

Introduction to R

Much of the content here is from Appendix A of my *Analysis of Categorical Data with R* book (www.chrisbilder.com/categorical). All R code is available in AppendixInitialExamples.R from my course website.

Background

R is a statistical software package that shares many similarities with the statistical programming language named S. A preliminary version of S was created by Bell Labs in the 1970s that was meant to be a programming language like C but for statistics. John Chambers was one of the primary inventors for the language, and he won the Association for Computing Machinery Award in 1999 for it. A nice video interview with John Chambers about the early days of S is available at <https://www.youtube.com/watch?v=jk9S3RTA138>.

By the 1980s, the popularity of S was growing outside of Bell Labs. This led to Statistical Sciences Inc. creating the S-Plus software package based on S in 1988 and then *selling* it to users. In 1993, Statistical Sciences merges with a company named Mathsoft and buys the exclusive license to market S. This prevents anyone from obtaining a *free* version from Bell Labs. After further company merges, S-Plus now resides with Tibco Software Inc.

New Zealand-based researchers, Ross Ihaka and Robert Gentleman, started to develop a new *free* software package named R that was “not unlike S” in the mid-1990s. They published a paper on R in the *Journal of Computational and Graphical Statistics* in 1996. Version 1.0.0 of R was released on February 29, 2000. Since this time, the use of R has skyrocketed. Interesting historical notes about it include:

- 2001: I first hear about R early this year!
- 2004: The first UseR! conference is held, and the non-profit R Foundation is formed. The conference is now held annually with its location alternating between the US and Europe each year.
- 2004: During a Joint Statistical Meetings (JSM) session that I attended, a SPSS executive says his company and other statistical software companies have felt R’s impact and they are changing their business model.
- 2004: Version 2.0.0 was released.
- 2007: Revolution Analytics was founded to sell a version of R that focuses on big data and parallel processing applications; the company was purchased by Microsoft in 2015.
- 2008: The editor for the *Journal of the American Statistical Association* says during a JSM presentation that R has become the de facto statistical software package for researchers.
- 2009: A New York Times article “Data Analysts Captivated by R’s Power” is published on January 6.¹ The article contains the now famous quote by Anne Milley of SAS:

I think it addresses a niche market for high-end data analysts that want free, readily available code. We have customers who build engines for aircraft. I am happy they are not using freeware when I get on a jet.

Milley later says that she should not have made the airplane comment. SAS starts promoting its ability to call R programs from `proc iml` later this same year.

- 2009: The first issue of the *R Journal* (<http://journal.r-project.org>) is published in May, and it replaces “R News”.

¹http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html?_r=3&enc=eta1

- 2013: Version 3.0.0 was released.
- 2015: The R Consortium is founded to support the R Foundation (<https://www.r-consortium.org>).²

Note that Chambers (2010) and Venables and Ripley (2000) provided much of the pre-2000 history above.

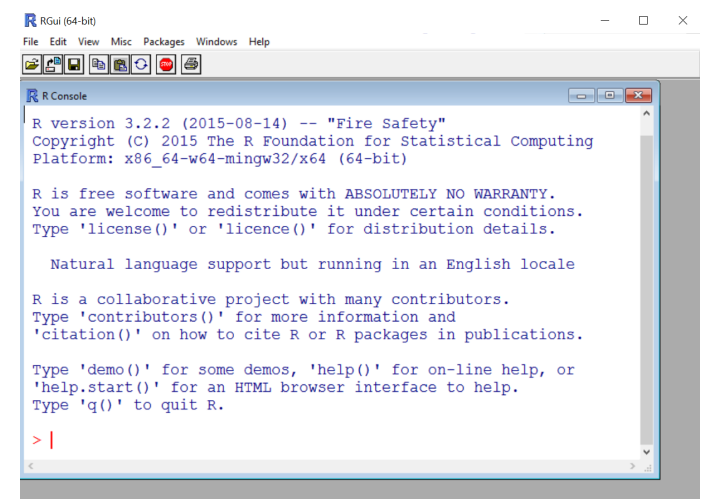
Installation

The R installation file for Windows can be downloaded from the Comprehensive R Archive Network (CRAN) at <http://cran.r-project.org/bin/windows/base>. Select the “Download R 3.*.* for Windows” link. You can simply execute the file on your computer to install (all the installation defaults are o.k. to use). Both a 32-bit and 64-bit version of R will be installed.

Basics

The R Console window is where commands are typed.

²<http://blog.revolutionanalytics.com/2015/06/r-consortium.html>



The Console can be used like a calculator. Below are some examples:

```

> 2 + 2
[1] 4
> qchisq(0.95, 1)
[1] 3.841
> pnorm(1.96)
[1] 0.975
> (2 - 3)/6
[1] -0.1667
> 2^2
[1] 4
> sin(pi/2)
[1] 1
> log(1)

```

```
[1] 0
> 1 > 2
[1] FALSE
> 2 > 1
[1] TRUE
```

Results from these calculations can be stored in an *object*. The combination of a less than sign and a dash as `<-` is used to make the assignment and is read as “gets”.

```
> save <- 2 + 2
> save
[1] 4
```

The objects are stored in R’s database. When you close R you will be asked if you would like to save or delete them. This is kind of like the SAS WORK library, but R gives you the choice to save them. To see a listing of the objects, you can do either of the following:

```
> ls()
[1] "save"
> objects()
[1] "save"
```

To delete an object, use `rm()` and insert the object name in the parentheses.

Functions

R performs calculations using functions. For example, the `qchisq()` and the `pnorm()` commands used earlier are functions. Writing your own function is fairly simple. For example, suppose you want to write a function to calculate the standard deviation. Below is an example where 5 observations are saved to an object using the *concatenate* or *combine* function `c()`. A function called

`sd2()` is written to find the standard deviation simply by using the square root of the variance. The `sd2` object is now stored in the R database.

```
> x <- c(1, 2, 3, 4, 5)
> sd2 <- function(numbers) {
  sqrt(var(numbers))
}
> sd2(x)
[1] 1.581
```

Note that there already is a function in R to calculate the standard deviation, and this function is `sd()`.

When a function has multiple lines of code in it, the last line corresponds to the returned value. For example,

```
> x <- c(1, 2, 3, 4, 5)
> sd2 <- function(numbers) {
  cat("Print the data \n", numbers, "\n")
  sqrt(var(numbers))
}
> save <- sd2(x)
Print the data
1 2 3 4 5
> save
[1] 1.581
```

Note that the `cat()` function is used to print text and the `\n` character tells R to go to a new line.

Help

To see a listing of all R functions which are “built in”, open the Help by selecting `HELP > HTML HELP` from the main R menu bar.

Under REFERENCE, select the link called PACKAGES. All built in R functions are stored in a package.

We have been using functions from the `base` and `stats` package. By selecting `stats`, you can scroll down to find help on the `pnorm()` function. Note the full syntax for `pnorm()` is

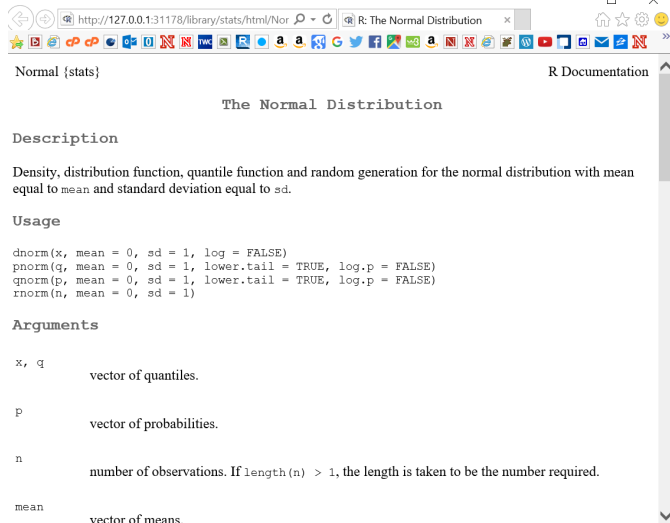
```
pnorm(q, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)
```

The `q` value corresponds to the 1.96 that was entered earlier. Then

```
> pnorm(1.96)
[1] 0.975
> pnorm(q = 1.96)
[1] 0.975
> pnorm(q = 1.96, mean = 0, sd = 1)
[1] 0.975
```

all produce the same results. The other entries in the function have default values set. For example, R assumes you want to work with the standard normal distribution by assigning `mean = 0` and `sd = 1` (standard deviation).

If you know the exact name of the function, simply type `help(<function name>)` at the R Console command prompt to open its help. For example, `help(pnorm)` results in



Using R functions on vectors

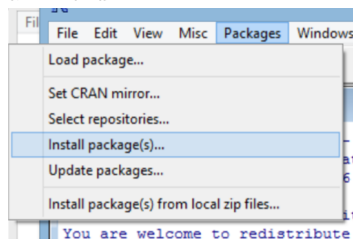
Many R functions are set up to work directly on vectors. For example,

```
> pnorm(q = c(-1.96, 1.96))
[1] 0.025 0.975
```

calculations like this so that unintended results do not occur.

Packages

If you want to use functions that are in other packages, you may need to install and then load the package into R. For example, we will be using the `car` package later in the course. While in the R console, select `PACKAGES > INSTALL PACKAGE(S)` from the main menu.



A number of locations around the world will come up. Choose one close to you. Next, the list of packages will appear. Select the `car` package and select OK. The package will now be installed onto your computer. This only needs to be done once per computer. To load the package into your current R session, type `library(package = car)` at the R Console prompt. This needs to be done only once in an R session. If you close R and reopen, you will need to use the `library()` function again.

A list of all packages is available at <http://cran.r-project.org/web/packages>. Because there are thousands of packages, it may be difficult to find a specific one that fits your needs. One way to find a package is to search the list of packages for keywords.³ For example, search for “group testing” to see a package that I co-authored! Another way to find a package is to use R’s task views at <https://cran.r-project.org/web/views>.

³https://cran.r-project.org/web/packages/available_packages_by_name.html

```
> qt(p = c(0.025, 0.975), df = 9)
[1] -2.262 2.262
```

The `qt()` function finds the 0.025 and 0.975 quantiles from a t-distribution with 9 degrees of freedom. The leading `[1]` in the output indicates the first number of a resulting vector. If the vector was long enough to extend over to a new line, a new leading `[]` would indicate the position of the first number in the second line displaying the vector.

As another example, suppose I want to find a 95% confidence interval for a population mean:

```
> x <- c(3.68, -3.63, 0.8, 3.03, -9.86, -8.66, -2.38, 8.94, 0.52,
1.25)
> x
[1] 3.68 -3.63 0.80 3.03 -9.86 -8.66 -2.38 8.94 0.52 1.25
> mean(x) + qt(p = c(0.025, 0.975), df = length(x) - 1) * sd(x)/sqrt(length(x))
[1] -4.707 3.445
> t.test(x = x, mu = 2, conf.level = 0.95)
```

One Sample t-test

```
data: x
t = -1.46, df = 9, p-value = 0.1782
alternative hypothesis: true mean is not equal to 2
95 percent confidence interval:
 -4.707 3.445
sample estimates:
mean of x
 -0.631
```

Notice how the calculations are done automatically even though the `qt()` function produces a vector with two elements in it. I checked my confidence interval calculation with the results from `t.test()`, which automatically calculates the confidence interval and does a hypothesis test for a specified mean (`mu`). Please be careful when intermixing vectors and scalar values when doing

These task views provide lists of packages by application type. For example, if you want a package that has functions available to optimize a general mathematical function, check out the optimization task view.

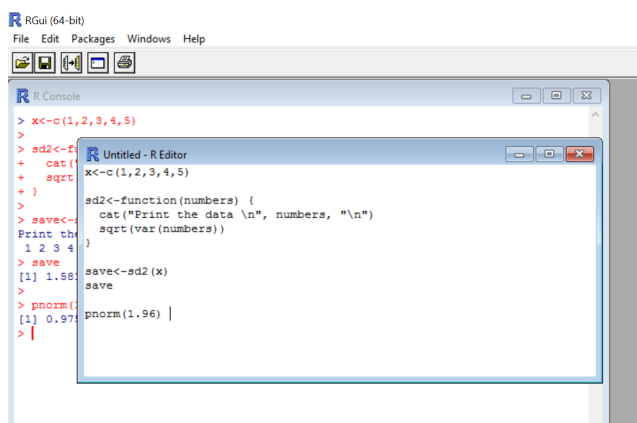
Characters

Object names can include periods and underscores. For example, `mod.fit` could be a name of an object and it is often said as “mod dot fit”. Note that R IS CASE SENSITIVE!

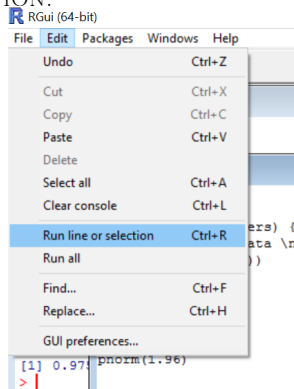
R’s program editor

Often, you will have a long list of commands that you would like to execute all at once—i.e., a program. Instead of typing all of the code line by line at the R Console prompt, you could type it in Notepad or some other text editor and copy and paste the code into R.

Starting with R 2.0, a VERY limited program editor was incorporated into R. Select `FILE > NEW SCRIPT` to create a new program. Below is what the editor looks like with some of the past examples.



To run the current line of code (where the cursor is positioned) or some highlighted code, select EDIT > RUN LINE OR SELECTION.



To run all of the program, select EDIT > RUN ALL. To save your

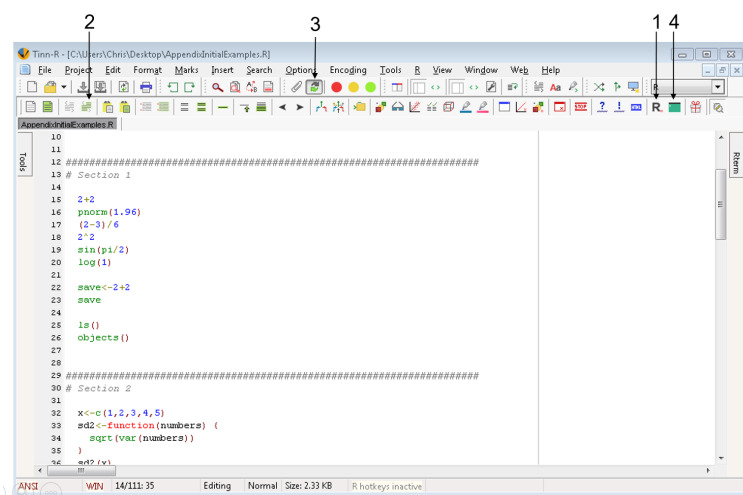
code as a program outside of R, select FILE > SAVE and make sure to use a .R extension on the file name. To open a program, select FILE > OPEN SCRIPT. Note that you can have more than one program open at the same time.


There are MUCH BETTER program editors! Each of the editors described next have syntax highlighting (different colors for different types of code) of the program code which makes reading programs MUCH easier! I recommend using one of these editors.

Tinn-R

Tinn-R (<http://nbcgib.uesc.br/lec/software/editores/tinn-r-en>) is a free, Windows-based program editor that is a separate software package outside of R. This editor is much more advanced than the R editor. Note that a program needs to be saved with the .R extension for syntax highlighting to appear by default.

Below is a screen capture of what version 4.0.3.5 looks like.



In order to run code from the editor, R's GUI needs to be open. This can be opened by selecting the "R control: gui (start/close)" icon from the R toolbar (see #1 for the icon appearance after R is open; before R is open, the icon is ).

Tinn-R subsequently opens R in its single-document interface (SDI), which is a little different from R's multiple-document interface (MDI) that we have been using so far. The difference between the two interfaces is simply that the MDI uses the R GUI to contain all windows that R opens (like a graphics window as shown later in the notes) and the SDI does not.

Once R is open in its SDI, program code in Tinn-R can be transferred to R by selecting specific icons on Tinn-R's R toolbar. For example, a highlighted portion of code can be transferred to and then run in R by selecting the "R send: selection (echo = TRUE)" icon (see #2). Note that the transfer of code from Tinn-

R to R does not work in the MDI.

Below are some additional important comments and tips for using Tinn-R:

- Upon Tinn-R's first use with R's SDI, the TinnRcom package is automatically installed within R to allow for the communication between the two softwares. This package is subsequently always loaded for later uses.
- When R code is sent from Tinn-R to R, the default behavior is for Tinn-R to return as the window of focus (i.e., the window location of the cursor) after R completes running the code. If Tinn-R and R are sharing the same location on a monitor, this prevents the user from immediately seeing the results in R due to it being hidden behind the Tinn-R window. In order to change this behavior, select OPTIONS > APPLICATION > R > RGUI and uncheck the RETURN to Tinn-R box. Alternatively, select the "Options: return focus after send/control Rgui" icon on the Misc toolbar (see #3).
- By default, the line containing the cursor is highlighted in yellow. To turn this option off, select OPTIONS > HIGHLIGHTERS (SETTINGS) and uncheck the ACTIVE LINE (CHOICE) box. The highlighters settings window also allow you to customize the color syntax for highlighting of code. For example, I always use a gray background for my comments so that they stand out better.

```
#####
# Section 2

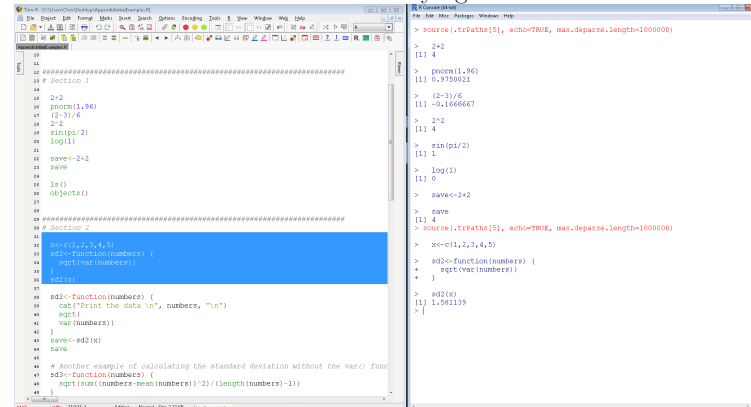
x<-c(1,2,3,4,5)
sd2<-function(numbers) {
  sqrt(var(numbers))
}
sd2(x)

sd2<-function(numbers) {
  cat("Print the data \n", numbers, "\n")
  sqrt(
    var(numbers))
}
save<-sd2(x)
save
```

- Long lines of code are wrapped to a new line by default. This behavior can be changed by selecting OPTIONS > APPLICATION > EDITOR and then selecting the NO RADIO button for LINE WRAPPING.
- Syntax highlighting can be maintained with code that is copied and pasted into Word. After highlighting the desired code to copy, select EDIT > COPY FORMATED (TO EXPORT) > RTF. The subsequently pasted code will retain its color.
- When more than a few lines of code are transferred to R, you will notice that much of the code is not displayed in R to save space. This behavior can be changed by selecting OPTIONS > APPLICATION > R > BASIC and then changing the “option (max.deparse.length (echo=TRUE))” value to a very large number. I use a value of 100000000. Note that ALL R code and output always needs to be shown for anything turned in for a grade!
- Tinn-R can run R within its interface by using a link to a terminal version of R rather than R’s GUI. To direct code to

the Rterm window (located on the right side of Tinn-R), select the “R control: term (start/close)” icon on the R toolbar (see #4). One benefit from using R in this manner is that the syntax highlighting in the program editor is maintained in the R terminal window.

When using Tinn-R and R’s GUI, it can be more efficient to use them in a multiple monitor environment. This allows for both to be viewable in different monitors at the same time. Code and output can be side-by-side in large windows without needing to switch back-and-forth between overlaying windows.

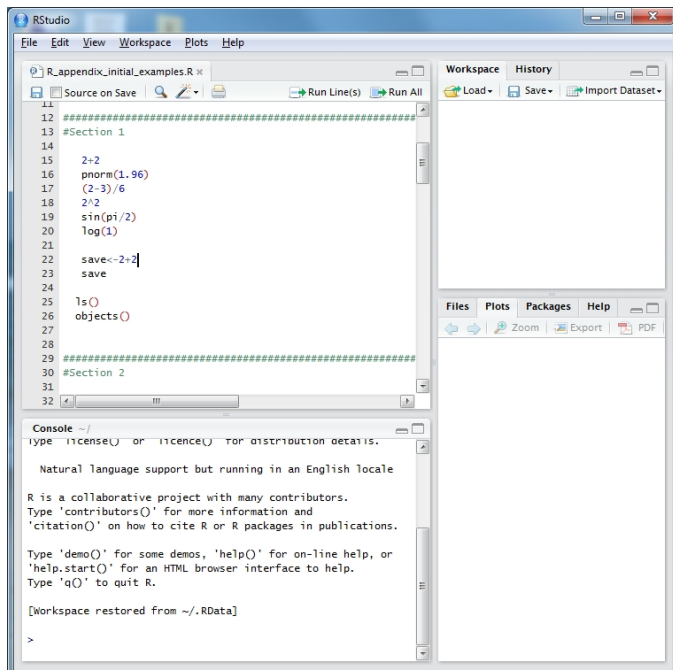


The same type of environment is achievable with a large, wide-screen monitor as well.

RStudio

While still fairly new in comparison to other editors, RStudio’s “RStudio Desktop” (<http://www.rstudio.com>; hereafter just referred to as “RStudio”) is likely the most used among all editors.

This software is actually more than an editor because it integrates a program editor with the R Console window, graphics window, R-help window, and other items within one overall window environment. Thus, RStudio is an integrated development environment (IDE) for constructing and running R programs. The software is available for free, and it runs on Linux, Mac, and Windows operating systems. Below is a screen capture of Rstudio version 0.92.23.



You can start writing a new program by selecting FILE > NEW

> R SCRIPT or open an existing program by selecting FILE > OPEN FILE (without a program open, you will not see the program editor). To run a segment of code, you can highlight it and then select the “Run” icon in the program editor window.

Also, the editor can suggest function or package names from any loaded package if <Tab> is pressed at the end of any text string. For example, typing “pn” and pressing <Tab> yields a pop-up window suggesting functions `pnbinom()`, `png()`, and `pnorm()`. Pressing <Tab> where an argument could be given within a function (e.g., after a function name and its opening parenthesis or after a comma following another argument) gives a list of possible arguments for that function.

The windows available on the right side of the screen provide some additional useful information. In the upper right corner, you can view the list of objects in R’s database (similar to using `ls()` or `objects()` in the R Console). In the bottom right corner, all graphs will be sent to the PLOTS tab and help is immediately available through the HELP tab. Also, in the bottom right corner window, packages can be installed via the PACKAGES tab.

Other editors

I have often used WinEdt with the RWinEdt add-on in the past on my Windows-based computers. Also, the Emacs editor (<http://www.gnu.org/software/emacs>) with the Emacs Speaks Statistics (<http://ess.r-project.org>) add-on are popular for Linux users.

With Microsoft’s purchase of Revolution Analytics, they have been integrating R into some of their products. In particular, Microsoft has developed R Tools for Visual Studio (RTVS) so that one can use Visual Studio to develop R programs in similar ways that RStudio does already. The Community version of Visual Studio and RTVS are available for free at <https://www>.

visualstudio.com/en-us/features/rtvs-vs.aspx. A video on its use is available at <https://www.youtube.com/watch?v=KPS0ytrt9SA>.