

Statistical Computing

Isaac Quintanilla Salinas

Table of contents

Welcome	5
Preface	6
I R Programming	7
1 Basic R Programming	8
1.1 Introduction	8
1.2 Basic Calculations	8
1.2.1 Calculator	8
1.2.2 Comparing Numbers	11
1.2.3 Help	13
1.3 Types of Data	13
1.3.1 Numeric	14
1.3.2 Logical	14
1.3.3 POSIX	15
1.3.4 Character	15
1.3.5 Integers	16
1.3.6 Complex Numbers	16
1.3.7 Raw	16
1.3.8 Missing	17
1.4 R Objects	17
1.4.1 Assigning objects	17
1.4.2 Vectors	18
1.4.3 Matrices	20
1.4.4 Arrays	21
1.4.5 Data Frames	21
1.4.6 Lists	22
2 Control Flow	27
2.1 If/Else Statements	27
2.2 for Loops	27
2.3 while Loops	27

3	Functional Programming	28
3.1	*apply Functions	28
4	Scripting in R	29
II	Data Manipulation, Summarization, and Graphics	30
5	Importing Data	31
6	Data Manipulation	32
7	Data Summarization	33
8	Graphics	34
8.1	Base R Plotting	34
8.2	ggplot2	34
III	Reporting Data	35
9	Markdown Reports	36
10	Notebooks	37
11	Markdown Reports	38
IV	Debugging and Efficient Porgramming	39
12	Debugging Code	40
13	Monte Carlo Integration	41
14	Vectorizing Code	42
15	Incorporating C++ into R	43
V	Permutations	44
16	Permutation Tests	45
17	Permutation Regression	46

VI Monte Carlo Methods	47
18 Monte Carlo Simulations	48
19 Monte Carlo Integration	49
20 Monte Carlo Hypothesis Testing	50
VII Bootstrapping	51
21 Parametric Bootstrapping	52
22 Nonparametric Bootstrapping	53

Welcome

Preface

This is a book created to be used for a statistical computing course at the undergraduate level.

Part I

R Programming

1 Basic R Programming

1.1 Introduction

This chapter focuses on the basics of R programming. While most of your statistical analysis will be done with R functions, it is important to at least have an idea of what is going on. Additionally, we will cover other topics that you may or may not need to know. The topics we will cover are:

1. Basic calculations in R
2. Types of Data
3. R Objects

There are many other topics that should be covered, but it may be unnecessary. If you are interested in those topics, I recommend using the ‘swirl’ package.

1.2 Basic Calculations

This section focuses the basic calculation that you can do in R. Essentially, we look at how R can be used as a calculator. This is done by using different operators in R. An operator is a symbol that tells R to do something. Some common operators are `+`, `-`, and `*` which corresponds to addition, subtraction, and division.

1.2.1 Calculator

1.2.1.1 Addition

To add numbers in R, all you need to use the `+` operator. For example $2+2=4$. When you type it in R you have:

```
2+2
```

```
[1] 4
```


When you ask R to perform a task, it prints out the result of the task. As we can see above, R prints out the number 4.

To add more than 2 numbers, you can simply just type it in.

```
2+2+2
```

```
[1] 6
```

This provides the number 6.

1.2.1.2 Subtraction

To subtract numbers, you need to use the `-` operator. Try `4-2`:

```
4-2
```

```
[1] 2
```

Try `4-6-4`

```
4-6-4
```

```
[1] -6
```

Notice that you get a negative number.

Now try `4+4-2+8`:

```
4+4-2+8
```

```
[1] 14
```

1.2.1.3 Multiplication

To multiply numbers, you will need to use the `*` operator. Try `4*4`:

```
4*4
```

```
[1] 16
```

1.2.1.4 Division

To divide numbers, you can use the `/` operator. Try `9/3`:

```
9/3
```

```
[1] 3
```

1.2.1.5 Exponents

To exponentiate a number to the power of another number, you can use the `^` operator. Try `2^5`:

```
2^5
```

```
[1] 32
```

If you want to take e to the power 2, you will use the `exp()` function. Try `exp(2)`:

```
exp(2)
```

```
[1] 7.389056
```

1.2.1.6 Roots

To take the n-th root of a value, use the `^` operator with the `/` operator to take the n-th root. For example, to take the 5th-root of 32, type `32^(1/5)`:

```
32^(1/5)
```

```
[1] 2
```

1.2.1.7 Logarithms

To take the natural logarithm of a value, you will use the `log()` function. Try `log(5)`:

```
log(5)
```

```
[1] 1.609438
```

If you want to take the logarithm of a different base, you will use the `log()` function with `base` argument. We will discuss this more in section 7 of this chapter.

1.2.2 Comparing Numbers

Another important part of R is comparing numbers. When you compare two numbers, R will tell you if that is true or false. We will talk about some of the basic comparisons and their operators.

1.2.2.1 Less than/Greater than

To check if one number is less than or greater than another number, you will use the `>` or `<` operators. Try `5>4`:

```
5 > 4
```

```
[1] TRUE
```

Notice that R states it's true. It evaluates the expression and tells you if it's true or not. Try `5<4`:

```
5 < 4
```

```
[1] FALSE
```

Notice that R tells you it is false.

1.2.2.2 Less than or equal to/Greater than or equal to

To check if one number is less than or equal to/greater than or equal to another number, you will use the `>=` or `<=` operators. Try `5>=5`:

```
5 >= 5
```

```
[1] TRUE
```

Try `5>=4`:

```
5 >= 4
```

```
[1] TRUE
```

Try `5<=4`

```
5 <= 4
```

```
[1] FALSE
```

1.2.2.3 Equals and Not Equals

To check if 2 numbers are equal to each other, you can use the `==` operator. Try `3==3`:

```
3 == 3
```

```
[1] TRUE
```

Try `4==3`

```
3 == 4
```

```
[1] FALSE
```

Another way to see if 2 numbers are not equal to each other, you can use the `!=`. Try `3!=4`:

```
3 != 4
```

```
[1] TRUE
```

Try `3!=3`:

```
3 != 3
```

```
[1] FALSE
```

You may be asking why use `!=` instead of `==`. They both provides similar results. Well the reason is that you may need the ‘TRUE’ output for analysis. One is only true when they are equal, while the other is true when they are not equal.

1.2.3 Help

The last operator we will discuss is the help operator `?`. If you want to know more about anything we talked about you can type `?` in front of a function and a help page will pop-up in your browser or in RStudio’s ‘Help’ tab. For example you can type `?Arithmetic` or `?Comparison`, to review what we talked about. For other operators we didn’t talk about use `?assignOps` and `?Logic`.

1.3 Types of Data

In R, the type of data, also known as class, that we are using dictates how the programming works. For the most part, users will use ‘numeric’, ‘logical’, ‘POSIX’ and ‘character’ data types. Other types of data you may encounter are ‘integer’, ‘complex’, and ‘raw’. These types of data are rarely used. To obtain more information on them, use the `?` operator.

1.3.1 Numeric

The numeric class is the data that are numbers. Almost every analysis that you use will be based on the numeric class. To check if you have a numeric class, you just need to use the `is.numeric()` function. For example, try `is.numeric(5)`:

```
is.numeric(5)
```

```
[1] TRUE
```

Notice that when you input an number into R, it automatically changes it to a numeric class. R changes data to the class that it most likely needs to be. Now this is great because you do not need to do anything on your end. However, if you need a different class, you will need to change it.

1.3.2 Logical

A logical class are data where the only value is 'TRUE' or 'FALSE'. Sometimes the data is coded as 1 for 'TRUE' and 0 for 'FALSE'. The data may also be coded as 'T' or 'F'. To check if data belongs in the logical class, you will need the `is.logical()` function. Try `is.logical(3<4)`:

```
is.logical(3 < 4)
```

```
[1] TRUE
```

Remember when we ran `3<4` in the previous section. The output was 'TRUE'. Now R is checking whether the output is of a logical class. Since it is, R returns 'TRUE'. Now try `is.logical(3>4)`:

```
is.logical(3 > 4)
```

```
[1] TRUE
```

The output is 'TRUE' as well even though the condition `3>4` is 'FALSE'. Since the output is a logical data type, it is a logical variable.

1.3.3 POSIX

The POSIX class are date-time data. Where the data value is a time component. The POSIX class can be very complex in how it is formatted. IF you would like to learn more try `?POSIXct` or `?POSIXlt`. First, lets run `Sys.time()` to check what is today's data and time:

```
Sys.time()
```

```
[1] "2022-12-23 18:14:42 PST"
```

Now lets check if its of POSIX class, you can use the `class()` function to figure out which class is it. Try `class(Sys.time())`:

```
class(Sys.time())
```

```
[1] "POSIXct" "POSIXt"
```

1.3.4 Character

A character value is where the data values follow a string format. Examples of characters values are letters, words and even numbers. A character value is any value surrounded by quotation marks. For example, the phrase “Hello World!” is considered as one character value. Another example if you data is coded with the actual words “yes” or “no”. To check if you have character data, use the `is.character()` function. Try `is.character("Hello World!")`:

```
is.character("Hello World!")
```

```
[1] TRUE
```

Notice that the output says ‘TRUE’. Character values can be created with single quotations. Try `is.character('Hello World!')`:

```
is.character('Hello World!')
```

```
[1] TRUE
```

1.3.5 Integers

Integers are just whole numbers for the most part. To create an integer, type the letter 'L' after a number. To check if you are using integer data, use the `is.integer()` function. Try `is.integer(5L)`:

```
is.integer(5L)
```

```
[1] TRUE
```

1.3.6 Complex Numbers

Complex numbers are data values where there is a real component and an imaginary component. The imaginary component is a number multiplied by $i = \sqrt{-1}$. To create a complex number, use the `complex()` function. To check if a number is complex, use the `is.complex()` function. Try the following to create a complex number `complex(1,4,5)`:

```
complex(1, 4, 5)
```

```
[1] 4+5i
```

Now try `is.complex(complex(1,4,5))`:

```
is.complex(complex(1, 4, 5))
```

```
[1] TRUE
```

1.3.7 Raw

You will probably never use raw data. I have never used raw data in R. To create a raw value, use the `raw()` or `charToRaw()` functions. Try `charToRaw('Hello World!')`:

```
charToRaw('Hello World!')
```

```
[1] 48 65 6c 6c 6f 20 57 6f 72 6c 64 21
```

To check if you have raw data, use the `is.raw()` function. Try `is.raw(charToRaw('Hello World!'))`:


```
is.raw(charToRaw('Hello World!'))
```

```
[1] TRUE
```

1.3.8 Missing

The last data class in R is missing data denoted as `NA`. Whenever you see `NA` in any of the analysis you see, it means that the data is missing. To check if you have missing data, use the `is.na()` function. Try `is.na(NA)`:

```
is.na(NA)
```

```
[1] TRUE
```

1.4 R Objects

R objects are where most of the statistical analysis is conducted on. An R object can be thought of as a container of data. For the most part, you will only use a data frame (or tibble) for your data analysis. However, it is always a good idea to have some basic understanding of the other R objects.

1.4.1 Assigning objects

To create an R object, all we need to do is assign data to a variable. The variable is the name of the R object. it can be called anything, but you can only use alphanumeric values, underscore, and periods. To assign a value to a variable, use the `<-` operator. This is known as a left assignment. Kinda like an arrow pointing left. Try assigning 9 to 'x' (`x<-9`):

```
x <- 9
```

To see if `x` contains 9, type `x` in the console:

```
x
```

```
[1] 9
```

Now `x` can be treated as data and we can perform data analysis on it. For example, try squaring it:

```
x^2
```

```
[1] 81
```

You can use any mathematical operation from the previous sections. Try some other operations and see what happens.

The output R prints out can be stored in a variable using the assign operator, `<-`. Try storing `x^3` in a variable called `x_cubed`:

```
x_cubed <- x^3
```

To see what is stored in `x_cubed` you can either type `x_cubed` in the console or use the `print()` function with '`x_cubed`' inside the parenthesis.

```
x_cubed
```

```
[1] 729
```

```
print(x_cubed)
```

```
[1] 729
```

1.4.2 Vectors

A vector is a set data values of a certain length. The R object `x` is considered as a numerical vector (because it contains a number) with the length 1. To check, try `is.numeric(x)` and `is.vector(x)`:

```
is.numeric(x)
```

```
[1] TRUE
```

```
is.vector(x)
```

```
[1] TRUE
```

Now let's create a logical vector that contains 4 elements (have it follow this sequence: T,F,T,F) and assign it to `y`. To create a vector use the `c()` function and type all the values and separating it with columns. Type `y<-c(T,F,T,F)`:

```
y <- c(T, F, T, F)
```

Now, lets see how `y` looks like. Type `y`:

```
y
```

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

Now lets see if it's a logical vector:

```
is.logical(y)
```

```
[1] TRUE
```

```
is.vector(y)
```

```
[1] TRUE
```

Fortunately, this vector is really small to count how many elements it has, but what if the vector is really large? To find out how many elements a vector has, use the `length()` function. Try `length(y)`:

```
length(y)
```

```
[1] 4
```

The `c()` function allows you to put any data type and as many values as you wish. The only condition of a vector is that it must be the same data type.

1.4.3 Matrices

A matrix can be thought as a square or rectangular grid of data values. This grid can be constructed in any shape. Similar to vectors they must contain the same data type. The size of a matrix is usually denoted as $n \times k$, where n represents the number of rows and k represents the number of columns. To get a rough idea of how a matrix may look like, type `matrix(rep(1,12),nrow=4,ncol=3)`¹:

```
matrix(rep(1, 12), nrow = 4, ncol = 3)
```

	[,1]	[,2]	[,3]
[1,]	1	1	1
[2,]	1	1	1
[3,]	1	1	1
[4,]	1	1	1

Notice that this is a 4×3 matrix. Each element in the matrix has the value 1. Now try this `matrix(rbinom(12,1.5),nrow=4,ncol=3)`²:

```
matrix(rbinom(12, 1, .5), nrow = 4, ncol = 3)
```

	[,1]	[,2]	[,3]
[1,]	1	1	0
[2,]	0	0	1
[3,]	1	0	1
[4,]	1	1	0

Your matrix may look different, but that is to be expected. Notice that some elements in a matrix are 0's and some are 1's. Each element in a matrix can hold any value.

Constructing a matrix can be a bit difficult to do because the data values may need to be arranged in a certain way. Notice that I used the `matrix()` function to create the matrix. The examples above contain other components in the function that we will discuss later.

¹The function `rep()` creates a vector by repeating a value for a certain length. `rep(1,12)` creates a vector of length 12 with each element being 1

²The `rbinom()` function generates binomial random variables and stores them in a vector. `rbinom(12,1,5)` This creates 12 random binomial numbers with parameter $n = 1$ and $p = 0.5$.

1.4.4 Arrays

Matrices can be considered as a 2-dimensional block of numbers. An array is an n-dimensional block of numbers. While you may never need to use an array for data analysis. It may come in handy when programming by hand. To create an array, use the `array()` function. Below is an example of a $3 \times 3 \times 3$ with the numbers 1, 2, and 3 representing the 3rd dimension stored in an R object called `first_array`³.

```
(first_array <- array(c(rep(1, 9), rep(2, 9), rep(3, 9)),  
                      dim=c(3,3,3)))
```

, , 1

	[,1]	[,2]	[,3]
[1,]	1	1	1
[2,]	1	1	1
[3,]	1	1	1

, , 2

	[,1]	[,2]	[,3]
[1,]	2	2	2
[2,]	2	2	2
[3,]	2	2	2

, , 3

	[,1]	[,2]	[,3]
[1,]	3	3	3
[2,]	3	3	3
[3,]	3	3	3

1.4.5 Data Frames

Data frames seems like a data set that you may encounter in an excel file. However, there are a couple of differences. First, each row represents an observation, and each column represents a characteristic of the observation. Additionally, each column in a data frame will be the same data type. To get an idea of what a data frame looks like, try `head(iris)`:

³Notice the code is surrounded by parenthesis. This tells R to store the array and print out the results. You can surround code with parenthesis everytime you create an object to also print what is stored.

```
head(iris)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

The `head()` function just tells R to only print the top few components of the data frame.

Now try `tail(iris)`:

```
tail(iris)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
145	6.7	3.3	5.7	2.5	virginica
146	6.7	3.0	5.2	2.3	virginica
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica
149	6.2	3.4	5.4	2.3	virginica
150	5.9	3.0	5.1	1.8	virginica

The `tail()` function provides the last 6 rows of the data frame.

1.4.6 Lists

To me a list is just a container that you can store practically anything. It is compiled of elements, where each element contains an R object. For example, the first element of a list may contain a data frame, the second element may contain a vector, and the third element may contain another list. It is just a way to store things.

To create a list, use the `list()` function. Create a list compiled of first element with the `mtcars` data set, second element with a vector of zeros of size 4, and a matrix 3×3 identity matrix⁴. Store the list in an object called `list_one`:

⁴An identity matrix is a matrix where the diagonal elements are 1 and the non-diagonal elements are 0

```
list_one <- list(mtcars, rep(0, 4),
                diag(rep(1, 3)))
```

Type `list_one` to see what pops out:

```
list_one
```

```
[[1]]
```

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

```
[[2]]
[1] 0 0 0 0
```

```
[[3]]
      [,1] [,2] [,3]
[1,]     1     0     0
[2,]     0     1     0
[3,]     0     0     1
```

Each element in the list is labeled as a number. It is more useful to have the elements named. An element is named by typing the name in quotes followed by the = symbol before your object in the `list()` function (`mtcars=mtcars`).

```
list_one <- list(mtcars = mtcars,
                 vector = rep(0, 4),
                 identity = diag(rep(1, 3)))
```

Here I am creating an object called `list_one`, where the first element is `mtcars` labeled `mtcars`, the second element is a vector of zeros labeled `vector` and the last element is the identity matrix labeled `identity`.

Now create a new list called `list_two` and store `list_one` labeled as `list_one` and `first_array` labeled as `array`.

```
(list_two <- list(list_one = list_one,
                  array = first_array))
```

```
$list_one
$list_one$mtcars
```

	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	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4

Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

\$list_one\$vector

[1] 0 0 0 0

\$list_one\$identity

	[,1]	[,2]	[,3]
[1,]	1	0	0
[2,]	0	1	0
[3,]	0	0	1

\$array

, , 1

	[,1]	[,2]	[,3]
[1,]	1	1	1
[2,]	1	1	1
[3,]	1	1	1

, , 2

	[,1]	[,2]	[,3]
[1,]	2	2	2
[2,]	2	2	2
[3,]	2	2	2

, , 3

	[,1]	[,2]	[,3]
[1,]	3	3	3
[2,]	3	3	3
[3,]	3	3	3

2 Control Flow

2.1 If/Else Statements

2.2 for Loops

2.3 while Loops

3 Functional Programming

3.1 *apply Functions

4 Scripting in R

Part II

Data Manipulation, Summarization, and Graphics

5 Importing Data

6 Data Manipulation

7 Data Summarization

8 Graphics

8.1 Base R Plotting

8.2 ggplot2

Part III

Reporting Data

9 Markdown Reports

10 Notebooks

11 Markdown Reports

Part IV

Debugging and Efficient Programming

12 Debugging Code

13 Monte Carlo Integration

14 Vectorizing Code

15 Incorporating C++ into R

Part V

Permutations

16 Permutation Tests

17 Permutation Regression

Part VI

Monte Carlo Methods

18 Monte Carlo Simulations

19 Monte Carlo Integration

20 Monte Carlo Hypothesis Testing

Part VII

Bootstrapping

21 Parametric Bootstrapping

22 Nonparametric Bootstrapping