Contents

Part I

Part: Preliminaries

1 R Preliminaries

1.1 R and R Studio

You covered this part in the "pre-course".

1.1.1 What is R?

- R is a statistical programming language
- R is a free/libre and open-source software, freely and openly distributed under the GNU General Public License (GPL).
- The base version of R that you download when you install R includes the core code, but you can also install adds-on ("packages") that people all over the world have developed to extend R.

R is becoming more and more useful for a variety of programming tasks. However, where it really shines is in working with data and doing statistical analysis.

Other programming languages popular for statistical computing include:

- SAS
- SPSS
- Matlab
- Python

R has some of the same strengths (quick and easy to code, interfaces well with other languages, easy to work interactively) and weaknesses (slower than compiled languages) as Python.

For data-related tasks, R and Python are fairly neck-and-neck.

However, R is still the first choice of statisticians in most fields, so it may be argued that R has a an advantage if you want to have access to cutting-edge statistical methods.

> "The best thing about R is that it was developed by statisticians. The worst thing about R is that… it was developed by statisticians." -Bo Cowgill, Google, at the Bay Area R Users Group

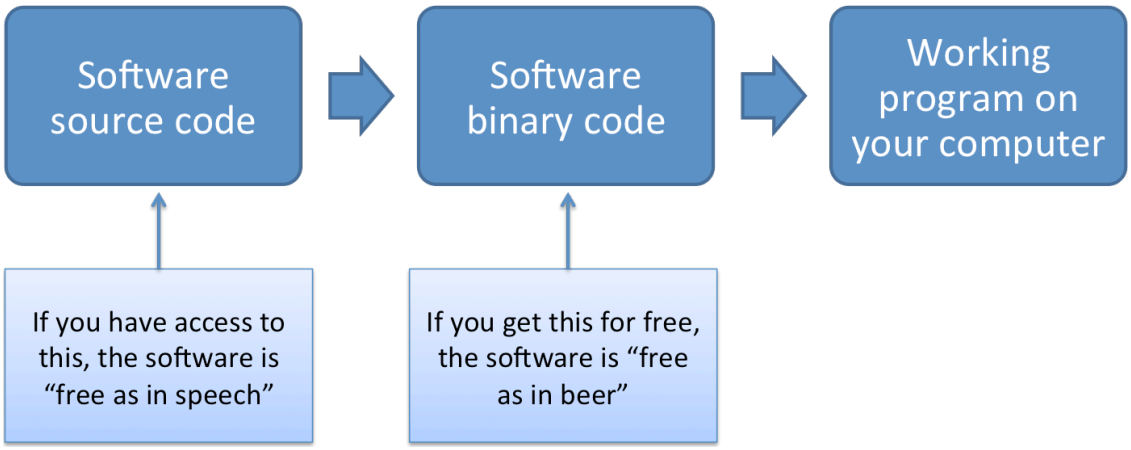

Figure 1: An overview of how software can be each type of free. For software programs developed using a compiled programming language, the final product on your computer is run by machine-readable binary code. A developer can give you this code for free without sharing any of the original source code. By contrast, open-source software is software for which you have access to the human-readable code that was used as in input in creating the software binaries.

1.1.2 Open-source software

> "Life is too short to run proprietary software." – Bdale Garbee

R is open-source software. Many other popular statistical programming languages, conversely, are proprietary.

R is free, and it's tempting to think of open-source software just as "free software".

It helps to consider some different meanings of the word "free". "Free" can mean:

- *Gratis*: Free as in beer
- *Libre*: Free as in speech

Open-source software software is the *libre* type of free (Figure 1). This means that, with software that is open-source, you can:

- Access all of the code that makes up the software
- Change the code as you'd like for your own applications
- Build on the code with your own extensions
- Share the software and its code, as well as your extensions, with others

---

It also means that you can build your own software on top of existing R software and its extensions.

This open-source nature of R (and other languages, including Python) has created a large community of people worldwide who develop and share extensions to R.

As a result, you can pull in packages that let you do all kinds of things in R, like visualizing Tweets, cleaning up accelerometer data, analyzing complex surveys, fitting maching learning models, and a wealth of other cool things.

> "Despite its name, open-source software is less vulnerable to hacking than the secret, black box systems like those being used in polling places now. That's because anyone can see how open-source systems operate. Bugs can be spotted and remedied, deterring those who would attempt attacks. This makes them much more secure than closed-source models like Microsoft's, which only Microsoft employees can get into to fix." – Woolsey and Fox. *To Protect Voting, Use Open-Source Software.* New York Times. August 3, 2017.

You can download the latest version of R from CRAN.

Be sure to select the distribution for your type of computer system. Check your current R version (one way is by running `sessionInfo()`) to make sure you're not using an outdated version of R.

1.1.3 What is RStudio?

RStudio is an integrated development environment (IDE) for R.

This basically means that it provides you an interface for running R and coding in R, with a lot of nice extras that will make your life easier.

Once you have installed R, download RStudio. You want the Desktop version with the free license.

1.2 The "package" system

1.2.1 R packages

Your original download of R is only a starting point (Figure 2).

You can expand functionality of R with **packages** (Figure 3).

Each package is basically a bundle of extra R functions.

You can get these "add-on" packages in a number of ways.

- CRAN (the Comprehensive R Archive Network), the main source
- Bioconductor, specifically for bioinformatics-related packages)
- GitHub is growing in popularity, especially for packages that are still in development.
- You can also create and share packages among your collaborators or co-workers.

1.2.2 Installing from CRAN

The most popular place from which to get packages is currently CRAN, which has over 10,000 R packages available.

The `ggplot2` package for creating beutiful graphs. Since this package is on CRAN, you can install it to your computer using the `install.packages` function:

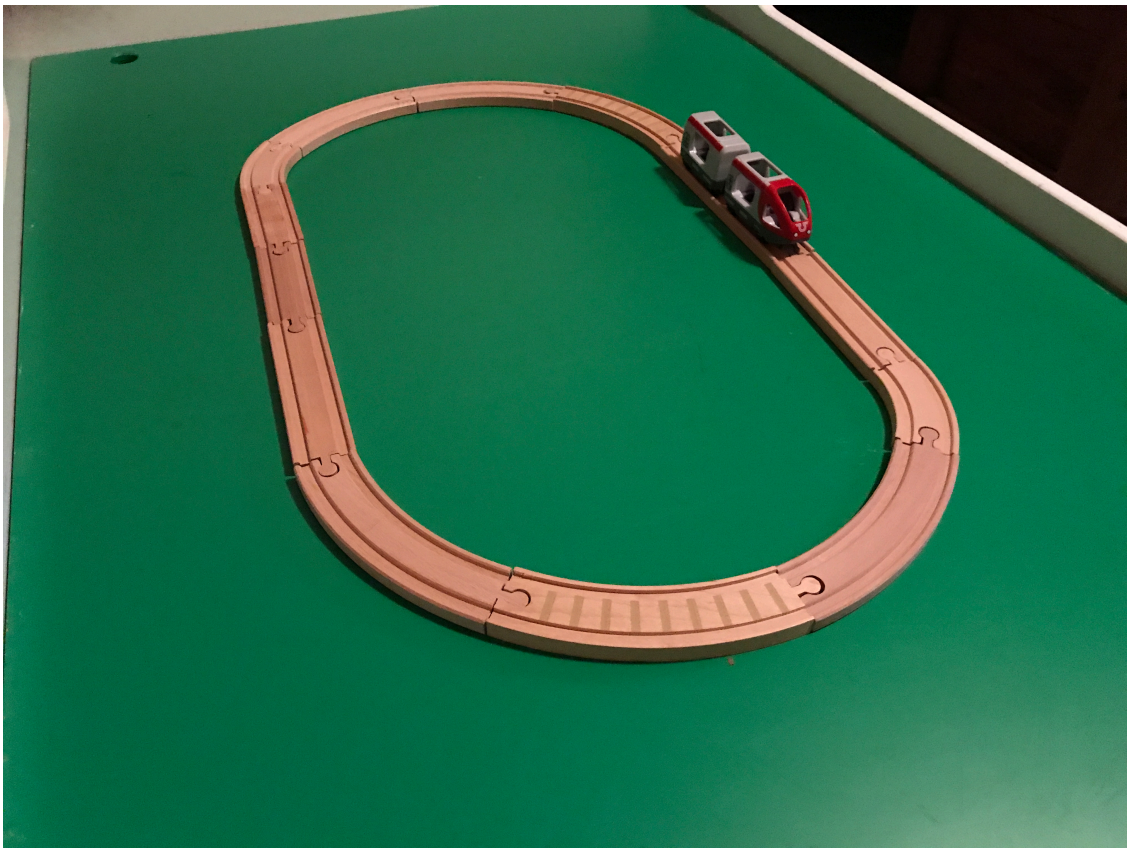

Figure 2: The toy version of base R.

```
install.packages("ggplot2")
```

This downloads the package from CRAN and saves it in a special location on your computer where R can load it when you're ready to use it.

1.2.3 Loading an installed package

Once you have installed a package, it will be saved to your computer.

However, you won't be able to access its functions within an R session until you load it in that R session.

Loading a package essentially makes all of the package's functions available to you.

You can load a package in an R session using the `library` function, with the package name inside the parentheses.

```
library(ggplot2)
```

Figure 3: The toy version of what your R set-up will look like once you add on packages to use for your research.

1.2.4 Package vignettes

Many packages will come with a "vignette", or a tutorial on how to use the package.

To get a list of all the vignettes a package has, use the `vignette` function, specifying a package name with `package` option:

```
vignette(package = "ggplot2")
```

To open a vignette, you can also use the `vignette` function. For example, to open the vignette called "extending-ggplot2" for the `ggplot2` package, run:

```
vignette("extending-ggplot2", package = "ggplot2")
```

1.3 Basic code conventions of R

1.3.1 R's MVP: The *gets arrow*

The *gets arrow*, `<-`, is R's assignment operator. It takes whatever you've created on the right hand side of the `<-` and saves it as an object with the name you put on the left hand side.

```
## Note: Generic code
[name of object] <- [thing I want to save]
```

In R, objects are the way to save something to use again later. If you do not assign something to an object, R will just print it back out to you. For example, just type `"Hello"`:

```
"Hello"
#> [1] "Hello"
```

However, if I assign `"Hello"` to an object, I can print it out or use it later by typing ("referencing") that object name:

```
word <- "Hello"
word
#> [1] "Hello"
```

You can assign the output of a function call directly to an object. For example:

```
n <- nchar(word)
n
#> [1] 5
```

If you would like to see all the R objects that are currently defined in your R session, you can do that with the `ls` command:

```
ls()
#> [1] "n"         "proj_path" "word"
```

You can also check out the "Environment" pane in RStudio to see some of the R objects defined in your current R session.

1.3.2 Assignment operator wars: <- vs. =

You can make assignments in R using either the gets arrow (`<-`) or `=`.

R gurus advise using `<-` rather than `=` when coding in R, and as you move to doing more complex things, some subtle problems might crop up if you use `=`.

RStudio's keyboard shortcut for the gets arrow Alt + - on Windows and Option + - on Macs.

To see a full list of RStudio keyboard shortcuts, go to the "Help" tab in RStudio and select "Keyboard Shortcuts".

1.3.3 Naming objects

When you assign objects, you will need to choose names for them.

There are only two fixed rules for naming objects in R:

- Use only letters, numbers, and underscores
- Don't start with anything but a letter

In addition to these fixed rules, there are also some guidelines for naming objects. From Hadley Wickham's R style guide:

- Use lower case for variable names
- Use an underscore as a separator
- Avoid using names that are already defined in R

1.4 R's most basic object types

An R *object* stores some type of data that you want to use later in your R code.

The content of R objects can vary from very simple (the `"Hello"` string in the example above) to very complex objects (for example, a machine learning model).

There are a variety of different object types in R, shaped to fit different types of objects.

The most basic types of objects are **vectors** (1D) and **dataframes** (2D).

1.4.1 Vectors

- A *vector* is a string of values.
- All values in a vector must be of the same class (i.e., all numbers, all characters, all dates).
- You can use the concatenation function, `c` to join values together, with commas between the values.

For example,

```
fibonacci <- c(1, 1, 2, 3, 5)
fibonacci
#> [1] 1 1 2 3 5
```

```
one_to_five <- c("one", "two", "three", "four", "five")
one_to_five
#> [1] "one"   "two"   "three" "four"  "five"
```

If you mix classes, it will default to most generic:

```
mixed_classes <- c(1, 3, "five")
mixed_classes
#> [1] "1"    "3"    "five"
```

You can use the `class` function to figure out the class of a vector.

```
class(fibonacci)
#> [1] "numeric"
class(one_to_five)
#> [1] "character"
class(mixed_classes)
#> [1] "character"
```

A vector's *length* is the number of elements in the vector. You can use the `length` function to determine a vector's length:

```
length(mixed_classes)
#> [1] 3
```

You can pull out certain values from a vector by using indexing with square brackets (`[...]`):

```
fibonacci[2] # Get the second value
#> [1] 1
fibonacci[c(1, 5)] # Get first and fifth values
#> [1] 1 5
fibonacci[1:3] # Get the first three values
#> [1] 1 1 2
```

You can also use logic to pull out some values of a vector. For example, you might only want to pull out even values from the `fibonacci` vector (make for exercise!).

```
# What is the code to pull out even values from the `fibonacci` vector?
#> [1] 2
# And as for odd values?
#> [1] 1 1 3 5
```

1.4.2 Dataframes

A dataframe is a 2D, and is made of one or more vectors of the same length (although they don't have to be the same class). It is the closest R has to an Excel spreadsheet-type structure.

You can create dataframes using the `data.frame` function. However, most often you will create a dataframe by reading in data from a file, using something like `read.csv`.

To create a dataframe using `data.frame`, in this case by sticking together vectors you already have saved as R objects, you can run:

```
fibonacci_seq <- data.frame(num_in_seq = one_to_five,
                            fibonacci_num = fibonacci)
fibonacci_seq
#>   num_in_seq fibonacci_num
#> 1        one             1
#> 2        two             1
#> 3      three             2
#> 4       four             3
#> 5       five             5
```

The general format for using `data.frame` is:

```
## Note: Generic code
[name of object] <- data.frame([1st column name] = [1st column content],
                               [2nd column name] = [2nd column content])
```

You can use square-bracket indexing (`[..., ...]`) for dataframes, too, but now they'll have two dimensions (rows, then columns). Put the rows you want before the comma, the columns after. If you want all of something, leave the designated spot blank. For example:

```
fibonacci_seq[1:2, 2] # First two rows, second column
#> [1] 1 1
fibonacci_seq[5, ] # Last row, all columns
#>   num_in_seq fibonacci_num
#> 5       five             5
```

So far, we've only shown how to create dataframes from scratch within an R session. Usually, however, you'll create R dataframes by reading in data from an outside file. For example, if you have data on all the guests that came on the *Daily Show* with Jon Stewart (from 538 blog) in a comma-separated (csv) file called "daily_show_guests.csv", you can read it with the following code:

```
daily_show <- ...
```

```
daily_show[1:2, ]
#>   YEAR GoogleKnowlege_Occupation    Show  Group  Raw_Guest_List
#> 1 1999                    actor 1/11/99 Acting  Michael J. Fox
#> 2 1999                 Comedian 1/12/99 Comedy Sandra Bernhard
```

You can use the functions `dim`, `nrow`, and `ncol` to figure out the dimensions (number of rows and columns) of a dataframe:

```
dim(daily_show)
#> [1] 2693    5
```

```
nrow(daily_show)
#> [1] 2693
ncol(daily_show)
#> [1] 5
```

Base R also has some useful functions for quickly exploring dataframes:

- `str`: show the structure of an R object,
- `summary`: give summaries of each column of a dataframe.

For example,

```
summary(daily_show[ , c(1, 2)])
#>       YEAR      GoogleKnowlege_Occupation
#>  Min.   :1999   actor     : 596
#>  1st Qu.:2003   actress   : 271
#>  Median :2007   journalist: 180
#>  Mean   :2007   author    : 102
#>  3rd Qu.:2011   Journalist:  72
#>  Max.   :2015   (Other)   :1446
#>                 NA's      :  26
```