survival times.

1.6.3 Applied Research

This section offers some applied research works that are connected with the topics in this chapter.

- Walter W. Hudson (1974). "Casework as a Causative Agent in Client Deterioration: A Research Note on the Fischer Assessment." *The Social Service Review* **48**(3): 422–49.
- Eduardo Leoni (2005). "Analyzing Cross-Country Survey Data: Results from Monte Carlo Experiments." Paper presented at the annual meeting of the The Midwest Political Science Association, Palmer House Hilton, Chicago, Illinois (April 7, 2005).
- David L. Lewis and David L. Bodde (1984). "Understanding Political Risk in Investment Planning." *Journal of Policy Analysis and Management* **3**(4): 544–60.
- John Loizides and Efthymios G. Tsionas (2004). "Dynamic Distributions of Productivity Growth in European Railways." *Journal of Transport Economics and Policy* **38**(1): 45–75.
- David A. M. Peterson (2009). "Campaign Learning and Vote Determinants." *American Journal of Political Science* **53**(2): 445–60.
- William B. P. Robson and William M. Scarth (1998). "Federal Debt Reduction: Choosing Paths." *Canadian Public Policy* **24**(3): 356–62.
- Branislav L. Slantchev (2004). "How Initiators End Their Wars: The Duration of Warfare and the Terms of Peace." *American Journal of Political Science* **48**(4): 813–29.

1.6.4 References and Additional Readings

This section provides a list of statistical works. Those works cited in the chapter are here. Also here are works that complement the chapter's topics.

- Emmanuel Paradis (2005). "R for Beginners."
  http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf
- The R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*.
  http://cran.r-project.org/doc/manuals/fullrefman.pdf
- William N. Venables and David M. Smith (2011). *An Introduction to R*.
  http://cran.r-project.org/doc/manuals/R-intro.pdf

❧ ❧ ❧

Many more reference manuals are available for download from:

http://cran.r-project.org/other-docs.html

Avail yourself of them.

❧ ❧ ❧

Furthermore, area experts have come together to ensure that their area is most up-to-date with respect to statistical analysis. The documents that outline the available packages are called "Task Views." Currently, there are 28 task views, ranging from Bayesian analysis to Time Series analysis. The entire list is located at

http://cran.r-project.org/web/views/