STAT 107 Outline of Class Notes

Rebecca Kurtz-Garcia

2021-09-12

Contents

Welcome

Welcome to STAT 107! Here is stuff that can be written.

Chapter 1

Introduction to R

This section was written primarily by Dr. Robert Desharnais [2020], and was modified for our course. I am grateful for his help.

1.1 The RStudio Interface

We will begin by looking at the RStudio software interface. Refer to Figure 1 as you follow the instructions below.

Launch RStudio. You will see a window that looks like Fig. 1. There are four panels of the window:

- The R Command Console is where you type R commands for immediate execution.
- The Notebook in the upper left portion of the window is an area for editing R source code for scripts and functions and for viewing R data frame objects. New tabs will be added as new R code files and data objects are opened.
- The Notebook in the upper right portion of the window is an area for browsing the variables in the R workspace environment and the R command line history.
- The Notebook in the lower right portion of the window has several tabs. The Files tab is an area for browsing the files in the current working directory. The Plot tab is for viewing graphics produced using R commands. The Packages tab lists the R packages available. Other packages can be loaded. The Help tab provides access to the R documentation. The Viewer tab is for viewing local web content in the temporary session directory (not files on the web).

Bottom Left Pane

Let's begin with the Console. This is where you type R commands for immediate execution. Click in the Command Console, ">" symbol is the system prompt. You should see a blinking cursor that tells you the console is the current focus of keyboard input. Type:

```
1+2
```

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

The result tells you that the line begins with the first (and only) element of the result which is the number 3. You can also execute R's built-in functions (or functions you add). Type the following command.

```
exp(pi)
```

```
## [1] 23.14069
```

In R, "pi" is a special constant to represent the number and "exp" is the exponential function. The result tells you that the first (and only) element of the result is the number $e^{\pi} = 23.14069$.

Bottom Right Pane

Now let's look at the *Files* tab of the notebook at the lower right of the window. Every R session has a working directory where R looks for and saves files. It is a good practice to create a different directory for every project and make that directory the working directory. For example, let's make a new directory called *MyDirectory*. (You can chose another name if you wish).

1) Click on the **Files** tab of the notebook. You should see a listing of files in your default working directory.

2) Click on the small button with an ellipsis image on the right side of the file path above the directory listing.

3) Navigate to the folder where you want to create the new directory and click the **OK** button.

4) Click on the **New Folder** button just below the Files tab (see right).

5) Type **MyDirectory** in the panel that opens click on the folder in the Notebook.

6) Click the **More** button to the right of the New Folder button and select the menu option **Set as Working Directory**. This new folder is now the working directory for the current R session. This menu option is a short cut for a command that was automatically entered into the R console.

Top Right Pane

Next we will look at the *R environment*, also called the *R workspace*. This is where you can see the names and other information on the variables that were created during your R session and are available for use in other commands.

In the R console type:

```
a = 29.325
b = log(a)
c = a/b
```

Look at the Environment pane. The variables a, b, and c are now part of your R work space. You can reuse those variables as part of other commands.

In the R console type:

```
v= c(a, b, c)
v
```

```
## [1] 29.325000  3.378440  8.680041
```

The variable v is a vector created using the *concatenate* function c(). (The concatenate should not be confused with the variable c that was created earlier. Functions are always followed by parentheses that contain the function arguments.) This function combines its arguments into a vector or list. Look at the Environment panel. The text num [1:3] tells us that the variable v is a vector with elements v[1], v[2], and v[3].

Top Left Pane

Now let's look at the R viewer notebook. This panel can be used to data which are data frame objects or *matrix objects* in R.

We will begin by taking advantage of a data frame object that was built into R for demonstration purposes. We will copy it into a data frame object. In the R console, type:

```
df = mtcars
```

Let's view the data. On the right side of the entry for the df object is a button we can use to view the entries of the data frame (see green arrow below). Click on the View Button.

If your look in the notebook area in the upper left portion of the window, you can see a spreadsheet-like view of the data. This is for viewing only; you cannot edit the data. Use the scroll bars to view the data entries.

You can also list the data in the console by typing the name of the data fame object:

```
df
```

```
##                   mpg cyl  disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4        21.0   6 160.0 110 3.90 2.620 16.46  0  1   4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1   4
## Datsun 710       22.8   4 108.0  93 3.85 2.320 18.61  1  1   4
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0   3
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0   3
## Valiant          18.1   6 225.0 105 2.76 3.460 20.22  1  0   3
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0   3
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0   4
## Merc 230         22.8   4 140.8  95 3.92 3.150 22.90  1  0   4
## Merc 280         19.2   6 167.6 123 3.92 3.440 18.30  1  0   4
## Merc 280C        17.8   6 167.6 123 3.92 3.440 18.90  1  0   4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0   3
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0   3
## Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0   3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0
## Chrysler Imperial  14.7   8 440.0 230 3.23 5.345 17.42  0  0   3
## Fiat 128         32.4   4  78.7  66 4.08 2.200 19.47  1  1   4
## Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1   4
## Toyota Corolla     33.9   4  71.1  65 4.22 1.835 19.90  1  1   4
## Toyota Corona     21.5   4 120.1  97 3.70 2.465 20.01  1  0   3
## Dodge Challenger   15.5   8 318.0 150 2.76 3.520 16.87  0  0   3
## AMC Javelin       15.2   8 304.0 150 3.15 3.435 17.30  0  0   3
## Camaro Z28        13.3   8 350.0 245 3.73 3.840 15.41  0  0   3
## Pontiac Firebird   19.2   8 400.0 175 3.08 3.845 17.05  0  0   3
## Fiat X1-9        27.3   4  79.0  66 4.08 1.935 18.90  1  1   4
## Porsche 914-2     26.0   4 120.3  91 4.43 2.140 16.70  0  1   5
## Lotus Europa      30.4   4  95.1 113 3.77 1.513 16.90  1  1   5
## Ford Pantera L    15.8   8 351.0 264 4.22 3.170 14.50  0  1   5
## Ferrari Dino      19.7   6 145.0 175 3.62 2.770 15.50  0  1   5
## Maserati Bora     15.0   8 301.0 335 3.54 3.570 14.60  0  1   5
## Volvo 142E        21.4   4 121.0 109 4.11 2.780 18.60  1  1   4
```

The columns are labeled with the names of the variables and the rows are labeled with the names of each car. Each row represents the data values for one car; that is, each row is one observation.

1.2 R Objects

At its core, `R` is an objected-oriented computational and programming environment. Everything in `R` is an object belonging to a certain *class*. `R`can represent different types of data. The types include `numeric`, `integer`,

`complex`, `logical`, and `character`. We will look at some examples.

Numbers

There are different types of numeric objects. Specifically, we will first consider real numbers (can have decimal values) and integers. We can examine how `R` stores these types of numbers using the `class()` function.

We will begin with decimal numbers.

```
a = 17.45
a
```

```
## [1] 17.45
```

```
class(a)
```

```
## [1] "numeric"
```

```
b = 5
b
```

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

```
class(b)
```

```
## [1] "numeric"
```

Both the variables `a` and `b` are `numeric` objects. When you type a number `R` will default to treating it as a `numeric` object which allows decimals. You can use the `as.integer()` function to create a variable that is specifically an integer number.

```
a = as.integer(a)
a
```

```
## [1] 17
```

```
class(a)
```

```
## [1] "integer"
```

```
b = as.integer(b)
b
```

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

```
class(b)
```

```
## [1] "integer"
```

Logical

Logical values are either “true” or “false” and are created by logical statements that compare variables. There are several ways to do logical statements. To check if two values or objects are equal to each other we use “==”. To see if a value/object is greater than or less than another we use the “<”, “>”, “<=”, or “>=”.

```
b = 5
2+3 == b
```

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

```
n = (10<11)
n
```

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

```
class(n)
```

```
## [1] "logical"
```

```
e = (b >10)
class(e)
```

```
## [1] "logical"
```

The symbols “&” “|”, and “!” are used for the logical operations “and”, “or”, and “not”. Two logical expressions connected by & (and) are true only if both are true. Two logical expressions connected by | (or) are true if either are true. The ! (not) operation turns a true into a false and vice-versa.

```
e & n
```

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

```
e | n
```

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

```
!n
```

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

Character

`Character` values are text. They are often used as data values and labels.

```
first = "George"
first
```

```
## [1] "George"
```

```
class(first)
```

```
## [1] "character"
```

```
last = "Washington"
last
```

```
## [1] "Washington"
```

```
class(last)
```

```
## [1] "character"
```

There are several functions that can operate on character strings.

```
full = paste(first, last)
full
```

```
## [1] "George Washington"
```

```
nchar(full)
```

```
## [1] 17
```

```
tolower(full)
```

```
## [1] "george washington"
```

```
toupper(full)
```

```
## [1] "GEORGE WASHINGTON"
```

The function `paste()` concatenates two or more character strings with a separator, which is a space by default. The function `nchar()` returns the number of characters in a string. The functions `tolower()` and `toupper()` changes any upper case characters to lower case and vice-versa.

Vectors

All the objects we have created this far are single element **vectors**. `R` is a vectorized language, meaning most of the procedures, functions, and operations have been optimized to work with vectors. It is often advantageous to utilize this feature. A vector is a one dimensional array of the same data type. We can use the concatenate function, `c()`, to create vectors, and to make a vector larger.

```
v1 = c(19, 390.3, pi, -32.1)
v1
```

```
##  [1]   19.000000  390.300000    3.141593  -
32.100000
```

```
class(v1)
```

```
## [1] "numeric"
```

```
v2 = c(1.1, 6, -9.4, 32.1)
v2
```

```
## [1]  1.1  6.0 -9.4 32.1
```

```
class(v2)
```

```
## [1] "numeric"
```

```
v3 = c(v1, first)
class(v3)
```

```
## [1] "character"
```

```
v4 = c(first, last)
class(v4)
```

```
## [1] "character"
```

The **length()** function can be used to obtain the number of elements in a vector.

```
length(v1)
```

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

Vectors can be used in arithmetic computations. If the two vectors are of the same length, the computations are performed element-by-element.

```
v1 + v2
```

```
## [1]  20.100000 396.300000  -6.258407   0.000000
```

```
v1 * v2
```

```
## [1]     20.90000   2341.80000    -29.53097 -
1030.41000
```

Single numbers (scalars) will operate on all the vector elements in an expression.

```
5*v1
```

```
##  [1]     95.00000  1951.50000     15.70796  -
160.50000
```

```
v1/3
```

```
##  [1]     6.333333  130.100000     1.047198  -
10.700000
```

Individual elements of a vector can be obtained using an index in square brackets. A negative index removes that element from the vector. The `v2[-1]` is the vector `v2` with the first element removed. The concatenate function can be used to obtain two or more elements of a vector in any desired order. Here `v1[c(3,2)]` returns the third and second elements of the vector `v1`.

```
v1[3]
```

```
## [1] 3.141593
```

```
v2[-1]
```

```
## [1]  6.0 -9.4 32.1
```

```
v3[c(3,2)]
```

```
## [1] "3.14159265358979" "390.3"
```

1.2.1 Factors

Matrices

A matrix is a two dimensional array of data of the same type. The matrix function, `matrix()`, can be used to create a new matrix.

```
m = matrix(c(1, 9, 2, 0, 5, 7, 3, 8, 4),
           nrow=3, ncol=3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    3
## [2,]    9    5    8
## [3,]    2    7    4
```

`R` labels the rows and columns for us in the output. The matrix is filled column-by-column using the elements of the vector created by the concatenate function.

As with vectors, matrices can be used in arithmetic expressions with scalars and other matrices of the same size.

```
m2 = m/2
m2
```

```
##      [,1] [,2] [,3]
## [1,]  0.5  0.0  1.5
## [2,]  4.5  2.5  4.0
## [3,]  1.0  3.5  2.0
```

```
m *m2
```

```
##       [,1] [,2] [,3]
## [1,]  0.5  0.0  4.5
## [2,] 40.5 12.5 32.0
## [3,]  2.0 24.5  8.0
```

Indices can be used to obtain the elements of a matrix, but now we must consider both the row and column.

```
m[2,2]
```

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

```
m[c(1,3), c(1,3)]
```

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

```
m[2,]
```

```
## [1] 9 5 8
```

```
m[,3]
```

```
## [1] 3 8 4
```

Some functions are particularly useful when using matrices. For instance, `t()`, `dim()`, and `c()`. The transpose function, `t()`, switches the column and rows of a matrix. The dimension function, `dim()`, returns the dimensions (number of rows, columns) of a matrix. The concatenate function, `c()`, turns a matrix into a vector by concatenating the columns of the matrix.

Data Frames

Like a matrix, a data frame is a rectangular array of values where each column is a vector, However, unlike a matrix, the columns can be different data types.

We can create a set of vectors of the same length and use the `data.frame()` function to make a data frame object.

```
age = c(25,37,23)
gender = c("Male","Male","Female")
married = c(FALSE, TRUE, FALSE)
friends = data.frame(age, gender, married)
friends
```

```
##   age gender married
## 1  25   Male   FALSE
## 2  37   Male    TRUE
## 3  23 Female   FALSE
```

Each column is given the name of the vector used to define it. The data type of the three columns are numeric, character, and logical.

There are a few functions that are particularly useful when working with data frames. The functions `nrow()` and `ncol()` return the number of rows and columns. The function `rownames()` and `colnames()` can be used to change the row and column names.

```
dim(friends)
```

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

```
nrow(friends)
```

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

```
ncol(friends)
```

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

```
rownames(friends) = c("Doug", "Juan", "Mary")
friends
```

```
##       age gender married
## Doug   25   Male   FALSE
## Juan   37   Male    TRUE
## Mary   23 Female   FALSE
```

We can refer to a data frame element using a row and column number or name.

```
friends[2,3]
```

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

```
friends["Juan", "married"]
```

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

We can also subset an individual columns using the '$' symbol and specifying the column name.

```
friends$age
```

```
## [1] 25 37 23
```

Lists

Chapter 2

More On Objects

Factors

In real-world problems, you often encounter data that can be classified in categories. For example, suppose a survey was conducted of a group of seven individuals, who were asked to identify their hair color and gender.

Name

Hair Color

Gender

Amy

Blonde

Female

Bob

Black

Male

Eve

Black

Female

Kim

Red

Female

Max

Blonde

Male

Ray

Brown

Male

Sam

Black

Male

Here, the hair color and gender are the examples of categorical data. To store such categorical data, R has a special data structure called factors. A factor is an ordered collection of items. The different values that the factor can take are called levels. In R, you can create a factor with the `factor()` function.

```
hcolors = c("Blonde", "Black", "Black", "Red", "Blonde", "Brow
f = factor(hcolors)
f
```

```
## [1] Blonde Black  Black  Red    Blonde Brown  Black
## Levels: Black Blonde Brown Red
```

A factor looks like a vector, but it has special properties. Levels are one of them. Notice that when you print the factor, R displays the distinct levels below the factor. R keeps track of all the possible values in a vector, and each value is called a level of the associated factor.The `levels()` function shows all the levels from a factor.

```
gender = c("Female", "Male", "Female", "Female", "Male", "Male
f = factor(gender)
levels(f)
```

```
## [1] "Female" "Male"
```

```
f
```

```
## [1] Female Male   Female Female Male   Male   Male
## Levels: Female Male
```

If your vector contains only a subset of all the possible levels, then R will have an incomplete picture of the possible levels. Consider the following example of a vector consisting of directions:

```
directions = c("North", "West", "North", "East", "North", "Wes
f = factor(directions)
f
```

```
## [1] North West  North East  North West  East
```

```
## Levels: East North West
```

Notice that the levels of your new factor do not contain the value “South”. So, R thinks that North, West, and East are the only possible levels. However, in practice, it makes sense to have all the possible directions as levels of your factor. To add all the possible levels explicitly, you specify the `levels` argument of the function `factor()`.

```
directions = c("North", "West", "North", "East", "North", "West", "Ea
f = factor(directions,
           levels = c("North", "East", "South", "West"))
f
```

```
## [1] North West  North East  North West  East
## Levels: North East South West
```

R lets you assign abbreviated names for the levels. You can do this by specifying the `labels` argument of `factor()`.

```
directions = c("North", "West", "South", "East", "West", "North", "So
f = factor(directions,
           levels = c("North", "East", "South", "West"),
           labels = c("N", "E", "S", "W"))
f
```

```
## [1] N W S E W N S
## Levels: N E S W
```

Sometimes data has some kind of natural order between elements. For example, sports analysts use a three-point scale to determine how well a sports team is competing:

loss < tie < win.

In market research, it’s very common to use a five point scale to measure perceptions:

strongly disagree < disagree < neutral < agree < strongly agree.

Such kind of data that is possible to place in order or scale is known as **Ordinal data**. In R, there is a special data type for ordinal data. This type is called ordered factors. To create an ordered factor, use the `factor()` function with the argument `ordered=TRUE`.

```
record = c("win", "tie", "loss", "tie", "loss", "win", "win")
f = factor(record,
           ordered = TRUE)
f
```

```
## [1] win  tie  loss tie  loss win  win
## Levels: loss < tie < win
```

You can also reverse the order of levels using the `rev()` function.

```
record = c("win", "tie", "loss", "tie", "loss", "win", "win")
f = factor(record,
           ordered = TRUE,
           levels = rev(levels(f)))
f
```

```
## [1] win  tie  loss tie  loss win  win
## Levels: win < tie < loss
```

If you have no observations in one of the levels, you can drop it using the `droplevels()` function.

```
record = c("win", "loss", "loss", "win", "loss", "win")
f = factor(record,
           levels = c("loss", "tie", "win"))

f
```

```
## [1] win  loss loss win  loss win
## Levels: loss tie win
```

```
droplevels(f)
```

```
## [1] win  loss loss win  loss win
## Levels: loss win
```

The `summary()` function will give you a quick overview of the contents of a factor.

```
gender = c("Female", "Male", "Female", "Female", "Male", "Male
f = factor(gender)
summary(f)
```

```
## Female   Male
##      3      4
```

The function `table()` tabulates observations.

```
table(f)
```

```
## f
## Female   Male
##      3      4
```

Lists

A *list* is an array of objects. Unlike vectors and matrices, the objects can belong to different classes. Lists are useful for packaging together a set of related

objects. We can create a list of objects in our workspace by using the `list()` function.

```
lst = list(1, 2, 3)

# A list of characters
lst = list("red", "green", "blue")

# A list of mixed datatypes
lst = list(1, "abc", 1.23, TRUE)
```

The best way to understand the contents of a list is to use the structure function `str()`. It provides a compact display of the internal structure of a list.

```
lst = list(1, "abc", 1.23, TRUE)
str(lst)
```

```
## List of 4
##  $ : num 1
##  $ : chr "abc"
##  $ : num 1.23
##  $ : logi TRUE
```

A list can contain sublists, which in turn can contain sublists themselves, and so on. This is known as *nested list* or *recursive vectors*.

```
lst = list(1, 3, "abc", list("a","b","c"), TRUE)
str(lst)
```

```
## List of 5
##  $ : num 1
##  $ : num 3
##  $ : chr "abc"
##  $ :List of 3
##   ..$ : chr "a"
##   ..$ : chr "b"
##   ..$ : chr "c"
##  $ : logi TRUE
```

There are two ways to extract elements from a list:

- Using `[[]]` gives you the element itself.
- Using `[]` gives you a list with the selected elements

You can use `[]` to extract either a single element or multiple elements from a list. However, the result will always be a list.

```
# extract 2nd element
lst[2]
```

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

```
# extract 5th element
lst[5]
```

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

```
# select 1st, 3rd and 5th element
lst[c(1,3,5)]
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "abc"
##
## [[3]]
## [1] TRUE
```

```
# exclude 1st, 3rd and 5th element
lst[c(-1,-3,-5)]
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [[2]][[1]]
## [1] "a"
##
## [[2]][[2]]
## [1] "b"
##
## [[2]][[3]]
## [1] "c"
```

You can use `[[]]` to extract only a single element from a list. Unlike `[]`, `[[]]` gives you the element itself.

```
# extract 2nd element
lst[[2]]
```

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

```
# extract 5th element
lst[[5]]
```

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

You can't use logical vectors or negative numbers as indices when using `[[]]`. The difference between `[]` and `[[]]` is really important for lists, because `[[]]` returns the element itself while `[]` returns a list with the selected elements. The difference becomes clear when we inspect the structure of the output – one is a character and the other one is a list.

```
lst = list("a","b","c","d","e","f")

class(lst[[1]])
```

```
## [1] "character"
```

```
class(lst[1])
```

```
## [1] "list"
```

Each list element can have a name. You can access individual element by specifying its name in double square brackets `[[]]` or use `$` operator.

```
months = list(JAN=1, FEB=2, MAR=3, APR=4)

# extract element by its name
months[["MAR"]]
```

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

```
# same as above but using the $ operator
months$MAR
```

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

```
# extract multiple elements
months[c("JAN","APR")]
```

```
## $JAN
## [1] 1
##
## $APR
## [1] 4
```

You can access individual items in a nested list by using the combination of `[[]]` or `$` operator and the `[]` operator.

```
lst = list(item1 = 3.14,
           item2 = list(item2a = 5:10,
                        item2b = c("a","b","c")))

# preserve the output as a list
lst[[2]][1]
```

```
## $item2a
```

```
## [1]  5  6  7  8  9 10
```

```
# same as above but simplify the output
lst[[2]][[1]]
```

```
## [1]  5  6  7  8  9 10
```

```
# same as above with names
lst[["item2"]][["item2a"]]
```

```
## [1]  5  6  7  8  9 10
```

```
# same as above with $ operator
lst$item2$item2a
```

```
## [1]  5  6  7  8  9 10
```

```
# extract individual element
lst[[2]][[2]][3]
```

```
## [1] "c"
```

Modifying a list element is pretty straightforward. You use either the `[[]]` or the `$` to access that element, and simply assign a new value.

```
# Modify 3rd list element
lst = list("a","b","c","d","e","f")
lst[[3]] = 1
str(lst)
```

```
## List of 6
##  $ : chr "a"
##  $ : chr "b"
##  $ : num 1
##  $ : chr "d"
##  $ : chr "e"
##  $ : chr "f"
```

You can modify components using `[]` as well, but you have to assign a list of components.

```
# Modify 3rd list element using []
lst = list("a","b","c","d","e","f")
lst[3] = list(1)
str(lst)
```

```
## List of 6
##  $ : chr "a"
##  $ : chr "b"
##  $ : num 1
##  $ : chr "d"
```

```
##  $ : chr "e"
##  $ : chr "f"
```

Using `[]` allows you to modify more than one component at once.

```
# Modify first three list elements
lst = list("a","b","c","d","e","f")
lst[1:3] = list(1,2,3)
str(lst)
```

```
## List of 6
##  $ : num 1
##  $ : num 2
##  $ : num 3
##  $ : chr "d"
##  $ : chr "e"
##  $ : chr "f"
```

You can use same method for modifying elements and adding new one. If the element is already present in the list, it is updated else, a new element is added to the list.

```
# Add elements to a list
lst = list(1, 2, 3)
lst[[4]] = 4
str(lst)
```

```
## List of 4
##  $ : num 1
##  $ : num 2
##  $ : num 3
##  $ : num 4
```

By using `append()` method you can append one or more elements to the list.

```
# Add more than one element to a list
lst = list(1, 2, 3)
lst = append(lst,c("a","b","c"))
str(lst)
```

```
## List of 6
##  $ : num 1
##  $ : num 2
##  $ : num 3
##  $ : chr "a"
##  $ : chr "b"
##  $ : chr "c"
```

To remove a list element, select it by position or by name, and then assign

`NULL` to it.

```
# Remove element from list
lst = list("a","b","c","d","e")
lst[[3]] = NULL
str(lst)
```

```
## List of 4
##  $ : chr "a"
##  $ : chr "b"
##  $ : chr "d"
##  $ : chr "e"
```

Using `[]`, you can delete more than one component at once.

```
# Remove multiple elements at once
lst = list("a","b","c","d","e")
lst[1:4] = NULL
str(lst)
```

```
## List of 1
##  $ : chr "e"
```

By using a logical vector, you can remove list elements based on the condition.

```
# Remove all negative list elements
lst = list(-4,-3,-2,-1,0,1,2,3,4)
lst[lst <= 0] = NULL
str(lst)
```

```
## List of 4
##  $ : num 1
##  $ : num 2
##  $ : num 3
##  $ : num 4
```

The `c()` does a lot more than just creating vectors. It can be used to combine lists into a new list as well.

```
lst1 = list("a","b","c")
lst2 = list(1,2,3)
lst = c(lst1, lst2)
str(lst)
```

```
## List of 6
##  $ : chr "a"
##  $ : chr "b"
##  $ : chr "c"
##  $ : num 1
##  $ : num 2
```

```
##  $ : num 3
```

Basic statistical functions work on vectors but not on lists. For example, you cannot directly compute the mean of list of numbers. In that case, you have to flatten the list into a vector using `unlist()` first and then compute the mean of the result.

```
lst = list(5, 10, 15, 20, 25)
mean(unlist(lst))
```

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

To find the length of a list, use `length()` function.

```
length(lst)
```

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

Chapter 3

Working with Data Sets

Here let me add some text. I'll add more text.

3.1 Loading Data Sets

Built-In Data

Importing Data

Downloading Data

3.2 Basic Data Manipulation

Chapter 4

Footnotes and citations

4.1 Footnotes

Footnotes are put inside the square brackets after a caret `^[]`. Like this one [1].

4.2 Citations

Reference items in your bibliography file(s) using `@key`.

For example, we are using the **bookdown** package [Xie, 2021] (check out the last code chunk in index.Rmd to see how this citation key was added) in this sample book, which was built on top of R Markdown and **knitr** [Xie, 2015] (this citation was added manually in an external file book.bib). Note that the `.bib` files need to be listed in the index.Rmd with the YAML `bibliography` key.

The RStudio Visual Markdown Editor can also make it easier to insert citations: https://rstudio.github.io/visual-markdown-editing/#/citations

[1] This is a footnote.

Chapter 5

Blocks

5.1 Equations

Here is an equation.

$$f\left(k\right) = \binom{n}{k} p^k \left(1-p\right)^{n-k} \tag{5.1}$$

You may refer to using `\@ref(eq:binom)`, like see Equation (5.1).

5.2 Theorems and proofs

Labeled theorems can be referenced in text using `\@ref(thm:tri)`, for example, check out this smart theorem 5.1.

Theorem 5.1. *For a right triangle, if c denotes the* length *of the hypotenuse and a and b denote the lengths of the* ***other*** *two sides, we have*

$$a^2 + b^2 = c^2$$

Read more here https://bookdown.org/yihui/bookdown/markdown-extensions-by-bookdown.html.

5.3 Callout blocks

The R Markdown Cookbook provides more help on how to use custom blocks to design your own callouts: https://bookdown.org/yihui/rmarkdown-cookbook/custom-blocks.html

Chapter 6

Sharing your book

6.1 Publishing

HTML books can be published online, see: https://bookdown.org/yihui/bookdown/publishing.html

6.2 404 pages

By default, users will be directed to a 404 page if they try to access a webpage that cannot be found. If you'd like to customize your 404 page instead of using the default, you may add either a `_404.Rmd` or `_404.md` file to your project root and use code and/or Markdown syntax.

6.3 Metadata for sharing

Bookdown HTML books will provide HTML metadata for social sharing on platforms like Twitter, Facebook, and LinkedIn, using information you provide in the `index.Rmd` YAML. To setup, set the `url` for your book and the path to your `cover-image` file. Your book's `title` and `description` are also used.

This `gitbook` uses the same social sharing data across all chapters in your book- all links shared will look the same.

Specify your book's source repository on GitHub using the `edit` key under the configuration options in the `_output.yml` file, which allows users to suggest an edit by linking to a chapter's source file.

Read more about the features of this output format here:

https://pkgs.rstudio.com/bookdown/reference/gitbook.html

Or use:

```
?bookdown::gitbook
```

Bibliography

Dr. Robert Desharnais. *Biology3000*. California State University, Los Angeles, Los Angeles, CA 90032, 2020.

Yihui Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL http://yihui.org/knitr/. ISBN 978-1498716963.

Yihui Xie. *bookdown: Authoring Books and Technical Documents with R Markdown*, 2021. URL https://CRAN.R-project.org/package=bookdown. R package version 0.23.