Introduction to Probability and Statistics Using R

Third Edition

G. Jay Kerns

2017-02-03

Contents

Chapter 1

An Introduction to Probability and Statistics

This chapter has proved to be the hardest to write, by far. The trouble is that there is so much to say – and so many people have already said it so much better than I could. When I get something I like I will release it here.

In the meantime, there is a lot of information already available to a person with an Internet connection. I recommend to start at Wikipedia, which is not a flawless resource but it has the main ideas with links to reputable sources.

In my lectures I usually tell stories about Fisher, Galton, Gauss, Laplace, Quetelet, and the Chevalier de Mere.

1.1 Probability

The common folklore is that probability has been around for millennia but did not gain the attention of mathematicians until approximately 1654 when the Chevalier de Mere had a question regarding the fair division of a game's payoff to the two players, supposing the game had to end prematurely.

1.2 Statistics

Statistics concerns data; their collection, analysis, and interpretation. In this book we distinguish between two types of statistics: descriptive and inferential.

Descriptive statistics concerns the summarization of data. We have a data set and we would like to describe the data set in multiple ways. Usually this entails calculating numbers from the data, called descriptive measures, such as percentages, sums, averages, and so forth.

Inferential statistics does more. There is an inference associated with the data set, a conclusion drawn about the population from which the data originated.

I would like to mention that there are two schools of thought of statistics: frequentist and bayesian. The difference between the schools is related to how the two groups interpret the underlying probability (see Section {#sec-interpreting-probabilities}). The frequentist school gained a lot of ground among statisticians due in large part to the work of Fisher, Neyman, and Pearson in the early twentieth century. That dominance lasted until inexpensive computing power became widely available; nowadays the bayesian school is garnering more attention and at an increasing rate.

This book is devoted mostly to the frequentist viewpoint because that is how I was trained, with the conspicuous exception of Sections {#sec-bayes-rule} and {#sec-conditional-distributions}. I plan to add more bayesian material in later editions of this book. 3.1

Chapter 2

An Introduction to R

Every R book I have ever seen has had a section/chapter that is an introduction to R, and so does this one. The goal of this chapter is for a person to get up and running, ready for the material that follows. See Section 2.5 for links to other material which the reader may find useful.

What do I want them to know?

- Where to find R to install on a home computer, and a few comments to help with the usual hiccups that occur when installing something.
- Abbreviated remarks about the available options to interact with R.
- Basic operations (arithmetic, entering data, vectors) at the command prompt.
- How and where to find help when they get in trouble.
- Other little shortcuts I am usually asked when introducing R.

2.1 Downloading and Installing R

The instructions for obtaining R largely depend on the user's hardware and operating system. The R Project has written an R Installation and Administration manual with complete, precise instructions about what to do, together with all sorts of additional information. The following is just a primer to get a person started.

2.1.1 Installing R

Visit one of the links below to download the latest version of R for your operating system:

- Microsoft Windows: http://cran.r-project.org/bin/windows/base/
- MacOS: http://cran.r-project.org/bin/macosx/
- Linux: http://cran.r-project.org/bin/linux/

On Microsoft Windows, click the `R-x.y.z.exe` installer to start installation. When it asks for "Customized startup options", specify `Yes`. In the next window, be sure to select the SDI (single document interface) option; this is useful later when we discuss three dimensional plots with the `rgl` package [1].

Installing R on a USB drive (Windows)

With this option you can use R portably and without administrative privileges. There is an entry in the R for Windows FAQ about this. Here is the procedure I use:

1. Download the Windows installer above and start installation as usual. When it asks *where* to install, navigate to the top-level directory of the USB drive instead of the default `C` drive.
2. When it asks whether to modify the Windows registry, uncheck the box; we do NOT want to tamper with the registry.
3. After installation, change the name of the folder from `R-x.y.z` to just plain R. (Even quicker: do this in step 1.)
4. Download this shortcut and move it to the top-level directory of the USB drive, right beside the R folder, not inside the folder. Use the downloaded shortcut to run R.

Steps 3 and 4 are not required but save you the trouble of navigating to the `R-x.y.z/bin` directory to double-click `Rgui.exe` every time you want to run the program. It is useless to create your own shortcut to `Rgui.exe`. Windows does not allow shortcuts to have relative paths; they always have a drive letter associated with them. So if you make your own shortcut and plug your USB drive into some *other* machine that happens to assign your drive a different letter, then your shortcut will no longer be pointing to the right place.

2.1.2 Installing and Loading Add-on Packages

There are *base* packages (which come with R automatically), and *contributed* packages (which must be downloaded for installation). For example, on the version of R being used for this document the default base packages loaded at startup are

```
getOption("defaultPackages")
```

```
## [1] "datasets"  "utils"     "grDevices" "graphics"  "stats"     "methods"
```

The base packages are maintained by a select group of volunteers, called R Core. In addition to the base packages, there are over ten thousand additional contributed packages written by individuals all over the world. These are stored worldwide on mirrors of the Comprehensive R Archive Network, or `CRAN` for short. Given an active Internet connection, anybody is free to download and install these packages and even inspect the source code.

To install a package named `foo`, open up R and type `install.packages("foo")` . To install `foo` and additionally install all of the other packages on which `foo` depends, instead type `install.packages("foo", depends = TRUE)`.

The general command `install.packages()` will (on most operating systems) open a window containing a huge list of available packages; simply choose one or more to install.

No matter how many packages are installed onto the system, each one must first be loaded for use with the `library` function. For instance, the `foreign` package [105] contains all sorts of functions needed to import data sets into R from other software such as SPSS, SAS, *etc*. But none of those functions will be available until the command `library("foreign")` is issued.

Type `library()` at the command prompt (described below) to see a list of all available packages in your library.

For complete, precise information regarding installation of R and add-on packages, see the R Installation and Administration manual.

2.2 Communicating with R

2.2.1 One line at a time

This is the most basic method and is the first one that beginners will use.

- RGui (Microsoft ® Windows)
- RStudio
- Terminal
- Emacs/ESS, XEmacs

2.2.2 Multiple lines at a time

For longer programs (called *scripts*) there is too much code to write all at once at the command prompt. Furthermore, for longer scripts it is convenient to be able to only modify a certain piece of the script and run it again in R. Programs called *script editors* are specially designed to aid the communication and code writing process. They have all sorts of helpful features including R syntax highlighting, automatic code completion, delimiter matching, and dynamic help on the R functions as they are being written. Even more, they often have all of the text editing features of programs like Microsoft®Word. Lastly, most script editors are fully customizable in the sense that the user can customize the appearance of the interface to choose what colors to display, when to display them, and how to display them.

- **R Editor (Windows):** In Microsoft® Windows, R Gui has its own built-in script editor, called R Editor. From the console window, select `File` ▹ `New Script`. A script window opens, and the lines of code can be written in the window. When satisfied with the code, the user highlights all of the commands and presses Ctrl+R. The commands are automatically run at once in R and the output is shown. To save the script for later, click `File` ▹ `Save as...` in R Editor. The script can be reopened later with `File` ▹} `Open Script...` in `RGui`. Note that R Editor does not have the fancy syntax highlighting that the others do.
- **RStudio:**
- **Emacs/ESS:** Emacs is an all purpose text editor. It can do absolutely anything with respect to modifying, searching, editing, and manipulating, text. And if Emacs can't do it, then you can write a program that extends Emacs to do it. Once such extension is called `ESS`, which stands

for /E/-macs /S/-peaks /S/-tatistics. With ESS a person can speak to R, do all of the tricks that the other script editors offer, and much, much, more. Please see the following for installation details, documentation, reference cards, and a whole lot more: http://ess.r-project.org. *Fair warning*: if you want to try Emacs and if you grew up with Microsoft® Windows or Macintosh, then you are going to need to relearn everything you thought you knew about computers your whole life. (Or, since Emacs is completely customizable, you can reconfigure Emacs to behave the way you want.) I have personally experienced this transformation and I will never go back.

2.2.3 Graphical User Interfaces (GUIs)

By the word "GUI" I mean an interface in which the user communicates with R by way of points-and-clicks in a menu of some sort. Again, there are many, many options and I only mention one that I have used and enjoyed.

- **R Commander** provides a point-and-click interface to many basic statistical tasks. It is called the "Commander" because every time one makes a selection from the menus, the code corresponding to the task is listed in the output window. One can take this code, copy-and-paste it to a text file, then re-run it again at a later time without the R Commander's assistance. It is well suited for the introductory level. `Rcmdr` [45] also allows for user-contributed "Plugins" which are separate packages on `CRAN` that add extra functionality to the `Rcmdr` package. The plugins are typically named with the prefix `RcmdrPlugin` to make them easy to identify in the `CRAN` package list. One such plugin is the `RcmdrPlugin.IPSUR` package [152] which accompanies this text.

2.3 Basic R Operations and Concepts

The R developers have written an introductory document entitled "An Introduction to R". There is a sample session included which shows what basic interaction with R looks like. I recommend that all new users of R read that document, but bear in mind that there are concepts mentioned which will be unfamiliar to the beginner.

Below are some of the most basic operations that can be done with R. Almost every book about R begins with a section like the one below; look around to see all sorts of things that can be done at this most basic level.

2.3.1 Arithmetic

```
2 + 3       # add
```

```
## [1] 5
```

```
4 # 5 / 6   # multiply and divide
```

```
## [1] 4
```

```
7^8          # 7 to the 8th power
```

```
## [1] 5764801
```

Notice the comment character #. Anything typed after a # symbol is ignored by R. We know that 20/6 is a repeating decimal, but the above example shows only 7 digits. We can change the number of digits displayed with `options`:

```
options(digits = 16)
10/3                  # see more digits
```

```
## [1] 3.333333333333333
```

```
sqrt(2)               # square root
```

```
## [1] 1.414213562373095
```

```
exp(1)                # Euler's constant, e
```

```
## [1] 2.718281828459045
```

```
pi
```

```
## [1] 3.141592653589793
```

```
options(digits = 7)  # back to default
```

Note that it is possible to set `digits` up to 22, but setting them over 16 is not recommended (the extra significant digits are not necessarily reliable). Above notice the `sqrt` function for square roots and the `exp` function for powers of e, Euler's number.

2.3.2 Assignment, Object names, and Data types

It is often convenient to assign numbers and values to variables (objects) to be used later. The proper way to assign values to a variable is with the <- operator (with a space on either side). The = symbol works too, but it is recommended by the R masters to reserve = for specifying arguments to functions (discussed later). In this book we will follow their advice and use <- for assignment. Once a variable is assigned, its value can be printed by simply entering the variable name by itself.

```
x <- 7*41/pi   # don't see the calculated value
x              # take a look
```

```
## [1] 91.35494
```

When choosing a variable name you can use letters, numbers, dots "`.`", or underscore "`_`" characters. You cannot use mathematical operators, and a leading dot may not be followed by a number. Examples of valid names are: `x`, `x1`, `y.value`, and `!y_hat`. (More precisely, the set of allowable characters in object names depends on one's particular system and locale; see An Introduction to R for more discussion on this.)

Objects can be of many *types*, *modes*, and *classes*. At this level, it is not necessary to investigate all of the intricacies of the respective types, but there are some with which you need to become familiar: * integer: the values 0, ±1, ±2, ...; these are represented exactly by R. * double: real numbers (rational and irrational); these numbers are not represented exactly (save integers or fractions with a denominator that is a power of 2, see Venables and Smith [164]). * character: elements that are wrapped with pairs of " or '; * logical: includes `TRUE`, `FALSE`, and `NA` (which are reserved words); the `NA` stands for "not available", *i.e.*, a missing value.

You can determine an object's type with the `typeof` function. In addition to the above, there is the `complex` data type:

```
sqrt(-1)              # isn't defined
```

```
## Warning in sqrt(-1): NaNs produced
```

```
## [1] NaN
```

```
sqrt(-1+0i)           # is defined
```

```
## [1] 0+1i
```

```
sqrt(as.complex(-1))  # same thing
```

```
## [1] 0+1i
```

```
(0 + 1i)^2            # should be -1
```

```
## [1] -1+0i
```

```
typeof((0 + 1i)^2)
```

```
## [1] "complex"
```

Note that you can just type `(1i)^2` to get the same answer. The `NaN` stands for "not a number"; it is represented internally as `double` .

2.3.3 Vectors

:PROPERTIES: :CUSTOM_ID: sub-Vectors :END:

All of this time we have been manipulating vectors of length 1. Now let us move to vectors with multiple entries.

Entering data vectors

The long way: If you would like to enter the data `74,31,95,61,76,34,23,54,96` into R, you may create a data vector with the `c` function (which is short for *concatenate*).

```
x <- c(74, 31, 95, 61, 76, 34, 23, 54, 96)
x
```

```
## [1] 74 31 95 61 76 34 23 54 96
```

The elements of a vector are usually coerced by R to the the most general type of any of the elements, so if you do `c(1, "2")` then the result will be `c("1", "2")`.

A shorter way: : The `scan` method is useful when the data are stored somewhere else. For instance, you may type `x <- scan()` at the command prompt and R will display `1:` to indicate that it is waiting for the first data value. Type a value and press `Enter`, at which point R will display `2:`, and so forth. Note that entering an empty line stops the scan. This method is especially handy when you have a column of values, say, stored in a text file or spreadsheet. You may copy and paste them all at the `1:` prompt, and R will store all of the values instantly in the vector `x`.

*Repeated data; regular patterns:# the `seq` function will generate all sorts of sequences of numbers. It has the arguments `from`, `to`, `by`, and `length.out` which can be set in concert with one another. We will do a couple of examples to show you how it works.

```
seq(from = 1, to = 5)
```

```
## [1] 1 2 3 4 5
```

```
seq(from = 2, by = -0.1, length.out = 4)
```

```
## [1] 2.0 1.9 1.8 1.7
```

Note that we can get the first line much quicker with the colon operator.

```
1:5
```

```
## [1] 1 2 3 4 5
```

The vector LETTERS has the 26 letters of the English alphabet in uppercase and `letters` has all of them in lowercase.

Indexing data vectors

Sometimes we do not want the whole vector, but just a piece of it. We can access the intermediate parts with the `[]` operator. Observe (with `x` defined above)

```
x[1]
```

```
## [1] 74
```

```
x[2:4]
```

```
## [1] 31 95 61
```

```
x[c(1,3,4,8)]
```

```
## [1] 74 95 61 54
```

```
x[-c(1,3,4,8)]
```

```
## [1] 31 76 34 23 96
```

Notice that we used the minus sign to specify those elements that we do *not* want.

```
LETTERS[1:5]
```

```
## [1] "A" "B" "C" "D" "E"
```

```
letters[-(6:24)]
```

```
## [1] "a" "b" "c" "d" "e" "y" "z"
```

2.3.4 Functions and Expressions

A function takes arguments as input and returns an object as output. There are functions to do all sorts of things. We show some examples below.

```
x <- 1:5
sum(x)
```

```
## [1] 15
```

```
length(x)
```

```
## [1] 5
```

```
min(x)
```

```
## [1] 1
```

```
mean(x)      # sample mean
```

```
## [1] 3
```

```
sd(x)        # sample standard deviation
```

```
## [1] 1.581139
```

It will not be long before the user starts to wonder how a particular function is doing its job, and since R is open-source, anybody is free to look under the hood of a function to see how things are calculated. For detailed instructions see the article "Accessing the Sources" by Uwe Ligges [86]. In short:

Type the name of the function without any parentheses or arguments. If you are lucky then the code for the entire function will be printed, right there looking at you. For instance, suppose that we would like to see how the `intersect` function works:

```
intersect
```

```
## function (x, y)
## {
##     y <- as.vector(y)
##     unique(y[match(as.vector(x), y, 0L)])
## }
## <bytecode: 0x4109230>
## <environment: namespace:base>
```

If instead it shows `UseMethod(something)` then you will need to choose the *class* of the object to be inputted and next look at the *method* that will be *dispatched* to the object. For instance, typing `rev` says

```
rev
```

```
## function (x)
## UseMethod("rev")
## <bytecode: 0x386be40>
## <environment: namespace:base>
```

The output is telling us that there are multiple methods associated with the `rev` function. To see what these are, type

```
methods(rev)
```

```
## [1] rev.default     rev.dendrogram*
## see '?methods' for accessing help and source code
```

Now we learn that there are two different `rev(x)` functions, only one of which being chosen at each call depending on what `x` is. There is one for `dendrogram` objects and a `default` method for everything else. Simply type the name to see what each method does. For example, the `default` method can be viewed with

```
rev.default
```

```
## function (x)
## if (length(x)) x[length(x):1L] else x
## <bytecode: 0x3d94768>
## <environment: namespace:base>
```

Some functions are hidden by a namespace (see An Introduction to R Venables and Smith [164]), and are not visible on the first try. For example, if we try to look at the code for `wilcox.test` (see Chapter **??**) we get the following:

```
wilcox.test
```

```
## function (x, ...)
## UseMethod("wilcox.test")
## <bytecode: 0x3749df0>
## <environment: namespace:stats>
```

```
methods(wilcox.test)
```

```
## [1] wilcox.test.default* wilcox.test.formula*
## see '?methods' for accessing help and source code
```

If we were to try `wilcox.test.default` we would get a “not found” error, because it is hidden behind the namespace for the package `stats` [115] (shown in the last line when we tried `wilcox.test`). In cases like these we prefix the package name to the front of the function name with three colons; the command `stats:::wilcox.test.default` will show the source code, omitted here for brevity.

If it shows `.Internal(something)` or `.Primitive(something)` , then it will be necessary to download the source code of R (which is *not* a binary version with an `.exe` extension) and search inside the code there. See Ligges [86] for more discussion on this. An example is `exp`:

```
exp
```

```
## function (x)  .Primitive("exp")
```

Be warned that most of the `.Internal` functions are written in other computer languages which the beginner may not understand, at least initially.

2.4 Getting Help

When you are using R, it will not take long before you find yourself needing help. Fortunately, R has extensive help resources and you should immediately become familiar with them. Begin by clicking `Help` on `RGui`. The following options are available.

- Console: gives useful shortcuts, for instance, `Ctrl+L`, to clear the R console screen.
- FAQ on R: frequently asked questions concerning general R operation.
- FAQ on R for Windows: frequently asked questions about R, tailored to the Microsoft Windows operating system.
- Manuals: technical manuals about all features of the R system including installation, the complete language definition, and add-on packages.
- R functions (text)...: use this if you know the *exact* name of the function you want to know more about, for example, `mean` or `plot`. Typing `mean` in the window is equivalent to typing `help("mean")` at the command line, or more simply, `?mean` . Note that this method only works if the function of interest is contained in a package that is already loaded into the search path with `library`.
- HTML Help: use this to browse the manuals with point-and-click links. It also has a Search Engine & Keywords for searching the help page titles, with point-and-click links for the search results. This is possibly the best help method for beginners. It can be started from the command line with the command `help.start()` .
- Search help ...: use this if you do not know the exact name of the function of interest, or if the function is in a package that has not been loaded yet. For example, you may enter `plo` and a text window will return listing all the help files with an alias, concept, or title matching `plo` using regular expression matching; it is equivalent to typing `help.search("plo")` at the command line. The advantage is that you do not need to know the exact name of the function; the disadvantage is that you cannot point-and-click the results. Therefore, one may wish to use the HTML Help search engine instead. An equivalent way is `??plo` at the command line.
- search.r-project.org ...: this will search for words in help lists and email archives of the R Project. It can be very useful for finding other questions that other users have asked.
- Apropos ...: use this for more sophisticated partial name matching of functions. See `?apropos` for details.

On the help pages for a function there are sometimes "Examples" listed at the bottom of the page, which will work if copy-pasted at the command line (unless marked otherwise). The `example` function will run the code automatically, skipping the intermediate step. For instance, we may try `example(mean)` to see a few examples of how the `mean` function works.

2.4.1 R Help Mailing Lists

There are several mailing lists associated with R, and there is a huge community of people that read and answer questions related to R. See here for an idea of what is available. Particularly pay attention to the bottom of the page which lists several special interest groups (SIGs) related to R.

Bear in mind that R is free software, which means that it was written by volunteers, and the people that frequent the mailing lists are also volunteers who are not paid by customer support fees. Consequently, if you want to use the mailing lists for free advice then you must adhere to some basic etiquette, or else you may not get a reply, or even worse, you may receive a reply which is a bit less cordial than you are used to. Below are a few considerations:

1. Read the FAQ. Note that there are different FAQs for different operating systems. You should read these now, even without a question at the moment, to learn a lot about the idiosyncrasies of R.
2. Search the archives. Even if your question is not a FAQ, there is a very high likelihood that your question has been asked before on the mailing list. If you want to know about topic `foo`, then you can do `RSiteSearch("foo")` to search the mailing list archives (and the online help) for it.
3. Do a Google search and an `RSeek.org` search.

If your question is not a FAQ, has not been asked on R-help before, and does not yield to a Google (or alternative) search, then, and only then, should you even consider writing to R-help. Below are a few additional considerations.

- Read the posting guide before posting. This will save you a lot of trouble and pain.
- Get rid of the command prompts (>) from output. Readers of your message will take the text from your mail and copy-paste into an R session. If you make the readers' job easier then it will increase the likelihood of a response.
- Questions are often related to a specific data set, and the best way to communicate the data is with a `dump` command. For instance, if your question involves data stored in a vector `x`, you can type `dump("x","")` at the command prompt and copy-paste the output into the body of your email message. Then the reader may easily copy-paste the message from your email into R and `x` will be available to him/her.
- Sometimes the answer the question is related to the operating system used, the attached packages, or the exact version of R being used. The `sessionInfo()` command collects all of this information to be copy-pasted into an email (and the Posting Guide requests this information). See Appendix **??** for an example.

2.5 External Resources

There is a mountain of information on the Internet about R. Below are a few of the important ones.

- The R- Project for Statistical Computing : Go there first.
- The Comprehensive R Archive Network : That is where R is stored along with thousands of contributed packages. There are also loads of contributed information (books, tutorials, *etc.*). There are mirrors all over the world with duplicate information.
- R-Forge : This is another location where R packages are stored. Here you can find development code which has not yet been released to `CRAN`.
- R Seek is a search engine based on Google specifically tailored for R queries.

2.6 Other Tips

It is unnecessary to retype commands repeatedly, since R remembers what you have recently entered on the command line. On the Microsoft® Windows R Gui, to cycle through the previous commands just push the ↑ (up arrow) key. On Emacs/ESS the command is `M-p` (which means hold down the `Alt` button and press "p"). More generally, the command `history()` will show a whole list of recently entered commands.

- To find out what all variables are in the current work environment, use the commands `objects()` or `ls()` . These list all available objects in the workspace. If you wish to remove one or more variables, use `remove(var1, var2, var3)` , or more simply use `rm(var1, var2, var3)`, and to remove all objects use `rm(list = ls())`.
- Another use of `scan` is when you have a long list of numbers (separated by spaces or on different lines) already typed somewhere else, say in a text file To enter all the data in one fell swoop, first highlight and copy the list of numbers to the Clipboard with `Edit` ▹ `Copy` (or by right-clicking and selecting `Copy`). Next type the `x <- scan()` command in the R console, and paste the numbers at the `1:` prompt with `Edit` ▹ `Paste`. All of the numbers will automatically be entered into the vector `x`.
- The command `Ctrl+l` clears the display in the Microsoft® Windows R Gui. In Emacs/ESS, press `Ctrl+l` repeatedly to cycle point (the place where the cursor is) to the bottom, middle, and top of the display.
- Once you use R for awhile there may be some commands that you wish to run automatically whenever R starts. These commands may be saved in a file called `Rprofile.site` which is usually in the `etc` folder, which lives in the R home directory (which on Microsoft® Windows usually is `C:\Program Files\R`). Alternatively, you can make a file `.Rprofile` to be stored in the user's home directory, or anywhere R is invoked. This allows for multiple configurations for different projects or users. See "Customizing the Environment" of *An Introduction to R* for more details.
- When exiting R the user is given the option to "save the workspace". I recommend that beginners DO NOT save the workspace when quitting. If `Yes` is selected, then all of the objects and data currently in R's memory is saved in a file located in the working directory called `.RData` . This file is then automatically loaded the next time R starts (in which case

R will say `[previously saved workspace   restored]`). This is a valuable feature for experienced users of R, but I find that it causes more trouble than it saves with beginners.

2.7 Exercises

Chapter 3

Data Description

In this chapter we introduce the different types of data that a statistician is likely to encounter, and in each subsection we give some examples of how to display the data of that particular type. Once we see how to display data distributions, we next introduce the basic properties of data distributions. We qualitatively explore several data sets. Once that we have intuitive properties of data sets, we next discuss how we may numerically measure and describe those properties with descriptive statistics.

What do I want them to know?

- different data types, such as quantitative versus qualitative, nominal versus ordinal, and discrete versus continuous
- basic graphical displays for assorted data types, and some of their (dis)advantages
- fundamental properties of data distributions, including center, spread, shape, and crazy observations
- methods to describe data (visually/numerically) with respect to the properties, and how the methods differ depending on the data type
- all of the above in the context of grouped data, and in particular, the concept of a factor

3.1 Types of Data

Loosely speaking, a datum is any piece of collected information, and a data set is a collection of data related to each other in some way. We will categorize data into five types and describe each in turn:

- **Quantitative:** data associated with a measurement of some quantity on an observational unit,
- **Qualitative:** data associated with some quality or property of an observational unit,
- **Logical:** data which represent true or false and play an important role later,
- **Missing:** data which should be there but are not, and
- **Other types:** everything else under the sun.

In each subsection we look at some examples of the type in question and introduce methods to display them.

3.1.1 Quantitative data

Quantitative data are any data that measure or are associated with a measurement of the quantity of something. They invariably assume numerical values. Quantitative data can be further subdivided into two categories. * *Discrete data* take values in a finite or countably infinite set of numbers, that is, all possible values could (at least in principle) be written down in an ordered list. Examples include: counts, number of arrivals, or number of successes. They are often represented by integers, say, 0, 1, 2, *etc*. * *Continuous data* take values in an interval of numbers. These are also known as scale data, interval data, or measurement data. Examples include: height, weight, length, time, *etc*. Continuous data are often characterized by fractions or decimals: 3.82, 7.0001, $4\frac{5}{8}$, *etc*.

Note that the distinction between discrete and continuous data is not always clear-cut. Sometimes it is convenient to treat data as if they were continuous, even though strictly speaking they are not continuous. See the examples.

Example 3.1 (Annual Precipitation in US Cities)**.** The vector `precip` contains average amount of rainfall (in inches) for each of 70 cities in the United States and Puerto Rico. Let us take a look at the data:

```
str(precip)
```

```
 Named num [1:70] 67 54.7 7 48.5 14 17.2 20.7 13 43.4 40.2 ...
 - attr(*, "names")= chr [1:70] "Mobile" "Juneau" "Phoenix" "Little Rock" ...
```

```
precip[1:4]
```

```
     Mobile      Juneau     Phoenix Little Rock 
       67.0        54.7         7.0        48.5 
```

The output shows that `precip` is a numeric vector which has been *named*, that is, each value has a name associated with it (which can be set with the `names` function). These are quantitative continuous data.

Example 3.2 (Lengths of Major North American Rivers)**.** The U.S. Geological Survey recorded the lengths (in miles) of several rivers in North America. They are stored in the vector `rivers` in the `datasets` package [108] (which ships with base R). See `?rivers`. Let us take a look at the data with the `str` function.

```
str(rivers)
```

```
 num [1:141] 735 320 325 392 524 ...
```

The output says that `rivers` is a numeric vector of length 141, and the first few values are 735, 320, 325, *etc*. These data are definitely quantitative and it appears that the measurements have been rounded to the nearest mile. Thus, strictly speaking, these are discrete data. But we will find it convenient later to take data like these to be continuous for some of our statistical procedures.

Example 3.3 (Yearly Numbers of Important Discoveries)**.** The vector `discoveries` contains numbers of "great" inventions/discoveries in each year from 1860 to 1959, as reported by the 1975 World Almanac. Let us take a look at the data:

```r
str(discoveries)
```

```
 Time-Series [1:100] from 1860 to 1959: 5 3 0 2 0 3 2 3 6 1 ...
```

The output is telling us that `discoveries` is a *time series* (see Section 3.1.7 for more) of length 100. The entries are integers, and since they represent counts this is a good example of discrete quantitative data. We will take a closer look in the following sections.

3.1.2 Displaying Quantitative Data

One of the first things to do when confronted by quantitative data (or any data, for that matter) is to make some sort of visual display to gain some insight into the data's structure. There are almost as many display types from which to choose as there are data sets to plot. We describe some of the more popular alternatives.

Strip charts also known as Dot plots

These can be used for discrete or continuous data, and usually look best when the data set is not too large. Along the horizontal axis is a numerical scale above which the data values are plotted. We can do it in R with a call to the `stripchart` function. There are three available methods.

- `overplot`: plots ties covering each other. This method is good to display only the distinct values assumed by the data set.
- `jitter`: adds some noise to the data in the y direction in which case the data values are not covered up by ties.
- `stack`: plots repeated values stacked on top of one another. This method is best used for discrete data with a lot of ties; if there are no repeats then this method is identical to overplot.

See Figure **??**, which was produced by the following code.

```r
stripchart(precip, xlab="rainfall")
stripchart(rivers, method="jitter", xlab="length")
stripchart(discoveries, method="stack", xlab="number")
```

The leftmost graph is a strip chart of the `precip` data. The graph shows tightly clustered values in the middle with some others falling balanced on either side, with perhaps slightly more falling to the left. Later we will call this a symmetric distribution, see Section 3.2.3. The middle graph is of the `rivers` data, a vector of length 141. There are several repeated values in the rivers data, and if we were to use the overplot method we would lose some of them in the display. This plot shows a what we will later call a right-skewed shape with perhaps some extreme values on the far right of the display. The third graph strip charts `discoveries` data which are literally a textbook example of a right skewed distribution.

The `DOTplot` function in the `UsingR` package [166] is another alternative.

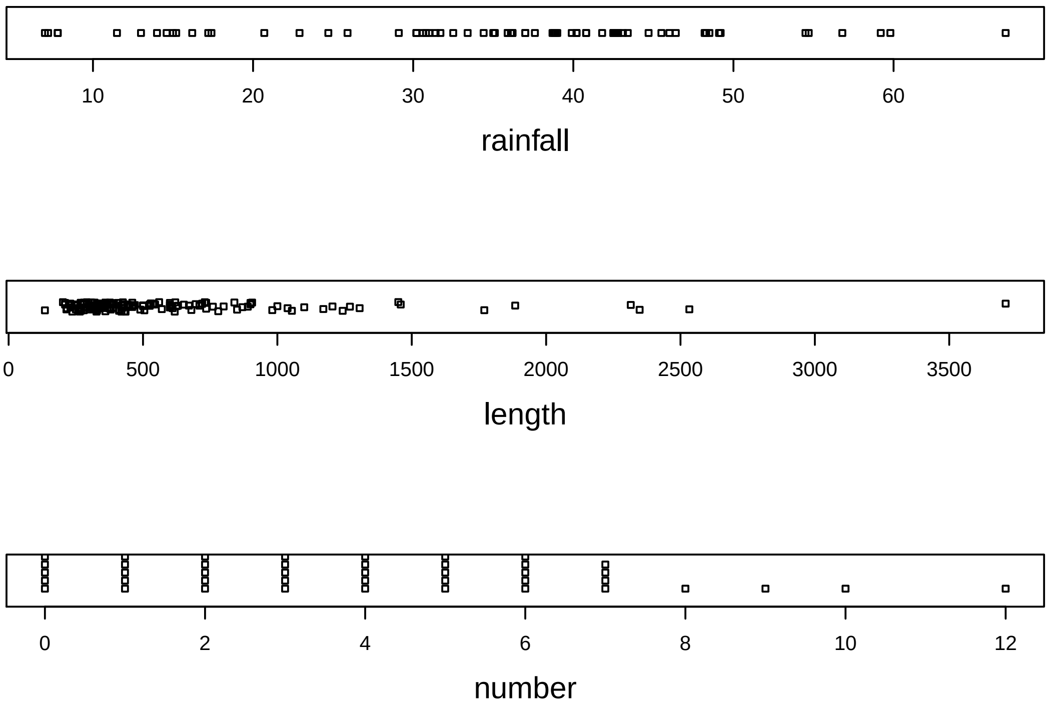


Figure 3.1: Three stripcharts of three data sets. The first graph uses the `overplot` method, the second the `jitter` method, and the third the `stack` method.

Histogram

These are typically used for continuous data. A histogram is constructed by first deciding on a set of classes, or bins, which partition the real line into a set of boxes into which the data values fall. Then vertical bars are drawn over the bins with height proportional to the number of observations that fell into the bin.

These are one of the most common summary displays, and they are often misidentified as "Bar Graphs" (see below.) The scale on the y axis can be frequency, percentage, or density (relative frequency). The term histogram was coined by Karl Pearson in 1891, see [98].

Example 3.4 (Annual Precipitation in US Cities)**.** We are going to take another look at the `precip` data that we investigated earlier. The strip chart in Figure 3.1 suggested a loosely balanced distribution; let us now look to see what a histogram says.

There are many ways to plot histograms in R, and one of the easiest is with the `hist` function. The following code produces the plots in Figure **??**.

```
hist(precip, main = "")
hist(precip, freq = FALSE, main = "")
```

Notice the argument `main` = "" which suppresses the main title from being displayed – it would have said "Histogram of `precip`" otherwise. The plot on the left is a frequency histogram (the default), and the plot on the right is a relative frequency histogram (`freq = FALSE`).

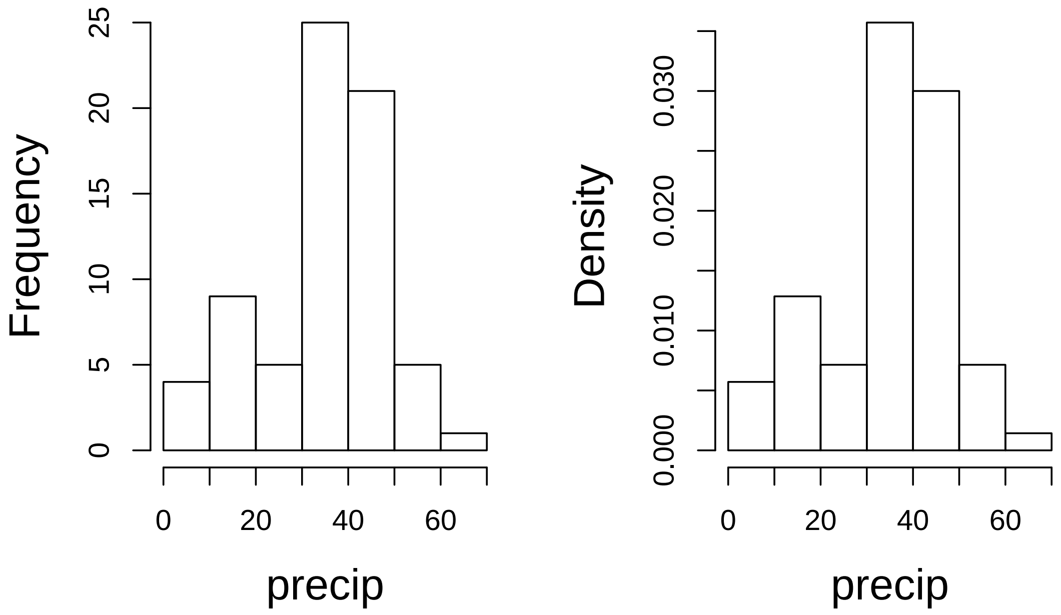


Figure 3.2: (Relative) frequency histograms of the `precip` data.

Please mind the biggest weakness of histograms: the graph obtained strongly depends on the bins chosen. Choose another set of bins, and you will get a different histogram. Moreover, there are not any definitive criteria by which bins should be defined; the best choice for a given data set is the one which illuminates the data set's underlying structure (if any). Luckily for us there are algorithms to automatically choose bins that are likely to display well, and more often than not the default bins do a good job. This is not always the case, however, and a responsible statistician will investigate many bin choices to test the stability of the display.

Recall that the strip chart in Figure 3.1 suggested a relatively balanced shape to the `precip` data distribution. Watch what happens when we change the bins slightly (with the `breaks` argument to `hist`). See Figure 3.3 which was produced by the following code.

```
hist(precip, breaks = 10)
hist(precip, breaks = 25)
hist(precip, breaks = 50)
```

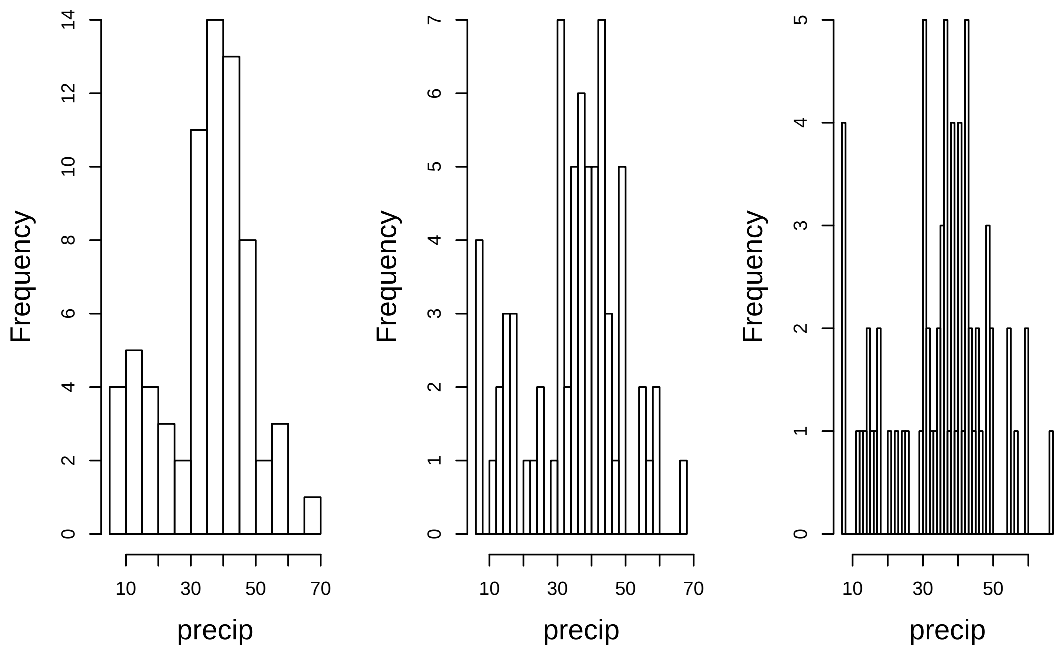


Figure 3.3: More histograms of the `precip` data.

The leftmost graph (with `breaks = 10`) shows that the distribution is not balanced at all. There are two humps: a big one in the middle and a smaller one to the left. Graphs like this often indicate some underlying group structure to the data; we could now investigate whether the cities for which rainfall was measured were similar in some way, with respect to geographic region, for example.

The rightmost graph in Figure 3.3 shows what happens when the number of bins is too large: the histogram is too grainy and hides the rounded appearance of the earlier histograms. If we were to continue increasing the number of bins we would eventually get all observed bins to have exactly one element, which is nothing more than a glorified strip chart.

Stem-and-leaf displays (more to be said in Section 3.4)

Stem-and-leaf displays (also known as stemplots) have two basic parts: *stems* and *leaves*. The final digit of the data values is taken to be a *leaf*, and the leading digit(s) is (are) taken to be *stems*. We draw a vertical line, and to the left of the line we list the stems. To the right of the line, we list the leaves beside their corresponding stem. There will typically be several leaves for each stem, in which case the leaves accumulate to the right. It is sometimes necessary to round the data values, especially for larger data sets.

Example 3.5 (Driver Deaths in the United Kingdom)**.** `UKDriverDeaths` is a time series that contains the total car drivers killed or seriously injured in Great Britain monthly from Jan 1969 to Dec 1984. See `?UKDriverDeaths`. Compulsory seat belt use was introduced on January 31, 1983. We construct a stem and leaf diagram in R with the `stem.leaf` function from the `aplpack` package [178].

```r
stem.leaf(UKDriverDeaths, depth = FALSE)
```

```
1 | 2: represents 120
 leaf unit: 10
            n: 192
   10 | 57
   11 | 136678
   12 | 123889
   13 | 0255666888899
   14 | 000012223444445555566667788889
   15 | 0000111112222223444455555566677779
   16 | 012222334444455555555678888889
   17 | 11233344566667799
   18 | 00011235568
   19 | 01234455667799
   20 | 0000113557788899
   21 | 145599
   22 | 013467
   23 | 9
   24 | 7
HI: 2654
```

The display shows a more or less balanced mound-shaped distribution, with one or maybe two humps, a big one and a smaller one just to its right. Note that the data have been rounded to the tens place so that each datum gets only one leaf to the right of the dividing line.

Notice that the `depths` have been suppressed. To learn more about this option and many others, see Section 3.4. Unlike a histogram, the original data values may be recovered from the stem-and-leaf display – modulo the rounding – that is, starting from the top and working down we can read off the data values 1050, 1070, 1110, 1130, and so forth.

Index plots

Done with the `plot` function. These are good for plotting data which are ordered, for example, when the data are measured over time. That is, the first observation was measured at time 1, the second at time 2, *etc*. It is a two dimensional plot, in which the index (or time) is the x variable and the measured value is the y variable. There are several plotting methods for index plots, and we mention two of them:

- `spikes`: draws a vertical line from the x-axis to the observation height.
- `points`: plots a simple point at the observation height.

Example 3.6 (Level of Lake Huron 1875-1972)**.** Brockwell and Davis [17] give the annual measurements of the level (in feet) of Lake Huron from 1875–1972. The data are stored in the time series `LakeHuron`. See `?LakeHuron`. Figure **??** was produced with the following code:

```
plot(LakeHuron)
plot(LakeHuron, type = "p")
plot(LakeHuron, type = "h")
```

The plots show an overall decreasing trend to the observations, and there appears to be some seasonal variation that increases over time.

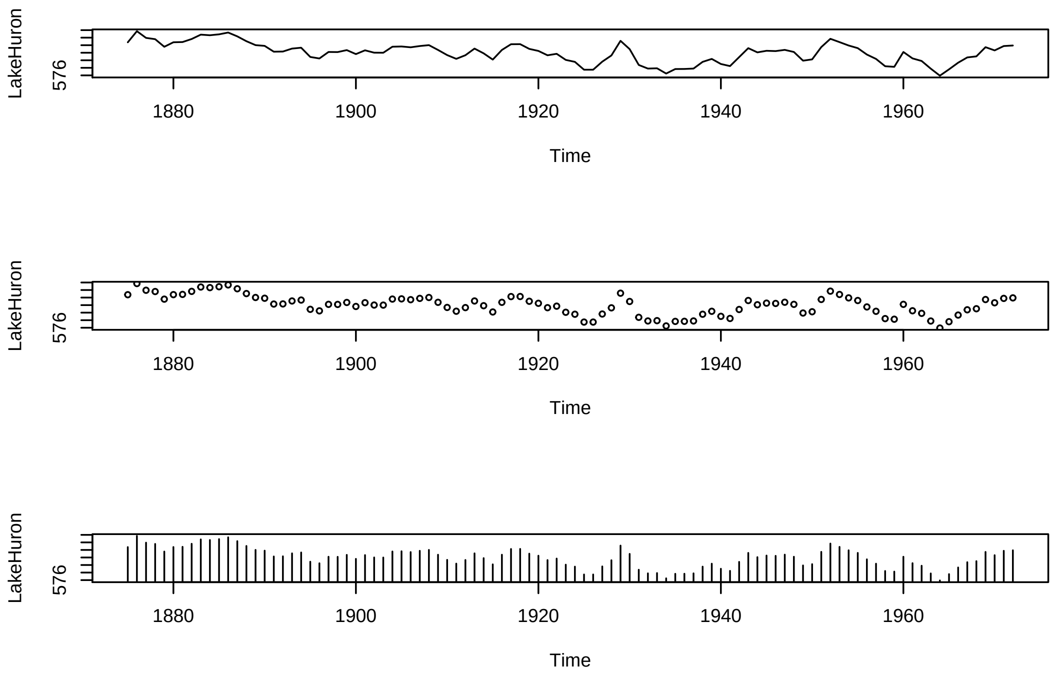


Figure 3.4: Index plots of the `LakeHuron` data.

Density estimates

The default method uses a Gaussian kernel density estimate.

```r
# The Old Faithful geyser data
d <- density(faithful$eruptions, bw = "sj")
d
plot(d)
hist(precip, freq = FALSE)
lines(density(precip))
```

3.1.3 Qualitative Data, Categorical Data, and Factors

Qualitative data are simply any type of data that are not numerical, or do not represent numerical quantities. Examples of qualitative variables include a subject's name, gender, race/ethnicity, political party, socioeconomic status, class rank, driver's license number, and social security number (SSN).

Please bear in mind that some data *look* to be quantitative but are *not*, because they do not represent numerical quantities and do not obey mathematical rules. For example, a person's shoe size is typically written with numbers: 8, or 9, or 12, or $12\frac{1}{2}$. Shoe size is not quantitative, however, because if we take a size 8 and combine with a size 9 we do not get a size 17.

Some qualitative data serve merely to *identify* the observation (such a subject's name, driver's license number, or SSN). This type of data does not usually play much of a role in statistics. But other qualitative variables serve to *subdivide* the data set into categories; we call these *factors*. In the above examples, gender, race, political party, and socioeconomic status would be considered factors (shoe size would be another one). The possible values of a factor are called its *levels*. For instance, the factor of gender would have two levels, namely, male and female. Socioeconomic status typically has three levels: high, middle, and low.

Factors may be of two types: *nominal* and *ordinal*. Nominal factors have levels that correspond to names of the categories, with no implied ordering. Examples of nominal factors would be hair color, gender, race, or political party. There is no natural ordering to "Democrat" and "Republican"; the categories are just names associated with different groups of people.

In contrast, ordinal factors have some sort of ordered structure to the underlying factor levels. For instance, socioeconomic status would be an ordinal categorical variable because the levels correspond to ranks associated with income, education, and occupation. Another example of ordinal categorical data would be class rank.

Factors have special status in R. They are represented internally by numbers, but even when they are written numerically their values do not convey any numeric meaning or obey any mathematical rules (that is, Stage III cancer is not Stage I cancer + Stage II cancer).

Example 3.7. The `state.abb` vector gives the two letter postal abbreviations for all 50 states.

```
str(state.abb)
```

```
 chr [1:50] "AL" "AK" "AZ" "AR" "CA" "CO" ...
```

These would be ID data. The `state.name` vector lists all of the complete names and those data would also be ID.

Example 3.8 (U.S. State Facts and Features)**.** The U.S. Department of Commerce of the U.S. Census Bureau releases all sorts of information in the *Statistical Abstract of the United States*, and the `state.region` data lists each of the 50 states and the region to which it belongs, be it Northeast, South, North Central, or West. See `?state.region`.

```
str(state.region)
```

```
 Factor w/ 4 levels "Northeast","South",..: 2 4 4 2 4 4 1 2 2 2 ...
```

```
state.region[1:5]
```

```
[1] South West  West  South West
Levels: Northeast South North Central West
```

The `str` output shows that `state.region` is already stored internally as a factor and it lists a couple of the factor levels. To see all of the levels we printed the first five entries of the vector in the second line.

3.1.4 Displaying Qualitative Data

Tables

One of the best ways to summarize qualitative data is with a table of the data values. We may count frequencies with the `table` function or list proportions with the `prop.table` function (whose input is a frequency table). In the R Commander you can do it with `Statistics` ▹ `Frequency Distribution...` Alternatively, to look at tables for all factors in the `Active data set` you can do `Statistics` ▹ `Summaries` ▹ `Active Dataset`.

```
Tbl <- table(state.division)
Tbl
```

```
state.division
       New England    Middle Atlantic     South Atlantic 
                 6                  3                  8 
East South Central West South Central East North Central 
                 4                  4                  5 
West North Central           Mountain            Pacific 
                 7                  8                  5 
```

```
Tbl/sum(Tbl)      # relative frequencies
```

```
state.division
```

```
       New England    Middle Atlantic     South Atlantic
              0.12               0.06               0.16
East South Central West South Central East North Central
              0.08               0.08               0.10
West North Central           Mountain            Pacific
              0.14               0.16               0.10
```

```
prop.table(Tbl)    # same thing
```

```
state.division
       New England    Middle Atlantic     South Atlantic
              0.12               0.06               0.16
East South Central West South Central East North Central
              0.08               0.08               0.10
West North Central           Mountain            Pacific
              0.14               0.16               0.10
```

Bar Graphs

A bar graph is the analogue of a histogram for categorical data. A bar is displayed for each level of a factor, with the heights of the bars proportional to the frequencies of observations falling in the respective categories. A disadvantage of bar graphs is that the levels are ordered alphabetically (by default), which may sometimes obscure patterns in the display.

Example 3.9 (U.S. State Facts and Features)**.** The `state.region` data lists each of the 50 states and the region to which it belongs, be it Northeast, South, North Central, or West. See `?state.region`. It is already stored internally as a factor. We make a bar graph with the `barplot` function:

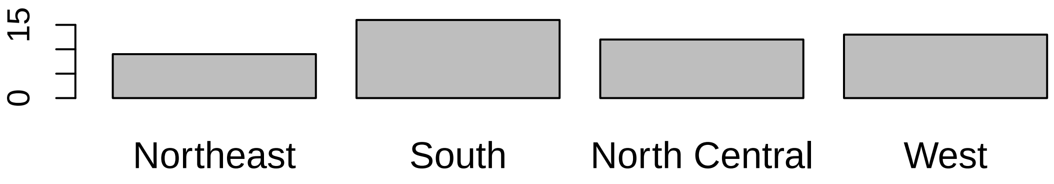
```
barplot(table(state.region), cex.names = 1.20)
barplot(prop.table(table(state.region)), cex.names = 1.20)
```

See Figure 3.5. The display on the left is a frequency bar graph because the y axis shows counts, while the display on the left is a relative frequency bar graph. The only difference between the two is the scale. Looking at the graph we see that the majority of the fifty states are in the South, followed by West, North Central, and finally Northeast. Over 30% of the states are in the South.

Notice the `cex.names` argument that we used, above. It expands the names on the x axis by 20% which makes them easier to read. See `?par` for a detailed list of additional plot parameters.

Pareto Diagrams

A pareto diagram is a lot like a bar graph except the bars are rearranged such that they decrease in height going from left to right. The rearrangement is handy because it can visually reveal structure (if any) in how fast the bars decrease – this is much more difficult when the bars are jumbled.



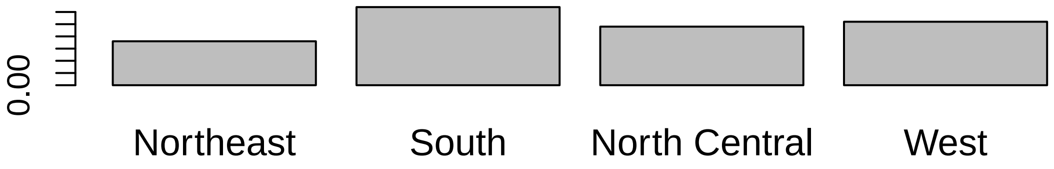


Figure 3.5: The top graph is a frequency barplot made with `table` and the bottom is a relative frequency barplot made with `prop.table`.

Example 3.10 (U.S. State Facts and Features)**.** The `state.division` data record the division (New England, Middle Atlantic, South Atlantic, East South Central, West South Central, East North Central, West North Central, Mountain, and Pacific) of the fifty states. We can make a pareto diagram with either the `RcmdrPlugin.IPSUR` package [152] or with the `pareto.chart` function from the `qcc` package [138]. See Figure 3.6. The code follows.

```
pareto.chart(table(state.division), ylab="Frequency", cex.lab = cexlab)
```

```
Pareto chart analysis for table(state.division)
                     Frequency Cum.Freq. Percentage
  South Atlantic             8         8         16
  Mountain                   8        16         16
  West North Central         7        23         14
  New England                6        29         12
  East North Central         5        34         10
  Pacific                    5        39         10
  East South Central         4        43          8
  West South Central         4        47          8
  Middle Atlantic            3        50          6

Pareto chart analysis for table(state.division)
                     Cum.Percent.
  South Atlantic               16
  Mountain                     32
```

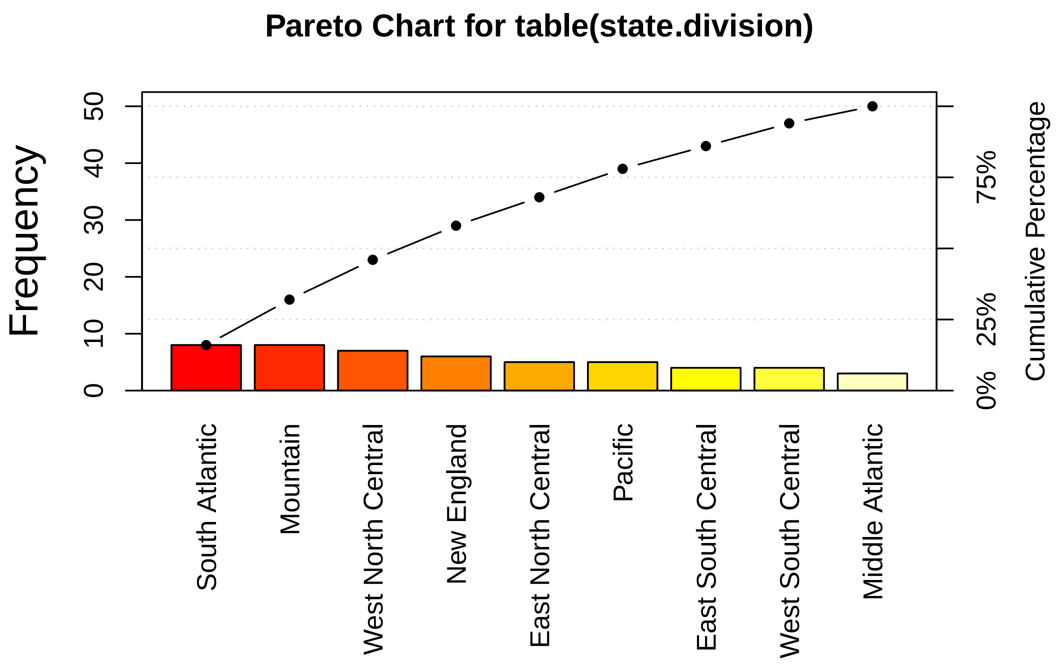


Figure 3.6: Pareto chart of the `state.division` data.

```
West North Central          46
New England                 58
East North Central          68
Pacific                     78
East South Central          86
West South Central          94
Middle Atlantic            100
```

Dot Charts

These are a lot like a bar graph that has been turned on its side with the bars replaced by dots on horizontal lines. They do not convey any more (or less) information than the associated bar graph, but the strength lies in the economy of the display. Dot charts are so compact that it is easy to graph very complicated multi-variable interactions together in one graph. See Section 3.6. We will give an example here using the same data as above for comparison. The graph was produced by the following code.

```
x <- table(state.region)
dotchart(as.vector(x), labels = names(x), cex.lab = cexlab)
```

See Figure 3.7. Compare it to Figure 3.5.

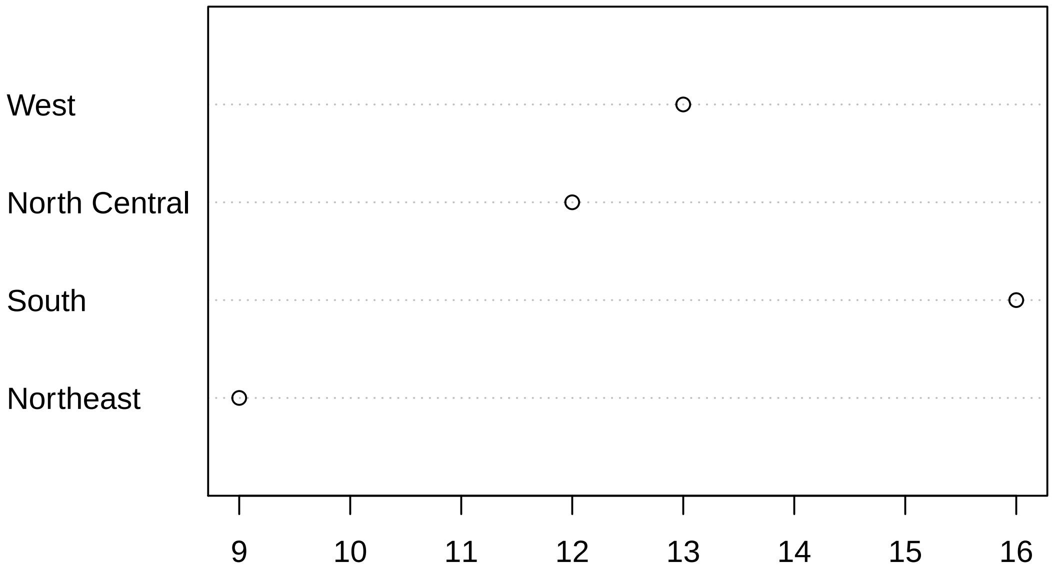


Figure 3.7: Dot chart of the `state.region` data.

Pie Graphs

These can be done with R and the R Commander, but they fallen out of favor in recent years because researchers have determined that while the human eye is good at judging linear measures, it is notoriously bad at judging relative areas (such as those displayed by a pie graph). Pie charts are consequently a very bad way of displaying information unless the number of categories is two or three. A bar chart or dot chart is a preferable way of displaying qualitative data. See `?pie` for more information.

We are not going to do any examples of a pie graph and discourage their use elsewhere.

3.1.5 Logical Data

There is another type of information recognized by R which does not fall into the above categories. The value is either `TRUE` or `FALSE` (note that equivalently you can use `1 = TRUE`, `0 = FALSE`). Here is an example of a logical vector:

```
x <- 5:9
y <- (x < 7.3)
y
```

```
[1]  TRUE  TRUE  TRUE FALSE FALSE
```

Many functions in R have options that the user may or may not want to activate in the function call. For example, the `stem.leaf` function has the `depths` argument which is `TRUE` by default. We saw in Section 3.1.1 how to turn the option off, simply enter `stem.leaf(x, depths = FALSE)` and they will not be shown on the display.

We can swap `TRUE` with `FALSE` with the exclamation point `!`.

```
!y
```

```
[1] FALSE FALSE FALSE  TRUE  TRUE
```

3.1.6 Missing Data

Missing data are a persistent and prevalent problem in many statistical analyses, especially those associated with the social sciences. R reserves the special symbol `NA` to representing missing data.

Ordinary arithmetic with `NA` values give `NA`'s (addition, subtraction, *etc.*) and applying a function to a vector that has an `NA` in it will usually give an `NA`.

```
x <- c(3, 7, NA, 4, 7)
y <- c(5, NA, 1, 2, 2)
x + y
```

```
[1]  8 NA NA  6  9
```

Some functions have a `na.rm` argument which when `TRUE` will ignore missing data as if they were not there (such as `mean`, `var`, `sd`, `IQR`, `mad`, . . .).

```
sum(x)
```

```
[1] NA
```

```
sum(x, na.rm = TRUE)
```

```
[1] 21
```

Other functions do not have a `na.rm` argument and will return `NA` or an error if the argument has `NA`s. In those cases we can find the locations of any `NA`s with the `is.na` function and remove those cases with the `[]` operator.

```
is.na(x)
```

```
[1] FALSE FALSE  TRUE FALSE FALSE
```

```
z <- x[!is.na(x)]
sum(z)
```

```
[1] 21
```

The analogue of `is.na` for rectangular data sets (or data frames) is the `complete.cases` function. See Appendix **??**.

3.1.7 Other Data Types

3.2 Features of Data Distributions

Given that the data have been appropriately displayed, the next step is to try to identify salient features represented in the graph. The acronym to remember is *C*-enter, *U*-nusual features, *S*-pread, and *S*-hape. (CUSS).

3.2.1 Center

One of the most basic features of a data set is its center. Loosely speaking, the center of a data set is associated with a number that represents a middle or general tendency of the data. Of course, there are usually several values that would serve as a center, and our later tasks will be focused on choosing an appropriate one for the data at hand. Judging from the histogram that we saw in Figure 3.3, a measure of center would be about 35.

3.2.2 Spread

The spread of a data set is associated with its variability; data sets with a large spread tend to cover a large interval of values, while data sets with small spread tend to cluster tightly around a central value.

3.2.3 Shape

When we speak of the *shape* of a data set, we are usually referring to the shape exhibited by an associated graphical display, such as a histogram. The shape can tell us a lot about any underlying structure to the data, and can help us decide which statistical procedure we should use to analyze them.

Symmetry and Skewness

A distribution is said to be *right-skewed* (or *positively skewed*) if the right tail seems to be stretched from the center. A *left-skewed* (or *negatively skewed*) distribution is stretched to the left side. A symmetric distribution has a graph that is balanced about its center, in the sense that half of the graph may be reflected about a central line of symmetry to match the other half.

We have already encountered skewed distributions: both the discoveries data in Figure 3.1 and the `precip` data in Figure 3.3 appear right-skewed. The `UKDriverDeaths` data in Example 3.5 is relatively symmetric (but note the one extreme value 2654 identified at the bottom of the stem-and-leaf display).

Kurtosis

Another component to the shape of a distribution is how “peaked” it is. Some distributions tend to have a flat shape with thin tails. These are called *platykurtic*, and an example of a platykurtic distribution is the uniform distribution; see Section **??**. On the other end of the spectrum are distributions with a steep peak, or spike, accompanied by heavy tails; these are called *leptokurtic*. Examples of leptokurtic distributions are the Laplace distribution and the logistic distribution. See Section **??**. In between are distributions (called *mesokurtic*) with a rounded peak and moderately sized tails. The standard example of a mesokurtic distribution is the famous bell-shaped curve, also known as the Gaussian, or normal, distribution, and the binomial distribution can be mesokurtic for specific choices of p. See Sections **??** and **??**.

3.2.4 Clusters and Gaps

Clusters or gaps are sometimes observed in quantitative data distributions. They indicate clumping of the data about distinct values, and gaps may exist between clusters. Clusters often suggest an underlying grouping to the data. For example, take a look at the `faithful` data which contains the duration of `eruptions` and the `waiting` time between eruptions of the Old Faithful geyser in Yellowstone National Park. Do not be frightened by the complicated information at the left of the display for now; we will learn how to interpret it in Section 3.4.

```r
with(faithful, stem.leaf(eruptions))
```

```
1 | 2: represents 1.2
 leaf unit: 0.1
            n: 272
   12     s | 667777777777
   51    1. | 888888888888888888888888888899999999999
   71    2* | 00000000000011111111
   87     t | 2222222222333333
   92     f | 44444
   94     s | 66
   97    2. | 889
   98    3* | 0
  102     t | 3333
  108     f | 445555
  118     s | 6666677777
  (16)   3. | 8888888888999999
  138    4* | 0000000000000000111111111111111
  107     t | 22222222222233333333333333333
   78     f | 44444444444445555555555555555555555
   43     s | 6666666666667777777777
   21    4. | 88888888888899999
    4    5* | 0001
```

There are definitely two clusters of data here; an upper cluster and a lower cluster.

3.2.5 Extreme Observations and other Unusual Features

Extreme observations fall far from the rest of the data. Such observations are troublesome to many statistical procedures; they cause exaggerated estimates and instability. It is important to identify extreme observations and examine the source of the data more closely. There are many possible reasons underlying an extreme observation:

- *Maybe the value is a typographical error.* Especially with large data sets becoming more prevalent, many of which being recorded by hand, mistakes are a common problem. After closer scrutiny, these can often be fixed.
- *Maybe the observation was not meant for the study*, because it does not belong to the population of interest. For example, in medical research some subjects may have relevant complications in their genealogical history that would rule out their participation in the experiment. Or when a manufacturing company investigates the properties of one of its devices, perhaps a particular product is malfunctioning and is not representative of the majority of the items.
- *Maybe it indicates a deeper trend or phenomenon.* Many of the most influential scientific discoveries were made when the investigator noticed an unexpected result, a value that was not predicted by the classical theory. Albert Einstein, Louis Pasteur, and others built their careers on exactly this circumstance.

3.3 Descriptive Statistics

One of my favorite professors would repeatedly harp, "You cannot do statistics without data."

What do I want them to know?

- The fundamental data types we encounter most often, how to classify given data into a likely type, and that sometimes the distinction is blurry.

3.3.1 Frequencies and Relative Frequencies

These are used for categorical data. The idea is that there are a number of different categories, and we would like to get some idea about how the categories are represented in the population.

3.3.2 Measures of Center

The *sample mean* is denoted $\overline{x}$ (read "x-bar") and is simply the arithmetic average of the observations:

$$\overline{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i. \tag{3.1}$$

- Good: natural, easy to compute, has nice mathematical properties
- Bad: sensitive to extreme values

It is appropriate for use with data sets that are not highly skewed without extreme observations.

The *sample median* is another popular measure of center and is denoted $\tilde{x}$. To calculate its value, first sort the data into an increasing sequence of numbers. If the data set has an odd number of observations then $\tilde{x}$ is the value of the middle observation, which lies in position $(n+1)/2$; otherwise, there are two middle observations and $\tilde{x}$ is the average of those middle values.

- Good: resistant to extreme values, easy to describe
- Bad: not as mathematically tractable, need to sort the data to calculate

One desirable property of the sample median is that it is *resistant* to extreme observations, in the sense that the value of $\tilde{x}$ depends only on those data values in the middle, and is quite unaffected by the actual values of the outer observations in the ordered list. The same cannot be said for the sample mean. Any significant changes in the magnitude of an observation x_k results in a corresponding change in the value of the mean. Consequently, the sample mean is said to be *sensitive* to extreme observations.

The *trimmed mean* is a measure designed to address the sensitivity of the sample mean to extreme observations. The idea is to “trim” a fraction (less than 1/2) of the observations off each end of the ordered list, and then calculate the sample mean of what remains. We will denote it by $\overline{x}_{t=0.05}$.

- Good: resistant to extreme values, shares nice statistical properties
- Bad: need to sort the data

How to do it with R

- You can calculate frequencies or relative frequencies with the `table` function, and relative frequencies with `prop.table(table())`.
- You can calculate the sample mean of a data vector `x` with the command `mean(x)`.
- You can calculate the sample median of `x` with the command `median(x)`.
- You can calculate the trimmed mean with the `trim` argument; `mean(x, trim = 0.05)`.

3.3.3 Order Statistics and the Sample Quantiles

A common first step in an analysis of a data set is to sort the values. Given a data set $x_1, x_2, \ldots, x_n$, we may sort the values to obtain an increasing sequence

$$x_{(1)} \leq x_{(2)} \leq x_{(3)} \leq \cdots \leq x_{(n)} \tag{3.2}$$

and the resulting values are called the *order statistics*. The k^{th} entry in the list, $x_{(k)}$, is the k^{th} order statistic, and approximately $100(k/n)\%$ of the observations fall below $x_{(k)}$. The order statistics give an indication of the shape of the data distribution, in the sense that a person can look at the order statistics and have an idea about where the data are concentrated, and where they are sparse.

The *sample quantiles* are related to the order statistics. Unfortunately, there is not a universally accepted definition of them. Indeed, R is equipped to calculate quantiles using nine distinct definitions! We will describe the default method (`type = 7`), but the interested reader can see the details for the other methods with `?quantile`.

Suppose the data set has n observations. Find the sample quantile of order p ($0 < p < 1$), denoted $\tilde{q}_p$, as follows:

- **First step:** sort the data to obtain the order statistics $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$.
- **Second step:** calculate $(n-1)p + 1$ and write it in the form $k.d$, where k is an integer and d is a decimal.
- **Third step:** The sample quantile $\tilde{q}_p$ is

$$\tilde{q}_p = x_{(k)} + d(x_{(k+1)} - x_{(k)}). \tag{3.3}$$

The interpretation of $\tilde{q}_p$ is that approximately $100p$ % of the data fall below the value $\tilde{q}_p$.

Keep in mind that there is not a unique definition of percentiles, quartiles, *etc.* Open a different book, and you'll find a different definition. The difference is small and seldom plays a role except in small data sets with repeated values. In fact, most people do not even notice in common use.

Clearly, the most popular sample quantile is $\tilde{q}_{0.50}$, also known as the sample median, $\tilde{x}$. The closest runners-up are the *first quartile* $\tilde{q}_{0.25}$ and the *third quartile* $\tilde{q}_{0.75}$ (the *second quartile* is the median).

How to do it with R

At the command prompt We can find the order statistics of a data set stored in a vector `x` with the command `sort(x)`.

We can calculate the sample quantiles of any order p where $0 < p < 1$ for a data set stored in a data vector `x` with the `quantile` function, for instance, the command `quantile(x, probs = c(0, 0.25, 0.37))` will return the smallest observation, the first quartile, $\tilde{q}_{0.25}$, and the 37th sample quantile, $\tilde{q}_{0.37}$. For $\tilde{q}_p$ simply change the values in the `probs` argument to the value p.

With the R Commander we can find the order statistics of a variable in the `Active data set` by doing `Data` ▷ `Manage variables in Active data set...` ▷ `Compute new variable...` In the `Expression to compute` dialog simply type `sort(varname)`, where `varname` is the variable that it is desired to sort.

In `Rcmdr`, we can calculate the sample quantiles for a particular variable with the sequence `Statistics` ▷ `Summaries` ▷ `Numerical Summaries...` We can automatically calculate the quartiles for all variables in the `Active data set` with the sequence `Statistics` ▷ `Summaries` ▷ `Active Dataset`.

3.3.4 Measures of Spread

Sample Variance and Standard Deviation

The *sample variance* is denoted s^2 and is calculated with the formula

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2. \tag{3.4}$$

The *sample standard deviation* is $s = \sqrt{s^2}$. Intuitively, the sample variance is approximately the average squared distance of the observations from the sample mean. The sample standard deviation is used to scale the estimate back to the measurement units of the original data.

- Good: tractable, has nice mathematical/statistical properties
- Bad: sensitive to extreme values

We will spend a lot of time with the variance and standard deviation in the coming chapters. In the meantime, the following two rules give some meaning to the standard deviation, in that there are bounds on how much of the data can fall past a certain distance from the mean.

Fact: (Chebychev's Rule). The proportion of observations within k standard deviations of the mean is at least $1 - 1/k^2$, *i.e.*, at least 75%, 89%, and 94% of the data are within 2, 3, and 4 standard deviations of the mean, respectively.

Note that Chebychev's Rule does not say anything about when $k = 1$, because $1 - 1/1^2 = 0$, which states that at least 0% of the observations are within one standard deviation of the mean (which is not saying much).

Chebychev's Rule applies to any data distribution, *any* list of numbers, no matter where it came from or what the histogram looks like. The price for such generality is that the bounds are not very tight; if we know more about how the data are shaped then we can say more about how much of the data can fall a given distance from the mean.

Fact: (Empirical Rule). If data follow a bell-shaped curve, then approximately 68%, 95%, and 99.7% of the data are within 1, 2, and 3 standard deviations of the mean, respectively.

Interquartile Range

Just as the sample mean is sensitive to extreme values, so the associated measure of spread is similarly sensitive to extremes. Further, the problem is exacerbated by the fact that the extreme distances are squared. We know that the sample quartiles are resistant to extremes, and a measure of spread associated with them is the *interquartile range* (IQR) defined by $IQR = q_{0.75} - q_{0.25}$.

- Good: stable, resistant to outliers, robust to nonnormality, easy to explain
- Bad: not as tractable, need to sort the data, only involves the middle 50% of the data.

Median Absolute Deviation

A measure even more robust than the IQR is the *median absolute deviation* (MAD). To calculate it we first get the median $\widetilde{x}$, next the *absolute deviations* $|x_1 - \tilde{x}|, |x_2 - \tilde{x}|, \ldots, |x_n - \tilde{x}|$, and the MAD is proportional to the median of those deviations:

$$MAD \propto \text{median}(|x_1 - \tilde{x}|,\ |x_2 - \tilde{x}|, \ldots, |x_n - \tilde{x}|). \tag{3.5}$$

That is, the $MAD = c \cdot \text{median}(|x_1 - \tilde{x}|,\ |x_2 - \tilde{x}|, \ldots, |x_n - \tilde{x}|)$, where c is a constant chosen so that the MAD has nice properties. The value of c in R is by default $c = 1.4286$. This value is chosen to ensure that the estimator of σ is correct, on the average, under suitable sampling assumptions (see Section **??**).

- Good: stable, very robust, even more so than the *IQR*.
- Bad: not tractable, not well known and less easy to explain.

Comparing Apples to Apples

We have seen three different measures of spread which, for a given data set, will give three different answers. Which one should we use? It depends on the data set. If the data are well behaved, with an approximate bell-shaped distribution, then the sample mean and sample standard deviation are natural choices with nice mathematical properties. However, if the data have an unusual or skewed shape with several extreme values, perhaps the more resistant choices among the *IQR* or *MAD* would be more appropriate.

However, once we are looking at the three numbers it is important to understand that the estimators are not all measuring the same quantity, on the average. In particular, it can be shown that when the data follow an approximately bell-shaped distribution, then on the average, the sample standard deviation s and the *MAD* will be the approximately the same value, namely, σ, but the *IQR* will be on the average 1.349 times larger than s and the *MAD*. See **??** for more details.

How to do it with R

At the command prompt we may compute the sample range with `range(x)` and the sample variance with `var(x)`, where `x` is a numeric vector. The sample standard deviation is `sqrt(var(x))` or just `sd(x)`. The *IQR* is `IQR(x)` and the median absolute deviation is `mad(x)`.

With the R Commander we can calculate the sample standard deviation with the `Statistics` ▹ `Summaries` ▹ `Numerical Summaries...` combination. R Commander does not calculate the *IQR* or *MAD* in any of the menu selections, by default.

3.3.5 Measures of Shape

Sample Skewness

The *sample skewness*, denoted by g_1, is defined by the formula

$$g_1 = \frac{1}{n} \frac{\sum_{i=1}^{n} (x_i - \overline{x})^3}{s^3}. \tag{3.6}$$

The sample skewness can be any value $-\infty < g_1 < \infty$. The sign of g_1 indicates the direction of skewness of the distribution. Samples that have $g_1 > 0$ indicate right-skewed distributions (or positively skewed), and samples with $g_1 < 0$ indicate left-skewed distributions (or negatively skewed). Values of g_1 near zero indicate a symmetric distribution. These are not hard and fast rules, however. The value of g_1 is subject to sampling variability and thus only provides a suggestion to the skewness of the underlying distribution.

We still need to know how big is "big", that is, how do we judge whether an observed value of g_1 is far enough away from zero for the data set to be considered skewed to the right or left? A good rule of thumb is that data sets with skewness larger than $2\sqrt{6/n}$ in magnitude are substantially skewed, in the direction of the sign of g_1. See Tabachnick & Fidell [149] for details.

Sample Excess Kurtosis

The *sample excess kurtosis*, denoted by g_2, is given by the formula

$$g_2 = \frac{1}{n}\frac{\sum_{i=1}^{n}(x_i - \overline{x})^4}{s^4} - 3. \tag{3.7}$$

The sample excess kurtosis takes values $-2 \leq g_2 < \infty$. The subtraction of 3 may seem mysterious but it is done so that mound shaped samples have values of g_2 near zero. Samples with $g_2 > 0$ are called *leptokurtic*, and samples with $g_2 < 0$ are called *platykurtic*. Samples with $g_2 \approx 0$ are called *mesokurtic*.

As a rule of thumb, if $|g_2| > 4\sqrt{6/n}$ then the sample excess kurtosis is substantially different from zero in the direction of the sign of g_2. See Tabachnick & Fidell [149] for details.

Notice that both the sample skewness and the sample kurtosis are invariant with respect to location and scale, that is, the values of g_1 and g_2 do not depend on the measurement units of the data.

How to do it with R

The `e1071` package [96] has the `skewness` function for the sample skewness and the `kurtosis` function for the sample excess kurtosis. Both functions have a `na.rm` argument which is `FALSE` by default.

Example 3.11. We said earlier that the `discoveries` data looked positively skewed; let's see what the statistics say:

```
skewness(discoveries)
```

```
[1] 1.2076
```

```
2*sqrt(6/length(discoveries))
```

```
[1] 0.4898979
```

The data are definitely skewed to the right. Let us check the sample excess kurtosis of the `UKDriverDeaths` data:

```
kurtosis(UKDriverDeaths)
```

```
[1] 0.07133848
```

```
4*sqrt(6/length(UKDriverDeaths))
```

```
[1] 0.7071068
```

so that the `UKDriverDeaths` data appear to be mesokurtic, or at least not substantially leptokurtic.

3.4 Exploratory Data Analysis

This field was founded (mostly) by John Tukey (1915-2000). Its tools are useful when not much is known regarding the underlying causes associated with the data set, and are often used for checking assumptions. For example, suppose we perform an experiment and collect some data... now what? We look at the data using exploratory visual tools.

3.4.1 More About Stem-and-leaf Displays

There are many bells and whistles associated with stemplots, and the `stem.leaf` function can do many of them.

- **Trim Outliers:** Some data sets have observations that fall far from the bulk of the other data (in a sense made more precise in Section 3.4.4). These extreme observations often obscure the underlying structure to the data and are best left out of the data display. The `trim.outliers` argument (which is `TRUE` by default) will separate the extreme observations from the others and graph the stemplot without them; they are listed at the bottom (respectively, top) of the stemplot with the label `HI` (respectively `LO`).
- **Split Stems:** The standard stemplot has only one line per stem, which means that all observations with first digit `3` are plotted on the same line, regardless of the value of the second digit. But this gives some stemplots a "skyscraper" appearance, with too many observations stacked onto the same stem. We can often fix the display by increasing the number of lines available for a given stem. For example, we could make two lines per stem, say, `3*` and `3.`. Observations with second digit 0 through 4 would go on the upper line, while observations with second digit 5 through 9 would go on the lower line. (We could do a similar thing with five lines per stem, or even ten lines per stem.) The end result is a more spread out stemplot which often looks better. A good example of this was shown on page **??**.
- **Depths:** these are used to give insight into the balance of the observations as they accumulate toward the median. In a column beside the standard stemplot, the frequency of the stem containing the sample median is shown in parentheses. Next, frequencies are accumulated from the outside inward, including the outliers. Distributions that are more symmetric will have better balanced depths on either side of the sample median.

How to do it with R

The basic command is `stem(x)` or a more sophisticated version written by Peter Wolf called `stem.leaf(x)` in the R Commander. We will describe `stem.leaf` since that is the one used by R Commander.

WARNING: Sometimes when making a stem-and-leaf display the result will not be what you expected. There are several reasons for this:

- Stemplots by default will trim extreme observations (defined in Section 3.4.4) from the display. This in some cases will result in stemplots that are not as wide as expected.
- The leafs digit is chosen automatically by `stem.leaf` according to an algorithm that the computer believes will represent the data well. Depending on the choice of the digit, `stem.leaf` may drop digits from the data or round the values in unexpected ways.

Let us take a look at the `rivers` data set.

```
stem.leaf(rivers)
```

```
1 | 2: represents 120
 leaf unit: 10
            n: 141
    1     1 | 3
   29     2 | 0111133334555556666778888899
   64     3 | 00000111122223333455555666677888999
  (18)    4 | 011222233344566679
   59     5 | 000222234467
   47     6 | 0000112235789
   34     7 | 12233368
   26     8 | 04579
   21     9 | 0008
   17    10 | 035
   14    11 | 07
   12    12 | 047
    9    13 | 0
HI: 1450 1459 1770 1885 2315 2348 2533 3710
```

The stem-and-leaf display shows a right-skewed shape to the `rivers` data distribution. Notice that the last digit of each of the data values were dropped from the display. Notice also that there were eight extreme observations identified by the computer, and their exact values are listed at the bottom of the stemplot. Look at the scale on the left of the stemplot and try to imagine how ridiculous the graph would have looked had we tried to include enough stems to include these other eight observations; the stemplot would have stretched over several pages. Notice finally that we can use the depths to approximate the sample median for these data. The median lies in the row identified by `(18)`, which means that the median is the average of the ninth and tenth observation on that row. Those two values correspond to `43` and `43`, so a good guess for the median would be 430. (For the record, the sample median is $\widetilde{x} = 425$. Recall that stemplots round the data to the nearest stem-leaf pair.)

Next let us see what the `precip` data look like.

```
stem.leaf(precip)
```

```
1 | 2: represents 12
 leaf unit: 1
            n: 70
LO: 7 7.2 7.8 7.8
    8    1* | 1344
   13    1. | 55677
   16    2* | 024
   18    2. | 59
   28    3* | 0000111234
  (15)   3. | 555566677788899
   27    4* | 0000122222334
   14    4. | 56688899
    6    5* | 44
    4    5. | 699
HI: 67
```

Here is an example of split stems, with two lines per stem. The final digit of each datum has been dropped for the display. The data appear to be left skewed with four extreme values to the left and one extreme value to the right. The sample median is approximately 37 (it turns out to be 36.6).

3.4.2 Hinges and the Five Number Summary

Given a data set $x_1, x_2, \ldots, x_n$, the hinges are found by the following method:

- Find the order statistics $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$.
- The *lower hinge* h_L is in position $L = \lfloor (n+3)/2 \rfloor /2$, where the symbol $\lfloor x \rfloor$ denotes the largest integer less than or equal to x. If the position L is not an integer, then the hinge h_L is the average of the adjacent order statistics.
- The *upper hinge* h_U is in position $n + 1 - L$.

Given the hinges, the *five number summary* ($5NS$) is

$$5NS = (x_{(1)},\ h_L,\ \tilde{x},\ h_U,\ x_{(n)}). \tag{3.8}$$

An advantage of the $5NS$ is that it reduces a potentially large data set to a shorter list of only five numbers, and further, these numbers give insight regarding the shape of the data distribution similar to the sample quantiles in Section 3.3.3.

How to do it with R

If the data are stored in a vector `x`, then you can compute the $5NS$ with the `fivenum` function.

3.4.3 Boxplots

A boxplot is essentially a graphical representation of the $5NS$. It can be a handy alternative to a stripchart when the sample size is large.

A boxplot is constructed by drawing a box alongside the data axis with sides located at the upper and lower hinges. A line is drawn parallel to the sides to denote the sample median. Lastly, whiskers are extended from the sides of the box to the maximum and minimum data values (more precisely, to the most extreme values that are not potential outliers, defined below).

Boxplots are good for quick visual summaries of data sets, and the relative positions of the values in the $5NS$ are good at indicating the underlying shape of the data distribution, although perhaps not as effectively as a histogram. Perhaps the greatest advantage of a boxplot is that it can help to objectively identify extreme observations in the data set as described in the next section.

Boxplots are also good because one can visually assess multiple features of the data set simultaneously:

- **Center:** can be estimated by the sample median, $\tilde{x}$.
- **Spread:** can be judged by the width of the box, $h_U - h_L$. We know that this will be close to the IQR, which can be compared to s and the MAD, perhaps after rescaling if appropriate.
- **Shape:** is indicated by the relative lengths of the whiskers, and the position of the median inside the box. Boxes with unbalanced whiskers indicate skewness in the direction of the long whisker. Skewed distributions often have the median tending in the opposite direction of skewness. Kurtosis can be assessed using the box and whiskers. A wide box with short whiskers will tend to be platykurtic, while a skinny box with wide whiskers indicates leptokurtic distributions.
- **Extreme observations:** are identified with open circles (see below).

3.4.4 Outliers

A *potential outlier* is any observation that falls beyond 1.5 times the width of the box on either side, that is, any observation less than $h_L - 1.5(h_U - h_L)$ or greater than $h_U + 1.5(h_U - h_L)$. A *suspected outlier* is any observation that falls beyond 3 times the width of the box on either side. In R, both potential and suspected outliers (if present) are denoted by open circles; there is no distinction between the two.

When potential outliers are present, the whiskers of the boxplot are then shortened to extend to the most extreme observation that is not a potential outlier. If an outlier is displayed in a boxplot, the index of the observation may be identified in a subsequent plot in `Rcmdr` by clicking the `Identify outliers with mouse` option in the `Boxplot` dialog.

What do we do about outliers? They merit further investigation. The primary goal is to determine why the observation is outlying, if possible. If the observation is a typographical error, then it should be corrected before continuing. If the observation is from a subject that does not belong to the population of interest, then perhaps the datum should be removed. Otherwise, perhaps the value is hinting at some hidden structure to the data.

How to do it with R

The quickest way to visually identify outliers is with a boxplot, described above. Another way is with the `boxplot.stats` function.

Example 3.12 (Lengths of Major North American Rivers)**.** We will look for potential outliers in the `rivers` data.

```
boxplot.stats(rivers)$out
```

```
 [1] 1459 1450 1243 2348 3710 2315 2533 1306 1270 1885 1770
```

We may change the `coef` argument to 3 (it is 1.5 by default) to identify suspected outliers.

```
boxplot.stats(rivers, coef = 3)$out
```

```
[1] 2348 3710 2315 2533 1885
```

3.4.5 Standardizing variables

It is sometimes useful to compare data sets with each other on a scale that is independent of the measurement units. Given a set of observed data $x_1, x_2, \ldots, x_n$ we get z scores, denoted $z_1, z_2, \ldots, z_n$, by means of the following formula

$$z_i = \frac{x_i - \overline{x}}{s}, \quad i = 1, 2, \ldots, n.$$

How to do it with R

The `scale` function will rescale a numeric vector (or data frame) by subtracting the sample mean from each value (column) and/or by dividing each observation by the sample standard deviation.

3.5 Multivariate Data and Data Frames

We have had experience with vectors of data, which are long lists of numbers. Typically, each entry in the vector is a single measurement on a subject or experimental unit in the study. We saw in Section **??** how to form vectors with the `c` function or the `scan` function.

However, statistical studies often involve experiments where there are two (or more) measurements associated with each subject. We display the measured information in a rectangular array in which each row corresponds to a subject, and the columns contain the measurements for each respective variable. For instance, if one were to measure the height and weight and hair color of each of 11 persons in a research study, the information could be represented with a rectangular array. There would be 11 rows. Each row would have the person's height in the first column and hair color in the second column.

The corresponding objects in R are called *data frames*, and they can be constructed with the `data.frame` function. Each row is an observation, and each column is a variable.

Example 3.13. Suppose we have two vectors `x` and `y` and we want to make a data frame out of them.

```
x <- 5:8
y <- letters[3:6]
A <- data.frame(v1 = x, v2 = y)
```

Notice that `x` and `y` are the same length. This is *necessary*. Also notice that `x` is a numeric vector and `y` is a character vector. We may choose numeric and character vectors (or even factors) for the columns of the data frame, but each column must be of exactly one type. That is, we can have a column for `height` and a column for `gender`, but we will get an error if we try to mix function `height` (numeric) and `gender` (character or factor) information in the same column.

Indexing of data frames is similar to indexing of vectors. To get the entry in row i and column j do `A[i,j]`. We can get entire rows and columns by omitting the other index.

```
A[3, ]
```

```
  v1 v2
3  7  e
```

```
A[ , 1]
```

```
[1] 5 6 7 8
```

```
A[ , 2]
```

```
[1] c d e f
Levels: c d e f
```

There are several things happening above. Notice that `A[3, ]` gave a data frame (with the same entries as the third row of `A`) yet `A[ , 1]` is a numeric vector. `A[ ,2]` is a factor vector because the default setting for `data.frame` is `stringsAsFactors = TRUE`.

Data frames have a `names` attribute and the names may be extracted with the `names` function. Once we have the names we may extract given columns by way of the dollar sign.

```
names(A)
```

```
[1] "v1" "v2"
```

```
A['v1']
```

```
  v1
1  5
2  6
3  7
4  8
```

The above is identical to `A[ ,1]`.

3.5.1 Bivariate Data

- Stacked bar charts
- odds ratio and relative risk
- Introduce the sample correlation coefficient.

The *sample Pearson product-moment correlation coefficient*:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})}}$$

- independent of scale
- $-1 < r < 1$
- measures *strength* and *direction* of linear association
- Two-Way Tables. Done with `table`, or in the R Commander by following `Statistics` ▹ `Contingency Tables` ▹} `Two-way Tables`. You can also enter and analyze a two-way table.
- table
- prop.table
- addmargins
- rowPercents (Rcmdr)
- colPercents (Rcmdr)
- totPercents(Rcmdr)
- A <- xtabs(~ gender + race, data = RcmdrTestDrive)
- xtabs(Freq ~ Class + Sex, data = Titanic) from built in table
- barplot(A, legend.text=TRUE)
- barplot(t(A), legend.text=TRUE)
- barplot(A, legend.text=TRUE, beside = TRUE)
- spineplot(gender ~ race, data = RcmdrTestDrive)
- Spine plot: plots categorical versus categorical
- Scatterplot: look for linear association and correlation.
- carb ~ optden, data = Formaldehyde (boring)
- conc ~ rate, data = Puromycin
- xyplot(accel ~ dist, data = attenu) nonlinear association
- xyplot(eruptions ~ waiting, data = faithful) (linear, two groups)
- xyplot(Petal.Width ~ Petal.Length, data = iris)

- xyplot(pressure ~ temperature, data = pressure) (exponential growth)
- xyplot(weight ~ height, data = women) (strong positive linear)

3.5.2 Multivariate Data

Multivariate Data Display

- Multi-Way Tables. You can do this with `table`, or in R Commander by following `Statistics` ▹ `Contingency Tables` ▹ `Multi-way Tables`.
- Scatterplot matrix. used for displaying pairwise scatterplots simultaneously. Again, look for linear association and correlation.
- 3D Scatterplot. See Figure **??**
- `plot(state.region, state.division)`
- `barplot(table(state.division,state.region), legend.text=TRUE)`

```r
require(graphics)
mosaicplot(HairEyeColor)
x <- apply(HairEyeColor, c(1, 2), sum)
x
mosaicplot(x, main = "Relation between hair and eye color")
y <- apply(HairEyeColor, c(1, 3), sum)
y
mosaicplot(y, main = "Relation between hair color and sex")
z <- apply(HairEyeColor, c(2, 3), sum)
z
mosaicplot(z, main = "Relation between eye color and sex")
```

3.6 Comparing Populations

Sometimes we have data from two or more groups (or populations) and we would like to compare them and draw conclusions. Some issues that we would like to address:

- Comparing centers and spreads: variation within versus between groups
- Comparing clusters and gaps
- Comparing outliers and unusual features
- Comparing shapes.

3.6.1 Numerically

I am thinking here about the `Statistics` ▹ `Numerical Summaries` ▹ `Summarize by groups` option or the `Statistics` ▹ `Summaries` ▹ `Table of Statistics` option.

3.6.2 Graphically

- Boxplots
- Variable width: the width of the drawn boxplots are proportional to $\sqrt{n_i}$, where n_i is the size of the i^{th} group. Why? Because many statistics have variability proportional to the reciprocal of the square root of the sample size.
- Notches: extend to $1.58 \cdot (h_U - h_L)/\sqrt{n}$. The idea is to give roughly a 95% confidence interval for the difference in two medians. See Chapter **??**.
- Stripcharts
- stripchart(weight ~ feed, method= "stack", data=chickwts)
- Bar Graphs
- barplot(xtabs(Freq ~ Admit + Gender, data = UCBAdmissions)) stacked bar chart
- barplot(xtabs(Freq ~ Admit, data = UCBAdmissions))
- barplot(xtabs(Freq ~ Gender + Admit, data = UCBAdmissions, legend = TRUE, beside = TRUE) oops, discrimination.
- barplot(xtabs(Freq ~ Admit+Dept, data = UCBAdmissions), legend = TRUE, beside = TRUE) different departments have different standards
- barplot(xtabs(Freq ~ Gender+Dept, data = UCBAdmissions), legend = TRUE, beside = TRUE) men mostly applied to easy departments, women mostly applied to difficult departments
- barplot(xtabs(Freq ~ Gender+Dept, data = UCBAdmissions), legend = TRUE, beside = TRUE)
- barchart(Admit ~ Freq, data = C)
- barchart(Admit ~ Freq|Gender, data = C)
- barchart(Admit ~ Freq | Dept, groups = Gender, data = C)
- barchart(Admit ~ Freq | Dept, groups = Gender, data = C, auto.key = TRUE)
- Histograms
- ~ breaks | wool{*}tension, data = warpbreaks
- ~ weight | feed, data = chickwts
- ~ weight | group, data = PlantGrowth
- ~ count | spray, data = InsectSprays
- ~ len | dose, data = ToothGrowth
- ~ decrease | treatment, data = OrchardSprays (or rowpos or colpos)
- Scatterplots

```
xyplot(Petal.Width ~ Petal.Length, data = iris, group = Species)
```

- Scatterplot matrices

- splom(~ cbind(GNP.deflator,GNP,Unemployed,Armed.Forces,Population,Year,Employed), data = longley)

- splom(~ cbind(pop15,pop75,dpi), data = LifeCycleSavings)

- splom(~ cbind(Murder, Assault, Rape), data = USArrests)

- splom(~ cbind(CONT, INTG, DMNR), data = USJudgeRatings)

- splom(~ cbind(area,peri,shape,perm), data = rock)

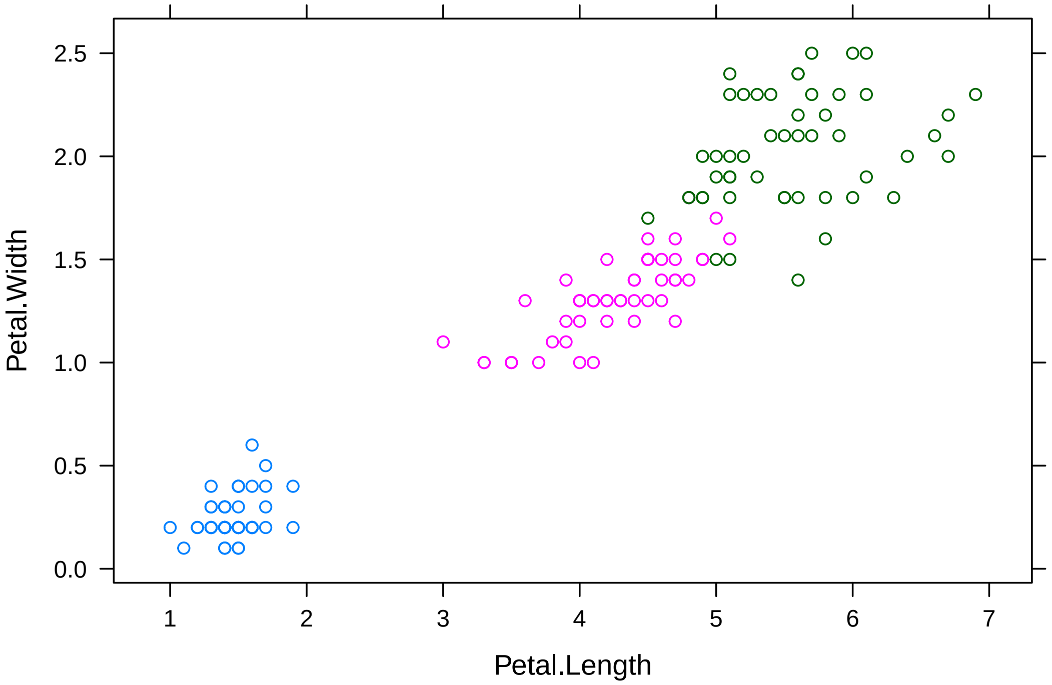


Figure 3.8: Scatterplot of Petal width versus length in the `iris` data.

- splom(~ cbind(Air.Flow, Water.Temp, Acid.Conc., stack.loss), data = stackloss)
- splom(~ cbind(Fertility,Agriculture,Examination,Education,Catholic,Infant.Mortality), data = swiss)
- splom(~ cbind(Fertility,Agriculture,Examination), data = swiss) (positive and negative)
- Dot charts
- dotchart(USPersonalExpenditure)
- dotchart(t(USPersonalExpenditure))
- dotchart(WorldPhones) (transpose is no good)
- freeny.x is no good, neither is volcano
- dotchart(UCBAdmissions{[},,1{]})
- dotplot(Survived ~ Freq | Class, groups = Sex, data = B)
- dotplot(Admit ~ Freq | Dept, groups = Gender, data = C)
- Mosaic plot
- mosaic(~ Survived + Class + Age + Sex, data = Titanic) (or just mosaic(Titanic))
- mosaic(~ Admit + Dept + Gender, data = UCBAdmissions)

- Spine plots
- spineplot(xtabs(Freq ~ Admit + Gender, data = UCBAdmissions))
- rescaled barplot
- Quantile-quantile plots: There are two ways to do this. One way is to compare two independent samples (of the same size). qqplot(x,y). Another way is to compare the sample quantiles of one variable to the theoretical quantiles of another distribution.

Given two samples x_1, x_2, ..., x_n, and y_1, y_2, ..., y_n, we may find the order statistics $x_{(1)} \leq x_{(2)} \leq \cdots \leq x_{(n)}$ and $y_{(1)} \leq y_{(2)} \leq \cdots \leq y_{(n)}$. Next, plot the n points $(x_{(1)}, y_{(1)})$, $(x_{(2)}, y_{(2)})$, ..., $(x_{(n)}, y_{(n)})$.

It is clear that if $x_{(k)} = y_{(k)}$ for all $k = 1, 2, \ldots, n$, then we will have a straight line. It is also clear that in the real world, a straight line is NEVER observed, and instead we have a scatterplot that hopefully had a general linear trend. What do the rules tell us?

- If the y-intercept of the line is greater (less) than zero, then the center of the Y data is greater (less) than the center of the X data.
- If the slope of the line is greater (less) than one, then the spread of the Y data is greater (less) than the spread of the X data.

3.6.3 Lattice Graphics

The following types of plots are useful when there is one variable of interest and there is a factor in the data set by which the variable is categorized.

It is sometimes nice to set `lattice.options(default.theme = "col.whitebg")`

Side by side boxplots

```r
bwplot(~weight | feed, data = chickwts)
```

Histograms

```r
histogram(~age | education, data = infert)
```

Scatterplots

```r
xyplot(Petal.Length ~ Petal.Width | Species, data = iris)
```

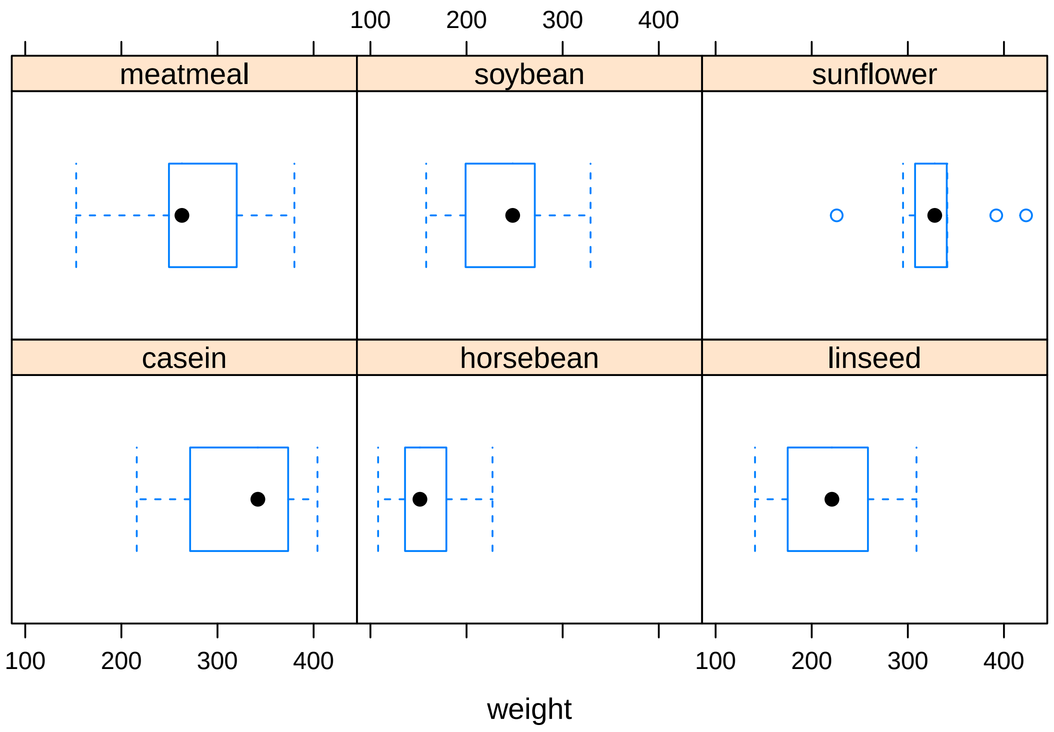


Figure 3.9: Boxplots of `weight` by `feed` type in the `chickwts` data.

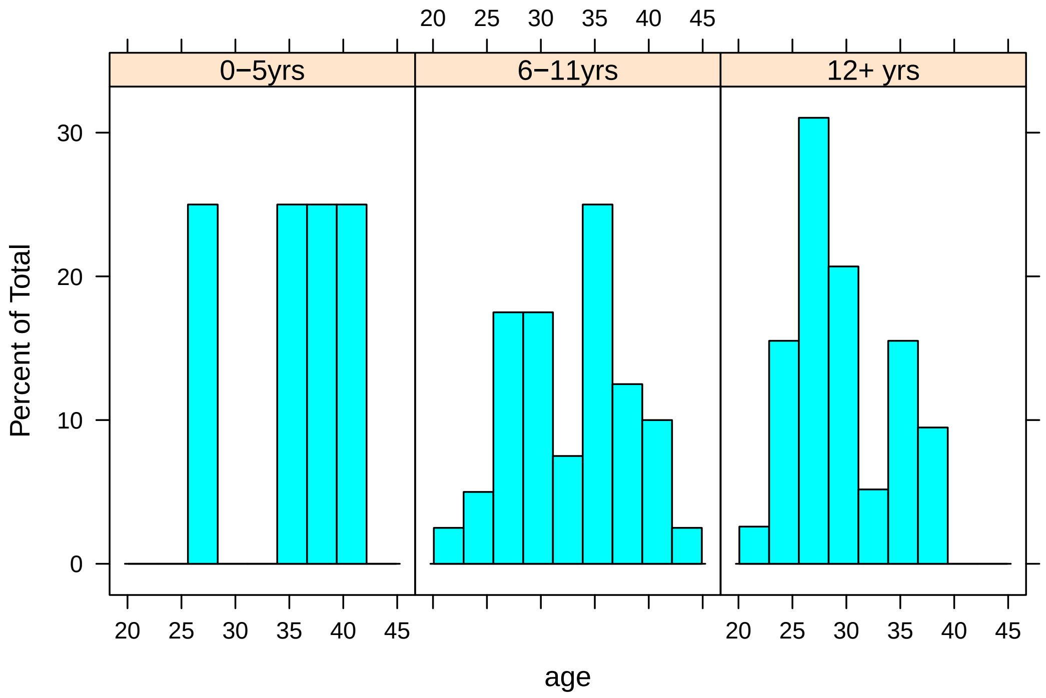


Figure 3.10: Histograms of `age` by `education` level from the `infert` data.

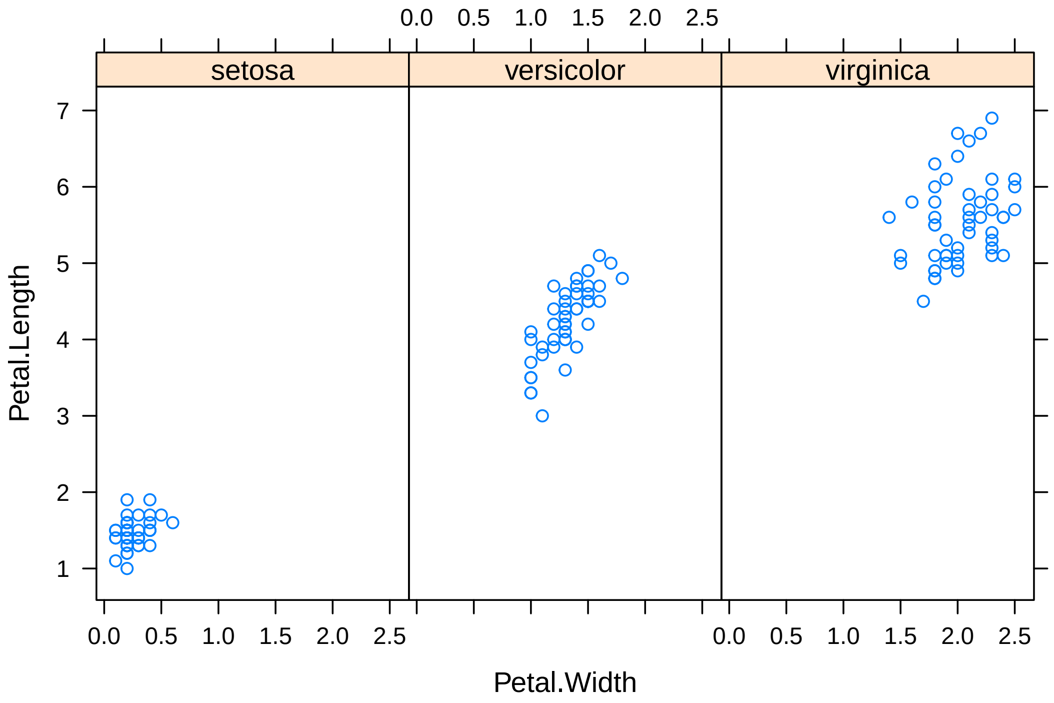


Figure 3.11: An `xyplot` of `Petal.Length` versus `Petal.Width` by `Species` in the `iris` data.

Coplots

```
coplot(conc ~ uptake | Type * Treatment, data = CO2)
```

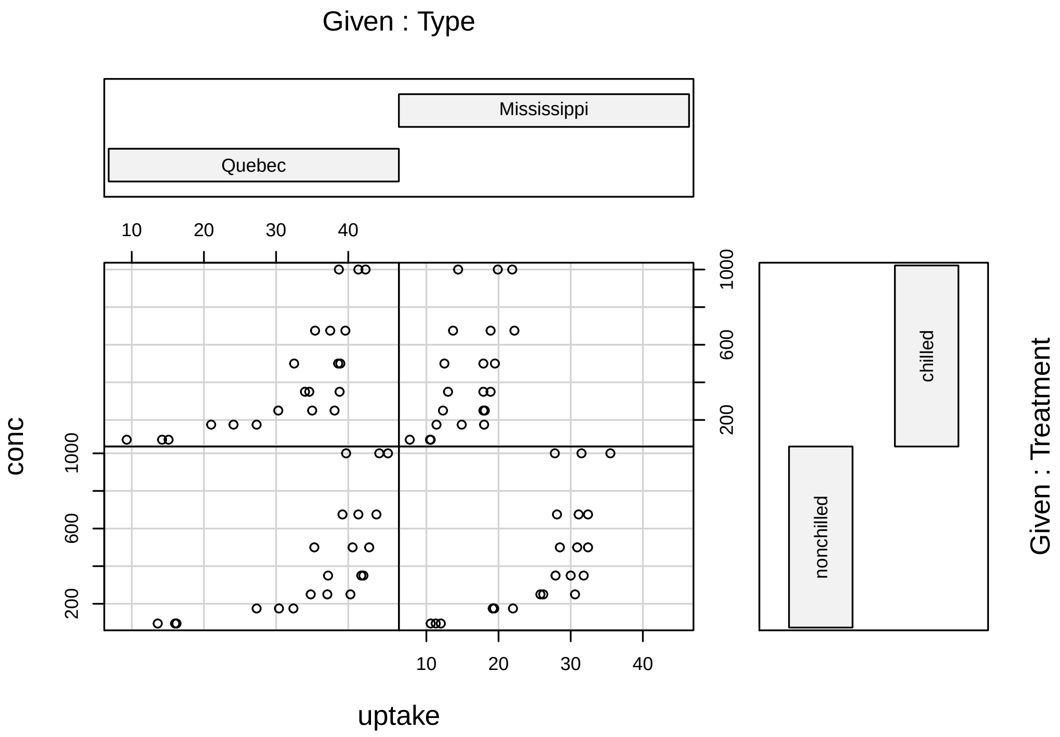


Figure 3.12: A `coplot` of `conc` versus `uptake` by `Type` and `Treatment`.

```
NULL
```

3.7 Some Remarks about Plotting

Getting your labels to look right

```
library(ggplot2)
a <- qplot(state.division, geom = "bar")
a + opts(axis.text.x = theme_text(angle = -90, hjust = 0))
```

```
hist(precip, freq = FALSE)
lines(density(precip))
qplot(precip, geom = "density")
m <- ggplot(as.data.frame(precip), aes(x = precip))
```

```
m + geom_histogram()
m + geom_histogram(aes(y = ..density..)) + geom_density()
```

3.8 Exercises

Open R and issue the following commands at the command line to get started. Note that you need to have the `RcmdrPlugin.IPSUR` package [152] installed, and for some exercises you need the `e1071` package [96].

```
library("RcmdrPlugin.IPSUR")
data(RcmdrTestDrive)
attach(RcmdrTestDrive)
names(RcmdrTestDrive)
```

To load the data in the R Commander (`Rcmdr`), click the `Data Set` button, and select `RcmdrTestDrive` as the active data set. To learn more about the data set and where it comes from, type `?RcmdrTestDrive` at the command line.

Exercise. <> Perform a summary of all variables in `RcmdrTestDrive`. You can do this with the command `summary(RcmdrTestDrive)`.

Alternatively, you can do this in the `Rcmdr` with the sequence `Statistics` ▹ `Summaries` ▹ `Active Data Set`. Report the values of the summary statistics for each variable.

Exercise. Make a table of the `race` variable. Do this with `Statistics` ▹ `Summaries` ▹ `Frequency Distributions - IPSUR...` 1. Which ethnicity has the highest frequency? 1. Which ethnicity has the lowest frequency? 1. Include a bar graph of `race`. Do this with `Graphs` ▹ `IPSUR - Bar Graph...`

Exercise. Calculate the average `salary` by the factor `gender`. Do this with `Statistics` ▹ `Summaries` ▹ `Table of Statistics...` 1. Which `gender` has the highest mean `salary`? 1. Report the highest mean `salary`. 1. Compare the spreads for the genders by calculating the standard deviation of `salary` by `gender`. Which `gender` has the biggest standard deviation? 1. Make boxplots of `salary` by `gender` with the following method: #+BEGIN_quote On the `Rcmdr`, click `Graphs` ▹ `IPSUR - Boxplot...` In the `Variable` box, select `salary`. Click the `Plot by groups...` box and select `gender`. Click `OK`. Click `OK` to graph the boxplot. #+END_quote How does the boxplot compare to your answers to (1) and (3)?

Exercise. For this problem we will study the variable `reduction`. 1. Find the order statistics and store them in a vector `x`. *Hint:* `x <- sort(reduction)` 1. Find $x_{(137)}$, the 137th order statistic. 1. Find the IQR. 1. Find the Five Number Summary (5NS). 1. Use the 5NS to calculate what the width of a boxplot of `reduction` would be. 1. Compare your answers (3) and (5). Are they the same? If not, are they close? 1. Make a boxplot of `reduction`, and include the boxplot in your report. You can do this with the `boxplot` function, or in `Rcmdr` with `Graphs` ▹ `IPSUR - Boxplot...` 1. Are there any potential/suspected outliers? If so, list their values. *Hint:* use your answer to (a). 1. Using the rules discussed in the text, classify answers to (8), if any, as *potential* or *suspected* outliers.

Exercise. In this problem we will compare the variables `before` and `after`. Don't forget `library("e1071")`. 1. Examine the two measures of center for both variables. Judging from these measures, which variable has a higher center? 1. Which measure of center is more appropriate for `before`? (You may want to look at a boxplot.) Which measure of center is more appropriate for `after`? 1. Based on your answer to (2), choose an appropriate measure of spread for each variable, calculate it, and report its value. Which variable has the biggest spread? (Note that you need to make sure that your measures are on the same scale.) 1. Calculate and report the skewness and kurtosis for `before`. Based on these values, how would you describe the shape of `before`? 1. Calculate and report the skewness and kurtosis for `after`. Based on these values, how would you describe the shape of `after`? 1. Plot histograms of `before` and `after` and compare them to your answers to (4) and (5).

Exercise. Describe the following data sets just as if you were communicating with an alien, but one who has had a statistics class. Mention the salient features (data type, important properties, anything special). Support your answers with the appropriate visual displays and descriptive statistics. 1. Conversion rates of Euro currencies stored in `euro`. 2. State abbreviations stored in `state.abb`. 3. Areas of the world's landmasses stored in `islands`. 4. Areas of the 50 United States stored in `state.area`. 5. Region of the 50 United States stored in `state.region`.

Bibliography

[1] Daniel Adler and Duncan Murdoch. *rgl: 3D visualization device system (OpenGL)*. R package version 0.93.952. 2013. URL: http://CRAN.R-project.org/package=rgl.

[2] A. Agresti and B. A. Coull. "Approximate is better than "exact" for interval estimation of binomial proportions". In: *The American Statistician* 52 (1998), pp. 119–126.

[3] Alan Agresti. *Categorical Data Analysis*. Wiley, 2002.

[4] Martin Maechler et al. *sfsmisc: Utilities from Seminar fuer Statistik ETH Zurich*. R package version 1.0-24. 2013. URL: http://CRAN.R-project.org/package=sfsmisc.

[5] Fortran code by Alan Genz and R code by Adelchi Azzalini. *mnormt: The multivariate normal and t distributions*. R package version 1.4-5. 2012. URL: http://CRAN.R-project.org/package=mnormt.

[6] Jim Albert. *LearnBayes: Functions for Learning Bayesian Inference*. R package version 2.12. 2012. URL: http://CRAN.R-project.org/package=LearnBayes.

[7] Dirk Eddelbuettel with contributions by Antoine Lucas et al. *digest: Create cryptographic hash digests of R objects*. R package version 0.6.3. 2013. URL: http://CRAN.R-project.org/package=digest.

[8] Tom M. Apostol. *Calculus*. Second. Vol. I. Wiley, 1967.

[9] Tom M. Apostol. *Calculus*. Second. Vol. II. Wiley, 1967.

[10] Robert B. Ash and Catherine Doleans-Dade. *Probability & Measure Theory*. Harcourt Academic Press, 2000.

[11] Douglas Bates and Martin Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*. R package version 1.0-12. 2013. URL: http://CRAN.R-project.org/package=Matrix.

[12] S original by Berwin A. Turlach R port by Andreas Weingessel <Andreas.Weingessel@ci.tuwien.ac.at>. *quadprog: Functions to solve Quadratic Programming Problems*. R package version 1.5-5. 2013. URL: http://CRAN.R-project.org/package=quadprog.

[13] Peter J. Bickel and Kjell A. Doksum. *Mathematical Statistics*. Vol. I. Prentice Hall, 2001.

[14] Patrick Billingsley. *Probability and Measure*. Wiley Interscience, 1995.

[15] Ben Bolker. *emdbook: Ecological Models and Data in R*. R package version 1.3.4. 2013.

[16] Bruce L. Bowerman, Richard O'Connell, and Anne Koehler. *Forecasting, Time Series, and Regression: An Applied Approach*. South-Western College Pub, 2004.

[17] P. J. Brockwell and R. A. Davis. *Time Series and Forecasting Methods*. Second. Springer, 1991, p. 555.

[18] Angelo Canty and B. D. Ripley. *boot: Bootstrap R (S-Plus) Functions*. R package version 1.3-9. 2013.

[19] Neal L. Carothers. *Real Analysis*. Cambridge University Press, 2000.

[20] George Casella and Roger L. Berger. *Statistical Inference*. Duxbury Press, 2002.

[21] Kung-Sik Chan and Brian Ripley. *TSA: Time Series Analysis*. R package version 1.01. 2012. URL: http://CRAN.R-project.org/package=TSA.

[22] Scott Chasalow. *combinat: combinatorics utilities*. R package version 0.0-8. 2012. URL: http://CRAN.R-project.org/package=combinat.

[23] S original by Chris Fraley et al. *fracdiff: Fractionally differenced ARIMA aka ARFIMA(p,d,q) models*. R package version 1.4-2. 2012. URL: http://CRAN.R-project.org/package=fracdiff.

[24] Erhan Cinlar. *Introduction to Stochastic Processes*. Prentice Hall, 1975.

[25] William S. Cleveland. *The Elements of Graphing Data*. Hobart Press, 1994.

[26] Yves Croissant and Giovanni Millo. “Panel Data Econometrics in R: The plm Package”. In: *Journal of Statistical Software* 27.2 (2008). URL: http://www.jstatsoft.org/v27/i02/.

[27] David B. Dahl. *xtable: Export tables to LaTeX or HTML*. R package version 1.7-1. 2013. URL: http://CRAN.R-project.org/package=xtable.

[28] Peter Dalgaard. *Introductory Statistics with R*. Springer, 2008. URL: http://staff.pubhealth.ku.dk/~pd/ISwR.html.

[29] A. C. Davison and D. V. Hinkley. *Bootstrap Methods and Their Applications*. ISBN 0-521-57391-2. Cambridge: Cambridge University Press, 1997. URL: http://statwww.epfl.ch/davison/BMA/.

[30] A. C. Davison and D. V. Hinkley. *Bootstrap Methods and Their Applications*. Cambridge University Press, 1997.

[31] Thomas J. DiCiccio and Bradley Efron. “Bootstrap Confidence Intervals”. In: *Statistical Science* 11 (1996), pp. 189–228.

[32] M Dowle, T Short, and S Lianoglou. *data.table: Extension of data.frame for fast indexing, fast ordered joins, fast assignment, fast grouping and list columns.* R package version 1.8.8. 2013. URL: http://CRAN.R-project.org/package=data.table.

[33] Richard Durrett. *Probability: Theory and Examples*. Duxbury Press, 1996.

[34] Rick Durrett. *Essentials of Stochastic Processes*. Springer, 1999.

[35] Christophe Dutang, Vincent Goulet, and Mathieu Pigeon. “actuar: An R Package for Actuarial Science”. In: *Journal of Statistical Software* 25.7 (2008), p. 38. URL: http://www.jstatsoft.org/v25/i07.

[36] Dirk Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. ISBN 978-1-4614-6867-7. New York: Springer, 2013.

[37] Dirk Eddelbuettel and Romain François. “Rcpp: Seamless R and C++ Integration”. In: *Journal of Statistical Software* 40.8 (2011), pp. 1–18. URL: http://www.jstatsoft.org/v40/i08/.

[38] Dirk Eddelbuettel and Conrad Sanderson. "RcppArmadillo: Accelerating R with high-performance C++ linear algebra". In: *Computational Statistics and Data Analysis* in press (2013). URL: http://dx.doi.org/10.1016/j.csda.2013.02.005.

[39] Brian Everitt. *An R and S-Plus Companion to Multivariate Analysis*. Springer, 2007.

[40] Gerald B. Folland. *Real Analysis: Modern Techniques and Their Applications*. Wiley, 1999.

[41] Thomas Lumley using Fortran code by Alan Miller. *leaps: regression subset selection*. R package version 2.9. 2009. URL: http://CRAN.R-project.org/package=leaps.

[42] John Fox. *An R and S Plus Companion to Applied Regression*. Sage, 2002.

[43] John Fox. *Applied Regression Analysis, Linear Models, and Related Methods*. Sage, 1997.

[44] John Fox. "Effect Displays in R for Generalised Linear Models". In: *Journal of Statistical Software* 8.15 (2003), pp. 1–27. URL: http://www.jstatsoft.org/v08/i15/.

[45] John Fox. "The R Commander: A Basic Statistics Graphical User Interface to R". In: *Journal of Statistical Software* 14.9 (2005), pp. 1–42. URL: http://www.jstatsoft.org/v14/i09.

[46] John Fox and Jangman Hong. "Effect Displays in R for Multinomial and Proportional-Odds Logit Models: Extensions to the effects Package". In: *Journal of Statistical Software* 32.1 (2009), pp. 1–24. URL: http://www.jstatsoft.org/v32/i01/.

[47] John Fox, Zhenghua Nie, and Jarrett Byrnes. *sem: Structural Equation Models*. R package version 3.1-3. 2013. URL: http://CRAN.R-project.org/package=sem.

[48] John Fox and Sanford Weisberg. *An R Companion to Applied Regression*. Second. Thousand Oaks CA: Sage, 2011. URL: http://socserv.socsci.mcmaster.ca/jfox/Books/Companion.

[49] Michael Friendly. *Visualizing Categorical Data*. SAS Publishing, 2000.

[50] Andrew Gelman et al. *Bayesian Data Analysis*. CRC Press, 2004.

[51] Alan Genz and Frank Bretz. *Computation of Multivariate Normal and t Probabilities*. Lecture Notes in Statistics. Heidelberg: Springer-Verlag, 2009. ISBN: 978-3-642-01688-2.

[52] Alan Genz et al. *mvtnorm: Multivariate Normal and t Distributions*. R package version 0.9-9995. 2013. URL: http://CRAN.R-project.org/package=mvtnorm.

[53] Rob J Hyndman with contributions from George Athanasopoulos et al. *forecast: Forecasting functions for time series and linear models*. R package version 4.06. 2013. URL: http://CRAN.R-project.org/package=forecast.

[54] Gregor Gorjanc. "Using Sweave with LyX". In: *R News* 1 (2008), pp. 2–9.

[55] Charles M. Grinstead and J. Laurie Snell. *Introduction to Probability*. American Mathematical Society, 1997. URL: http://www.dartmouth.edu/~chance/.

[56] Bettina Grün and Achim Zeileis. "Automatic Generation of Exams in R". In: *Journal of Statistical Software* 29.10 (2009), pp. 1–14. URL: http://www.jstatsoft.org/v29/i10/.

[57] David Firth with contributions from Heather Turner. *relimp: Relative Contribution of Effects in a Regression Model*. R package version 1.0-3. 2011. URL: http://CRAN.R-project.org/package=relimp.

[58] Richard M. Heiberger. *HH: Statistical Analysis and Data Display: Heiberger and Holland*. R package version 2.3-37. 2013. URL: http://CRAN.R-project.org/package=HH.

[59] Richard M. Heiberger and Burt Holland. *Statistical Analysis and Data Display: An Intermediate Course with Examples in S-Plus, R, and SAS*. Springer, 2004. URL: http://astro.temple.edu/~rmh/HH/.

[60] Richard M. Heiberger and Erich Neuwirth. *R Through Excel: A Spreadsheet Interface for Statistics, Data Analysis, and Graphics*. Springer, 2009. URL: http://www.springer.com/statistics/computanional+statistics/book/978-1-4419-0051-7.

[61] Robert V. Hogg, Joseph W. McKean, and Allen T. Craig. *Introduction to Mathematical Statistics*. Pearson Prentice Hall, 2005.

[62] Robert V. Hogg and Elliot A. Tanis. *Probability and Statistical Inference*. Pearson Prentice Hall, 2006.

[63] Torsten Hothorn, Frank Bretz, and Peter Westfall. "Simultaneous Inference in General Parametric Models". In: *Biometrical Journal* 50.3 (2008), pp. 346–363.

[64] Torsten Hothorn and Kurt Hornik. *exactRankTests: Exact Distributions for Rank and Permutation Tests*. R package version 0.8-25. 2013. URL: http://CRAN.R-project.org/package=exactRankTests.

[65] Torsten Hothorn, Friedrich Leisch, and Achim Zeileis. *modeltools: Tools and Classes for Statistical Models*. R package version 0.2-19. 2012. URL: http://CRAN.R-project.org/package=modeltools.

[66] Torsten Hothorn et al. "A Lego System for Conditional Inference". In: *The American Statistician* 60.3 (2006), pp. 257–263.

[67] Torsten Hothorn et al. "Implementing a Class of Permutation Tests: The coin Package". In: *Journal of Statistical Software* 28.8 (2008), pp. 1–23. URL: http://www.jstatsoft.org/v28/i08/.

[68] Ross Ihaka et al. *colorspace: Color Space Manipulation*. R package version 1.2-2. 2013. URL: http://CRAN.R-project.org/package=colorspace.

[69] Norman L. Johnson, Samuel Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*. Second. Vol. 1. Wiley, 1994.

[70] Norman L. Johnson, Samuel Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*. Second. Vol. 2. Wiley, 1995.

[71] Norman L. Johnson, Samuel Kotz, and N. Balakrishnan. *Discrete Multivariate Distributions*. Wiley, 1997.

[72] Norman L. Johnson, Samuel Kotz, and Adrienne W. Kemp. *Univariate Discrete Distributions*. Second. Wiley, 1993.

[73] Roger W. Johnson. *How Many Fish are in the Pond?* URL: http://www.rsscse.org.uk/ts/gtb/johnson3.pdf.

[74] Frank E Harrell Jr, with contributions from Charles Dupont, and many others. *Hmisc: Harrell Miscellaneous*. R package version 3.12-2. 2013. URL: http://CRAN.R-project.org/package=Hmisc.

[75] Louis Kates and Thomas Petzoldt. *proto: Prototype object-based programming*. R package version 0.3-10. 2012. URL: http://CRAN.R-project.org/package=proto.

[76] G. Jay Kerns. *IPSUR: Introduction to Probability and Statistics Using R*. R package version 1.4. 2012. URL: http://CRAN.R-project.org/package=IPSUR.

[77] G. Jay Kerns. *prob: Elementary Probability on Finite Sample Spaces*. R package version 0.9-2. 2009. URL: http://CRAN.R-project.org/package=prob.

[78] Samuel Kotz, N. Balakrishnan, and Norman L. Johnson. *Continuous Multivariate Distributions*. Second. Vol. 1: Models and Applications. Wiley, 2000.

[79] Duncan Temple Lang. *XML: Tools for parsing and generating XML within R and S-Plus.* R package version 3.98-1.1. 2013. URL: http://CRAN.R-project.org/package=XML.

[80] Michael Lavine. *Introduction to Statistical Thought*. Lavine, Michael, 2009. URL: http://www.math.umass.edu/~lavine/Book/book.html.

[81] Peter M. Lee. *Bayesian Statistics: An Introduction*. Wiley, 1997.

[82] E. L. Lehmann. *Testing Statistical Hypotheses*. Springer-Verlag, 1986.

[83] E. L. Lehmann and George Casella. *Theory of Point Estimation*. Springer, 1998.

[84] Jim Lemon and Philippe Grosjean. *prettyR: Pretty descriptive stats.* R package version 2.0-7. 2013. URL: http://CRAN.R-project.org/package=prettyR.

[85] Andy Liaw and Matthew Wiener. “Classification and Regression by randomForest”. In: *R News* 2.3 (2002), pp. 18–22. URL: http://CRAN.R-project.org/doc/Rnews/.

[86] Uwe Ligges. “Accessing the Sources”. In: *R News* 6 (2006), pp. 43–45.

[87] Uwe Ligges and Martin Mächler. “Scatterplot3d - an R Package for Visualizing Multivariate Data”. In: *Journal of Statistical Software* 8.11 (2003), pp. 1–20. URL: http://www.jstatsoft.org.

[88] Catherine Loader. *locfit: Local Regression, Likelihood and Density Estimation.* R package version 1.5-9.1. 2013. URL: http://CRAN.R-project.org/package=locfit.

[89] Thomas Lumley. *dichromat: Color Schemes for Dichromats*. R package version 2.0-0. 2013. URL: http://CRAN.R-project.org/package=dichromat.

[90] Martin Maechler et al. *cluster: Cluster Analysis Basics and Extensions*. R package version 1.14.4 — For new features, see the ’Changelog’ file (in the package source). 2013.

[91] Jan R. Magnus and Heinz Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Wiley, 1999.

[92] John Maindonald and John Braun. *Data Analysis and Graphics Using R*. Cambridge University Press, 2003.

[93] John Maindonald and W. John Braun. *DAAG: Data Analysis And Graphics data and functions*. R package version 1.16. 2013. URL: http://CRAN.R-project.org/package=DAAG.

[94] David Meyer, Achim Zeileis, and Kurt Hornik. “The Strucplot Framework: Visualizing Multi-Way Contingency Tables with vcd”. In: *Journal of Statistical Software* 17.3 (2006), pp. 1–48. URL: http://www.jstatsoft.org/v17/i03/.

[95] David Meyer, Achim Zeileis, and Kurt Hornik. *vcd: Visualizing Categorical Data*. R package version 1.2-13. 2012.

[96] David Meyer et al. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*. R package version 1.6-1. 2012. URL: http://CRAN.R-project.org/package=e1071.

[97] Ben Mezrich. *Bringing Down the House: The Inside Story of Six M.I.T. Students Who Took Vegas for Millions*. Free Press, 2003.

[98] Jeff Miller. *Earliest Known Uses of Some of the Words of Mathematics*. URL: http://jeff560.tripod.com/mathword.html.

[99] John Neter et al. *Applied Linear Regression Models*. Third. McGraw Hill, 1996.

[100] Erich Neuwirth. *RColorBrewer: ColorBrewer palettes*. R package version 1.0-5. 2011. URL: http://CRAN.R-project.org/package=RColorBrewer.

[101] Frederick Novomestky. *matrixcalc: Collection of functions for matrix calculations*. R package version 1.0-3. 2012. URL: http://CRAN.R-project.org/package=matrixcalc.

[102] Jari Oksanen et al. *vegan: Community Ecology Package*. R package version 2.0-8. 2013. URL: http://CRAN.R-project.org/package=vegan.

[103] Jose Pinheiro et al. *nlme: Linear and Nonlinear Mixed Effects Models*. R package version 3.1-110. 2013.

[104] Tony Plate and Richard Heiberger. *abind: Combine multi-dimensional arrays*. R package version 1.4-0. 2011. URL: http://CRAN.R-project.org/package=abind.

[105] R Core Team. *foreign: Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, dBase, ...* R package version 0.8-54. 2013. URL: http://CRAN.R-project.org/package=foreign.

[106] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[107] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[108] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[109] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[110] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[111] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[112] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[113] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[114] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[115] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[116] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[117] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[118] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[119] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/.

[120] R Development Core Team. *R: A Language and Environment for Statistical Computing*. ISBN 3-900051-07-0. R Foundation for Statistical Computing. Vienna, Austria, 2009. URL: http://www.R-project.org.

[121] C. Radhakrishna Rao and Helge Toutenburg. *Linear Models: Least Squares and Alternatives*. Springer, 1999.

[122] Sidney I. Resnick. *A Probability Path*. Birkhauser, 1999.

[123] Brian Ripley and from 1999 to Oct 2002 Michael Lapsley. *RODBC: ODBC Database Access*. R package version 1.3-7. 2013. URL: http://CRAN.R-project.org/package=RODBC.

[124] Maria L. Rizzo. *Statistical Computing with R*. Chapman & Hall/CRC, 2008.

[125] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer, 2004.

[126] Kenneth A. Ross. *Elementary Calculus: The Theory of Calculus*. Springer, 1980.

[127] RStudio. *manipulate: Interactive Plots for RStudio*. R package version 0.97.551. 2011.

[128] RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, Inc. Boston, MA, 2012. URL: http://www.rstudio.com/.

[129] Peter Ruckdeschel. *startupmsg: Utilities for start-up messages*. R package version 0.8. 2013. URL: http://CRAN.R-project.org/package=startupmsg.

[130] Peter Ruckdeschel. *SweaveListingUtils: Utilities for Sweave together with TeX listings package*. R package version 0.6.1. 2013. URL: http://CRAN.R-project.org/package=SweaveListingUtils.

[131] P. Ruckdeschel et al. "S4 Classes for Distributions". English. In: *R News* 6.2 (May 2006), pp. 2–6. URL: http://www.uni-bayreuth.de/departments/math/org/mathe7/DISTR/distr.pdf.

[132] P. Ruckdeschel et al. "S4 Classes for Distributions". English. In: *R News* 6.2 (May 2006), pp. 2–6. URL: http://www.uni-bayreuth.de/departments/math/org/mathe7/DISTR/distr.pdf.

[133] P. Ruckdeschel et al. "S4 Classes for Distributions". English. In: *R News* 6.2 (May 2006), pp. 2–6. URL: http://www.uni-bayreuth.de/departments/math/org/mathe7/DISTR/distr.pdf.

[134] P. Ruckdeschel et al. *S4 Classes for Distributions—a manual for packages distr, distrSim, distrTEst, distrEx, distrMod, and distrTeach*. English. Tech. rep. Kaiserslautern, Germany: Fraunhofer ITWM, July 2008. URL: http://r-forge.r-project.org/plugins/scmsvn/viewcvs.php/*checkout*/pkg/distrDoc/inst/doc/distr.pdf?root=distr.

[135] Deepayan Sarkar. *Lattice: Multivariate Data Visualization with R*. ISBN 978-0-387-75968-5. New York: Springer, 2008. URL: http://lmdvr.r-forge.r-project.org.

[136] Deepayan Sarkar and Felix Andrews. *latticeExtra: Extra Graphical Utilities Based on Lattice*. R package version 0.6-24. 2012. URL: http://CRAN.R-project.org/package=latticeExtra.

[137] F. E. Satterthwaite. "An Approximate Distribution of Estimates of Variance Components". In: *Biometrics Bulletin* 2 (1946), pp. 110–114.

[138] Luca Scrucca. “qcc: an R package for quality control charting and statistical process control”. In: *R News* 4/1 (2004), pp. 11–17. URL: http://CRAN.R-project.org/doc/Rnews/.

[139] Robert J. Serfling. *Approximation Theorems of Mathematical Statistics*. Wiley, 1980.

[140] S. S. Shapiro and M. B. Wilk. “An analysis of variance test for normality (complete samples)”. In: *Biometrika* 52 (1965), pp. 591–611.

[141] Gavin L. Simpson. *permute: Functions for generating restricted permutations of data*. R package version 0.7-0. 2012. URL: http://CRAN.R-project.org/package=permute.

[142] Greg Snow. *TeachingDemos: Demonstrations for teaching and learning*. R package version 2.9. 2013. URL: http://CRAN.R-project.org/package=TeachingDemos.

[143] Karline Soetaert. *diagram: Functions for visualising simple graphs (networks), plotting flow diagrams*. R package version 1.6.1. 2013. URL: http://CRAN.R-project.org/package=diagram.

[144] Karline Soetaert. *shape: Functions for plotting graphical shapes, colors*. R package version 1.4.0. 2012. URL: http://CRAN.R-project.org/package=shape.

[145] Max Kuhn. Contributions from Steve Weston et al. *odfWeave: Sweave processing of Open Document Format (ODF) files*. R package version 0.8.2. 2012. URL: http://CRAN.R-project.org/package=odfWeave.

[146] James Stewart. *Calculus*. Thomson Brooks/Cole, 2008.

[147] Stephen M. Stigler. *The History of Statistics: The Measurement of Uncertainty before 1900*. Harvard University Press, 1986.

[148] Gilbert Strang. *Linear Algebra and Its Applications*. Harcourt, 1988.

[149] Barbara G. Tabachnick and Linda S. Fidell. *Using Multivariate Statistics*. Allyn and Bacon, 2006.

[150] Justin Talbot. *labeling: Axis Labeling*. R package version 0.2. 2013. URL: http://CRAN.R-project.org/package=labeling.

[151] Terry M. Therneau and Patricia M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. New York: Springer, 2000. ISBN: 0-387-98784-3.

[152] G. Jay Kerns with contributions by Theophilius Boye, Tyler Drombosky, and adapted from the work of John Fox et al. *RcmdrPlugin.IPSUR: An IPSUR Plugin for the R Commander*. R package version 0.1-8. 2012. URL: http://CRAN.R-project.org/package=RcmdrPlugin.IPSUR.

[153] Terry Therneau. *bdsmatrix: Routines for Block Diagonal Symmetric matrices*. R package version 1.3-1. 2013. URL: http://CRAN.R-project.org/package=bdsmatrix.

[154] Terry M Therneau. *A Package for Survival Analysis in S*. R package version 2.37-4. 2013. URL: http://CRAN.R-project.org/package=survival.

[155] Terry Therneau, Beth Atkinson, and Brian Ripley. *rpart: Recursive Partitioning*. R package version 4.1-1. 2013.

[156] Luke Tierney. *codetools: Code Analysis Tools for R*. R package version 0.2-8. 2011. URL: http://CRAN.R-project.org/package=codetools.

[157] Luke Tierney. *tkrplot: TK Rplot*. R package version 0.0-23. 2011. URL: http://CRAN.R-project.org/package=tkrplot.

[158] Adrian Trapletti and Kurt Hornik. *tseries: Time Series Analysis and Computational Finance*. R package version 0.10-32. 2013. URL: http://CRAN.R-project.org/package=tseries.

[159] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Fourth. ISBN 0-387-95457-0. New York: Springer, 2002. URL: http://www.stats.ox.ac.uk/pub/MASS4.

[160] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Fourth. ISBN 0-387-95457-0. New York: Springer, 2002. URL: http://www.stats.ox.ac.uk/pub/MASS4.

[161] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Fourth. ISBN 0-387-95457-0. New York: Springer, 2002. URL: http://www.stats.ox.ac.uk/pub/MASS4.

[162] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Fourth. ISBN 0-387-95457-0. New York: Springer, 2002. URL: http://www.stats.ox.ac.uk/pub/MASS4.

[163] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Fourth. ISBN 0-387-95457-0. New York: Springer, 2002. URL: http://www.stats.ox.ac.uk/pub/MASS4.

[164] William N. Venables and David M. Smith. *An Introduction to R*. 2010. URL: http://www.r-project.org/Manuals.

[165] John Verzani. *Using R for Introductory Statistics*. CRC Press, 2005. URL: http://www.math.csi.cuny.edu/UsingR/.

[166] John Verzani. *UsingR: Data sets for the text "Using R for Introductory Statistics"*. R package version 0.1-18. 2012. URL: http://CRAN.R-project.org/package=UsingR.

[167] Matt Wand. *KernSmooth: Functions for kernel smoothing for Wand and Jones (1995)*. R package version 2.23-10. 2013. URL: http://CRAN.R-project.org/package=KernSmooth.

[168] B. L. Welch. “The generalization of "Student’s" problem when several different population variances are involved”. In: *Biometrika* 34 (1947), pp. 28–35.

[169] Charlotte Wickham. *munsell: Munsell colour system*. R package version 0.4.2. 2013. URL: http://CRAN.R-project.org/package=munsell.

[170] Hadley Wickham. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009. ISBN: 978-0-387-98140-6. URL: http://had.co.nz/ggplot2/book.

[171] Hadley Wickham. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009. ISBN: 978-0-387-98140-6. URL: http://had.co.nz/ggplot2/book.

[172] Hadley Wickham. *gtable: Arrange grobs in tables.* R package version 0.1.2. 2012. URL: http://CRAN.R-project.org/package=gtable.

[173] Hadley Wickham. “Reshaping Data with the reshape Package”. In: *Journal of Statistical Software* 21.12 (2007), pp. 1–20. URL: http://www.jstatsoft.org/v21/i12/.

[174] Hadley Wickham. *scales: Scale functions for graphics.* R package version 0.2.3. 2012. URL: http://CRAN.R-project.org/package=scales.

[175] Hadley Wickham. *stringr: Make it easier to work with strings.* R package version 0.6.2. 2012. URL: http://CRAN.R-project.org/package=stringr.

[176] Hadley Wickham. “The Split-Apply-Combine Strategy for Data Analysis”. In: *Journal of Statistical Software* 40.1 (2011), pp. 1–29. URL: http://www.jstatsoft.org/v40/i01/.

[177] Wickham and Hadley. “Reshaping data with the reshape package”. In: *Journal of Statistical Software* 21.12 (2007). URL: http://www.jstatsoft.org/v21/i12/paper.

[178] Peter Wolf and Uni Bielefeld. *aplpack: Another Plot PACKage: stem.leaf, bagplot, faces, spin3R, and some slider functions*. R package version 1.2.7. 2012. URL: http://CRAN.R-project.org/package=aplpack.

[179] S. N. Wood. “Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models”. In: *Journal of the Royal Statistical Society (B)* 73.1 (2011), pp. 3–36.

[180] S. N. Wood. “Modelling and smoothing parameter estimation with multiple quadratic penalties”. In: *Journal of the Royal Statistical Society (B)* 62.2 (2000), pp. 413–428.

[181] S. N. Wood. “Stable and efficient multiple smoothing parameter estimation for generalized additive models”. In: *Journal of the American Statistical Association* 99.467 (2004), pp. 673–686.

[182] S. N. Wood. “Thin-plate regression splines”. In: *Journal of the Royal Statistical Society (B)* 65.1 (2003), pp. 95–114.

[183] S.N Wood. *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC, 2006.

[184] Achim Zeileis. “Econometric Computing with HC and HAC Covariance Matrix Estimators”. In: *Journal of Statistical Software* 11.10 (2004), pp. 1–17. URL: http://www.jstatsoft.org/v11/i10/.

[185] Achim Zeileis. “Object-oriented Computation of Sandwich Estimators”. In: *Journal of Statistical Software* 16.9 (2006), pp. 1–16. URL: http://www.jstatsoft.org/v16/i09/..

[186] Achim Zeileis and Yves Croissant. “Extended Model Formulas in R: Multiple Parts and Multiple Responses”. In: *Journal of Statistical Software* 34.1 (2010), pp. 1–13. URL: http://www.jstatsoft.org/v34/i01/.

[187] Achim Zeileis and Gabor Grothendieck. “zoo: S3 Infrastructure for Regular and Irregular Time Series”. In: *Journal of Statistical Software* 14.6 (2005), pp. 1–27. URL: http://www.jstatsoft.org/v14/i06/.

[188] Achim Zeileis, Kurt Hornik, and Paul Murrell. “Escaping RGBland: Selecting Colors for Statistical Graphics”. In: *Computational Statistics & Data Analysis* 53 (2009), pp. 3259–3270. DOI: 10.1016/j.csda.2008.11.033.

[189] Achim Zeileis and Torsten Hothorn. “Diagnostic Checking in Regression Relationships”. In: *R News* 2.3 (2002), pp. 7–10. URL: http://CRAN.R-project.org/doc/Rnews/.

[190] Achim Zeileis, David Meyer, and Kurt Hornik. “Residual-based Shadings for Visualizing (Conditional) Independence”. In: *Journal of Computational and Graphical Statistics* 16.3 (2007), pp. 507–525.

Index