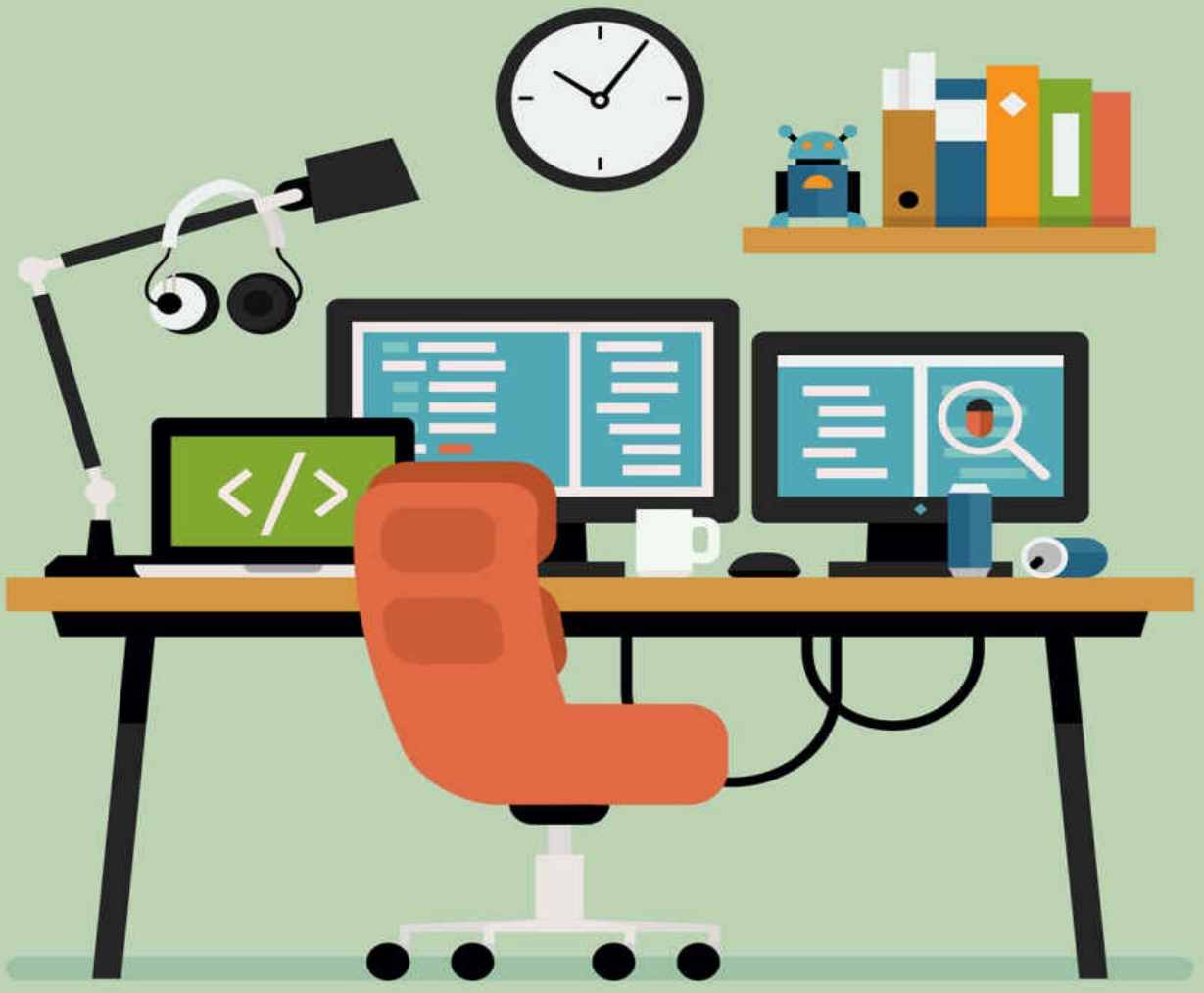
R PROGRAMMING for Beginners

CRASH COURSE



MARTIN LAREDO

Martin Laredo

R Programming for Beginners

For Data Science

Crash Course

January 2017

Table of Contents

The follow eBook is reproduced below with the goal of providing information that is as accurate and reliable as possible. Regardless, purchasing this eBook can be seen as consent to the fact that both the publisher and the author of this book are in no way experts on the topics discussed within and that any recommendations or suggestions that are made herein are for entertainment purposes only. Professionals should be consulted as needed prior to undertaking any of the action endorsed herein.

This declaration is deemed fair and valid by both the American Bar Association and the Committee of Publishers Association and is legally binding throughout the United States.

Furthermore, the transmission, duplication or reproduction of any of the following work including specific information will be considered an illegal act irrespective of if it is done electronically or in print. This extends to creating a secondary or tertiary copy of the work or a recorded copy and is only allowed with express written consent from the Publisher. All additional right reserved.

The information in the following pages is broadly considered to be a truthful and accurate account of facts and as such any inattention, use or misuse of the information in question by the reader will render any resulting actions solely under their purview. There are no scenarios in which the publisher or the original author of this work can be in any fashion deemed liable for any hardship or damages that may befall them after undertaking information described herein.

Additionally, the information in the following pages is intended only for informational purposes and should thus be thought of as universal. As befitting its nature, it is presented without assurance regarding its prolonged validity or interim quality. Trademarks that are mentioned are done without written consent and can in no way be considered an endorsement from the trademark holder.

Introduction

Congratulations on downloading *R Programming for Beginners: For Data Science* and thank you for doing so.

The following chapters will discuss what the R language is and will help you to learn how to use this program for your own. There is so much that you are able to do with this language if you would like to learn how to do data analysis, statistics, and other things that will deal with large amounts of data that you want to work with.

This guidebook is going to take some time to discuss how the R programming language is going to work. We will start out with a short introduction about what this language is all about and some of the syntax that goes with it as well. We will also discuss some of the operators that you will use, especially the arithmetic functions, and then we will move on to making directories, how to work with vectors and creating tables (or matrices), and so much more. You will soon see that this is one of the best programming languages that you are able to use and you will discover how much this is going to help you to get done.

When you are ready to learn how to easily get through large amounts of data with the help of the R programming language, make sure to check out this guidebook and learn how this could help you learn this basic, but powerful, language for your needs.

There are plenty of books on this subject on the market, thanks again for choosing this one! Every effort was made to ensure it is full of as much useful information as possible, please enjoy!

Chapter 1: What is R Programming

The R language is one of the fastest growing programming languages that are available in the arena of data science and statistics. Since it was started in the 1990s as an Academic Demonstration Language, this coding language has developed and extended to include more features and because if this, there are many organizations that are using this software in their daily activities.

This software was created as a free environment for the purpose of educating students. The inventors, Robert Gentleman and Ross Ihaka from the University of Auckland, New Zealand, created this language in order to help educate some of their students. They were well-versed in using the S language (the S language is the commercial programming language for statistics), and they used a syntax that was similar to help create the R language.

Right now, there are about 18 people who are part of the R Development Core Team and who possess the rights to alter the central archive of the source code, but there are others who have been able to help with the development of the R language to help it grow and be more important in an organizations day to day dealings.

So at this point, you may be wondering what the R programming language is. This is a coding language that is used by numerous statisticians, data analysts, and scientists so that they are able to analyze their databases while also carrying out some statistical scrutiny with graphs and figures.

Basically, this is the language that you would use when you want to analyze a large amount of data. If you are charged with going through many files and databases, you would be able to utilize R in order to fetch the data and to tell it how to perform certain processing tasks at the same time. There are also some supplementary packages that you can download and which are great for expanding or simplifying some of the various commands that you do all while you concentrate on analyzing your data.

There are a number of benefits of choosing to go with the R language over another choice include:

- R is free and open source: you will be able to access the R language through its open source license, which basically means that you are able to download the language as well as alter the code for your needs, all without having to pay anything. Because this is open sourced and programmers are able to develop and alter the code, the code is always growing and receiving new enhancements to make it modernized.
- The R language runs anywhere: you will be able to use the coding language no matter where you are. You won't have to download some new software or a new operating system in order to get your code done. You are able to use the R language on UNIX, Windows, and Mac computers.
- The R language is extensible: this language is really powerful and it is able to perform many functions such as manipulation of big databases, statistical analysis, and graphics. But another benefit of working with the R language is that you can extend it out with some add-on packages. You will be able to choose the add-ons that you would like to use based on the task at hand.
- Links with some other languages: there have been some attempts that will merge the R language with some other programming languages, making it easier than ever to get the work done.
- Can do more than just statistics: this language was first developed in order to simplify the process of doing statistics and while the language does concentrate on that a bit, it has developed in order to do even more. It can work on some processes that are non-statistical, such as data processing, graphics, and so much more with your data. When you combine the two together, you will find that all of your data management inside the company can be taken care of in no time.

Installing the R Language

Now that you know a little bit about the history and benefits of the R language, it is time to get it installed on your computer. You can use the code editor that is already on your computer if you have one, but most people choose to install an editor that works with the R language in particular because of all the advanced functions that are on it. The R

Studio is a good choice to go with because it is able to work on all the operating systems that you could be using and it will work smoothly, and without any issues, with the R language. Here we are going to spend some time talking about how to install the R Studio so you can use it as your code editor with this language.

The installation of the R language is not too hard, but there are a few things that you will need to do in order to fine tune the language to make it work the best for your needs. Yu will be able to go online and locate all of the files that are needed in order to install the R language. Pick the link that is with your specific operating system (if you are using a Windows computer, make sure that you pick the link that goes with Windows). From here, you will be able to follow the instructions to make sure that it is connected to your computer and operating system properly.

Installing the R Studio

The R Studio is actually pretty new and it is one of the best editors for the R language. It is simple to use and since it has been designed to work with R, it is often the one that is recommended to use. Installing the R Studio is pretty easy to accomplish. Use the following steps to get the R Studio set up:

- Visit the website https://www.rstudio.com/products/rstudio2/
- Find the button that says RStudio Desktop and click on it.
- Pick the Installer file that is a match to your system version. This would be the version of UNIX, Mac, or Windows that you have.
- Click the Run option.

You may have to wait a few minutes for the R Studio to get loaded on your computer, but if you already have R on your computer, things are going to be easy. The R Studio will be able to detect the installed R language and it will make it easier to use the R language inside of your new R Studio without having to do any more configuration to make them work.

The R Studio is one of the best editors to use with the R programming language, there are some other options that you can choose. The R Gui is a good one to use with a

Windows computer while the R.app is a great one for the Mac OS X version and Linux works well with Vim or Emacs. Yu can choose the editor that you like the best and works with your operating system, but this book is going to focus on using the R Studio since it is the easiest to work with.

Working with your R Studio editor

At this point, we are going to open up the editor and get it started for writing our codes. We will need to find the R Studio icon in the start-up menu. You can then click on File > New > R Script. In this area, you would be able to see four different work areas including:

- Source editor: this is the editor that is going to make it easier to work with the Source-Script-Files. You will be able to type out several rows of your code, save the script to a hard disk, and then carry out several important tasks. These have functions that are similar to other text editors, but it is going to be able to do special tasks that work with the R language.
- Console: this area is where you will be able to type out the instructions or the scripts. You are basically using this as the place to give R the right commands to perform helpful tasks.
- History/Workspace: this is the area where you are going to be able to take a look at the variables that were formed during the R session, along with some other values as well. In this area, you will be able to take a look at the history of commands that you have typed into the R language.
- Files/packages/plots/help: this area is where you will be able to find a wide variety of tools. Each of them is important to helping you to get the code written up:
    - Files: here you will have a chance to look through the files and folders that are in the system.
    - Packages: in this section, you will be able to see a list of all the packages that are installed on the system.
    - Plots: this is the area where R will show your plots, and it often looks like a chart or a graph.
    - Help: this is where you will be able to look through the help system that is

in R which is already built in.

Starting your first R session

Now that you have had time to download the R program and the text editor that you would like to use for this program, it is time to start our first session with R and see how the syntax works and how you would set up your first program. We are going to work on the Welcome Sir program to get a feel for how R works.

So to get started with this, you will need to start up a new session in R and then type in the following command in your console:

```
n <-("Welcome Sir!")
print(n)
```

Once this is in the system, press Enter. The program is going to respond with the following output:

[1] "Welcome Sir!"

And now you have written your first code inside of the R program. Try this out so that you get a chance to explore the text editor and get used to working with R. Now that you have that down, it is time to perform a simple calculation, which is something that you are more likely to do compared to writing out a statement in this program. For this one, type in the following command into the console in order to compute what the sum of these five digits would be:

```
>1+2+3+4+5
```

And the output of this would be 15. You would be able to add in as many of these numbers as you would like, and you can even add in some more operators, such as subtraction or multiplication, in order to get the answers that you would like out of the code. This is just a simple code to help you to get familiar with this program and to see what it would be able to do.

As mentioned above in this chapter, R is a great programming language to use if you would like to learn how to write out some codes to do a data analysis and to work on graphs and other statistics. This program has been used by many organizations who have to deal with a lot of data analysis and can make things so much easier. Take a chance to download the R language and the right text editor and try out a few of these exploratory codes before moving on to the next chapter where we will get more in depth about what you are able to do with this code.

Chapter 2: Some of the Fundamentals to Writing Code in the R Language

Once you have the R language on your computer and set up properly, it is time to move into some of the fundamentals that come with writing code inside this language. Luckily, this is an easy code to learn, but it is still a good idea to learn how it works and what all you are able to do with it. Let's take a look at some of the fundamentals that come with working in R and how they are going to be able to help you use this code.

Looking at the code again

While we wrote a few codes in the previous chapter, we are going to take a look at these again and determine how each of them works and what all of the parts are about. We will also discuss some extra parts that you are able to add into the code that will make it easier to expand on your work, leave comments, and do so much more. Let's start by writing out our code from earlier again:

```
n <- "Welcome Sir!"
print(n)
```

First we will look at the set operator. With the example above, the new variable is going to be called "n" and it sets the "Welcome Sir!" as the string that goes with the variable. You can write it in this way or you can invert it around and have the "n" and the corresponding side to the right, but this is one of the easier ways to write it and keep everything together.

In the second line, there is the print(n) command. This is telling the console that you would like to print the value of "n" which in this case would be the "Welcome Sir!" or another message that you placed into the parameters above. You will be able to make this message as long or as short as you want and the value of the n variable can be any set of numbers or characters that you wish.

If you have worked in programming in the past, you may notice that the n variable doesn't have any reserved words that are proceeding it. This variable is just typed and any value that you like can be set up with it in this case. Once you have the time to run the two lines that are in our code, the n variable is going to turn into an object and then will be loaded into the memory of the system.

At this time, you are ready to run the two-line script. You will just need to go into the editor panel and click on the "Re-run the previous code region"; there should be a button in the editor panel to make this easy to run. If you typed in the code the proper way, you should get the "Welcome Sir!" to show up in the console panel.

Now this may have seemed a bit easy, but going through each of the steps is helpful to showing you how the code works and what all you are able to do with it. As we progress through this book and start to work on things that are a bit more complicated, you will find that working with these same ideas is going to make things easier.

Adding in comments

There are times when you may want to add in some comments to your code. These are lines that other programmers are going to be able to read in the code, but the compiler for R is going to ignore because they are not important to the code running smoothly. If you want to leave a little note for the other programmers to help explain what is going on inside of your code, the comment is going to make this easier. The # sign is the one that you will use in R when you would like to leave a statement.

You are able to add in as many of these operators as you would like to your code, just make sure that they are helping to explain the code well and try to keep them to just doing as many as are needed. Adding in too many can make it harder to read your code for other programmers.

Arithmetic Operators in R

Arithmetic operators can be helpful inside of the R language and can add some more power to the codes that you are trying to write. There are a number of operators that you

are able to do in R including:

(+): this is the addition operator and will allow you to add numbers together.
(-): this is the subtraction operator which will allow you to subtract numbers from each other.
(*): this is the multiplication operator and it allows you to multiply two or more numbers together.
(/): this is the division operator:
(^): this is the power operator
(%) this is the modulator operator.

These are often helpful so that you are able to perform some calculations on your values. You will be able to write them out and just add in the signs like we did in the previous chapter, but you are able to combine together different signs to get the results that you would like. You can even use the result of one of your arithmetic calculations in order to set a variable value. An example of how this would look in your code includes:

```
r <-(10^2)*3
print(r)
```

The result of this would end up being 300 so the value of r is going to equal 300. This is also a good example of using more than one arithmetic operator inside of your code and show the order of operations a bit for what you are allowed to do inside of your code.

Note: if you would like to clear a command from your console, there is a simple way to do this. You would just need to type in cat("\014"). If you would like to run a new command in your console, you just need to type in the code that you are using and then press the Enter button when you are all done typing.

Case sensitivity

When you are working inside of a coding language, it is important that you learn about case sensitivity and how each one is going to handle this issue since each of them are a bit different. Since R is similar to the S language, it is going to be a case sensitive

language. This basically means that using upper case or lower case letters is going to matter. In the codes above, if you typed in (r) and then tried to print off the results for (R), you would get an invalid response because these are considered different things inside the code. Let's look at an example of this below:

```
#basic set operation in R Language
n <-10
print(N)
```

This is going to give you an error because the "N" and the "n" are considered different symbols in the language. You will need to either change both of them to upper case or lower case letters in order to get this to work the right way.

This case sensitivity is going to be the same for all of your functions and commands. This language is going to prefer the lowercase for so keep that in mind when you are working with functions and commands.

Logical Operators

The logical operators are going to be the symbols that you will use in order to compare at least two, but also possibly more, values among themselves. The value is going to be a logical one, which means that the value will be either true or false. In some of the other programming languages that you will use, it is only possible to do a comparison between values that are of the same data type, but when you are working in R, you will be able to do a comparison between different data types on occasion. There are a number of logical operators that you will be able to use in your code including:

(!=): this one means not equal.
(<=): this one means less than or equal to
(>=): this one means greater or equal to
(<): this one means less than.
(>): this one means greater than
(==): this one means equal.

Remember that when you are using logical operators, you will need to get an answer that is either true or false based on how the variables or the values work inside the code. It is going to be comparing these things so the logical operator is in charge of figuring out if they are equal to, greater than, less than or something else compared to the other values in the mix and the sign that you choose to use. You will always get either a true or a false answer.

Working with the control flow structures

When we are talking about the control flow structures, we are working with programming units that will make your code change the flow of execution based on your Boolean condition. For example, a simple control flow would be able to verify the value of an object and then perform or change up its operation according to the value of the object.

Using the if else statement

So basically with the control flow structures, we are going to be teaching the computer system how to make decisions based on the conditions that we set as well as the input that the user puts in, which could be ourselves in this case. The if else statement is one of the best ways to do this because it will set up at least two options, though you can use it for more if needed, so that the system is able to bring back the right information.

Here we are going to look at an example of how this would work. Let's say that we have a variable that is called test and we want to print out a string that says "Greater than five" any time that the value is bigger than five, and then we want a second message that will say "Lower or equal to five" for any times that the amount is equal or lower than five. So when the statement is true, the first part will come up and when it is false the second statement will come up. Here is how the code would look to make this happen:

```
#using if
test <- 10
if(test > 5){
        print("Greater than five")
```

```
} else {
        print("Lower or equal to five")
}
```

This allows the computer to have some freedom in what it is going to bring back based on what the user puts into the computer. The user will be able to put in the information that they want and the program is going to print off the right answer based on the conditions that you have set in the code. You can add in as many of these as you would like. For example, if you would like to have one statement for tests that are over five, one for tests that equal five, and then one for tests that are under five, then you would just need to add this into the code to make it work.

Using the R help feature

There are times when you may need a little bit of help with the commands that you are using inside of the R language. There is a nice HELP feature that you are able to use inside of R that will give you the information that you need to write your code or get answers to some common questions. For example, if you would like to learn more about your "print" command, you would be able to type in the code help(print) and get information about this command.

You will be able to use this Help tab with any of the commands that you would like to get some help with which really makes it helpful for those who are new to the whole idea of programming and want to learn what each of the commands is able to do.

These basics of the R language are going to make it so much easier for you to create a great code that will work in R. they are simple and will make sure that you are able to leave comments, write your code, get help when needed, and even allows you to set it up so that the system is able to make some of its own decisions inside. Take some time to look through these codes and give a few of them a try to see how they will work and how you can use them in our own codes.

Chapter 3: Vectors and Matrices in the R Language

Working with different types of data is one of the biggest reasons that people will choose to work with the R language. We are going to take some look at creating vectors and matrices so that you are able to work with the different options for data inside of your code.

Vectors

Let's get started with the idea of vectors inside of the R language. A vector is going to be a set of values of any type that was used by R, but you need to make sure that all of your values must be from the same mode. If you would like to create a brand new variable inside the memory of the system with the data from another function that you combined before, you would need to use the example below:

```
x<-c(0,1,1,2,3,5,6,13,21,34,55)
print(x)
[1] 0 1 1 2 3 5 8 13 21 34 55
```

The print command is important because it is going to give you the output, which is on the third line, that will result from this code. You can also just type in the name of the variable and then run them to get the data to show up in the console. In this example, the "x" variable is going to defined as your vector. The class function is going to be able to return the name of the class that is associated to all of your objects inside of R, and if you are using the "x" as your argument, you are going to get a numeric value as your result.

So what this means is that we are going to change this up a little bit. If you want to print out the code above, you are just going to get the answers to print out on the screen. But if you would like to name out the class, the program is going to look at the vector and name out the class that it is. Let's look at the example below to see how this would work:

```
x<-c(0,1,1,2,3,5,6,13,21,34,55)
<class(x)
[1] “numeric”
```

Making a Matrix

Now that we have taken some time to learn about vectors we can take some time to create matrices, which are vectors that are two dimensional. In vectors, you will see that the data is set out in a linear manner while with a matrix, the data is going to be laid out in columns and lines, just like a table.

To get started, we are going to create a matrix in R that is going to be empty. To do this, we will need to use the Matrix function by telling the program the number of rows and the numbers of columns that you would like to have. The code for this is pretty simple just using m <- matrix(nrow=2, ncol=2). The result of this will be:

```
     [,1] [,2]
[,1]   NA   NA
[,2]   NA   NA
```

Creating a new matrix that is old is pretty simple, but it is unlikely that you will want to create a chart or a table that is only going to be empty. You are going to want to place information and values into the table in order to create something new and to show your information. Let’s take the simple table that you had before and add values into each part of it using the code below:

```
m[1,1] <- 0
m[1,2] <- 1
m[2,1] < -2
m[2,2] <- 3

>m
```

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

At times you may have a large table and you would like to get some information out of a certain cell inside the table. You would just need to start with the “m” and then pick out the column and row that you want to get the information out of. Let’s say that you want to get information out of 2 and 1, you would just need to write out m[2,1]

Working with lists

Working inside the matrices that you have on your code can make things more interesting. Since you are working with data types and statistics, it makes sense that you would want to work on creating tables and other things to help you to read through the information that you have. Now we are going to move onto making some lists. You are able to turn your vector into a list, but you have to remember that it is going to be a function inside of this language. For example, if you want to take the “x” variable and use it as a function for list, you will need to give it the list mode and class attributes. Let’s take a look at how you would write out a code to make your own list.

```
>x <-list(c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55))

>class(x)
[1] “list”
>mode(x)
[1] “list”
```

Keep in mind that if you tried to execute a function like the max or sum function at this time, you are going to get an error message because there are going to take the vector and make them into an argument, and this is not going to work at all. At this point, if you check the variable mode and class, you are jus going to get “list” as your result, like what is shown above.

So let’s take the example that is above, if you would like to get a certain number on the

list to show up, you will just need to write it into there. You would just need to use >x[] and pick the number out of the list (such as you want to find the fifth number on the list) and that is going to be the output that you would like to find out. This can be useful if you are dealing with a really long list and you want to find a specific value that is somewhere inside the list. You can pick the number of the value and get that to show up rather than having to read through everything.

And now you are ready to create your own vectors, matrices, and lists. These can make it easier to find the information that you want out of the R language and will ensure that you are able to get the right information to come up on the screen when you are doing your own statistics or data analysis with this language.

Chapter 4: Working with the Arithmetic Functions

There are many functions that you will be able to use when working with the R language and we are going to talk about a few of them in this chapter. Since R is a complex language, going over all of them can take some time so we are going to start out with some of the most important ones that you are the most likely to use for your own needs.

First, we are going to talk about some of the arithmetic functions that you are able to use inside of your code. There are quite a few of them and all will work in slightly different ways to help you to write out the codes that you want to use. Let's take a look at these to show you how each of them can work.

Sum function

The first function on the list is the sum function. This one is going to give you the sum of all the values that you place inside the parameters. You can have just two numbers together or you can work with a long list of them. This can be like your own calculator inside the code because you are able to place as many of these into the code as you would like, as long as they are numeric values, and the program will take care of it all. A good example of how this works is below:

```
sum(c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55))
sum(x)
```

This command is going to go through and add together all of the numbers that you have in the parameter, or inside your parenthesis. You are going to end up with a result of 143.

Length function

The next one on the list is the length function. This one is going to take a look of the length of your code and let you know how many numeric values are inside. For example, let's say that you have a code that has eleven numerical values inside of it, the length is

going to be x and it will resume back to eleven. Your code is going to look like the following:

```
>length(x)
[1] 11
```

Average function

The arithmetic average, which is also known as the arithmetic mean, of a group of numeric values is going to be the sum of all the values and then you will divide it by how many values are in the group. So if you had 5+3+4 your average would end up being 4. The R language has a built in function that is meant to calculate the "mean" of the numbers. If you would like to determine what the average of the numeric set is stored in with the "x" variable, you would use the command:

```
print(mean(x))
[1] 13
```

You could also write this out as print(sum(x)/length(x)) in order to get the arithmetic mean or average as well.

Rep function

There are times when you will want to repeat the numerical values that you are using. This function is going to take two arguments and then the value is going to be repeated. You will be able to choose how many repetitions you would like to have created in the code. You will need to have the right parameters in place, but you can use this in order to return a vector that has as many of the repetitions that you wish. A good example of how this is going to work includes:

```
>rep(5, 10)
{1} 5 5 5 5 5 5 5 5 5 5
```

Seq Function:

The sequence function is going to create a numeric sequence that will be based on the arguments that you give. In the simplest form, you will be able to create your own sequence by starting from a simple numeric value and going over to another one. A good example of how this is going to look would include:

```
>seq(1, 10)
[1] 1 2 3 4 5 6 7 8 9 10
```

You can also do this with other options, such as going with decimals and fractions and they would just go up by one unless you add in another part. Now let's say that you would like to make a sequence, but you want it to go up by 2 instead of by the 1 that is listed in the previous part. The code that you would need to use to get this done includes:

```
<seq(1, 10, 2)
[1] 1 3 5 7 9
```

Notice that this one does not stop at 10 because if you added in another 2 to the 9, you would end up with 11, and that goes out of the parameter. The code is going to go up to the second number that you put into the code, but it is not going to go above the code that you are writing.

Sample:

You will be able to create a sample range that has random integer values inside of R by working with your sample function. You are able to use this as a way to create a range of random values that are given like the parameter. If you would like to get some random set of values that go from 1 to 20, you would just need to write it out as the following:

```
sample(1:20)
```

note that if you do go through and execute this sketch a few times, it is going to give you the same range of values, but the order is going to be random and will change up a bit

each time. You are also able to list out a range of values that are inside a limited set of results by adding in another parameter to the mix. For example, if you would like to have a random set of values that are in between 5 and 20 but you only want to have 6 numbers show up, you would need to type in the following code:

```
sample(5:20, 6)
```

Another thing that you may want to do with your code is to use the replace parameter and you are able to set it to be either False(F) or True(T). this parameter is going to set the result to show, or not to show, the repeated values based on what you are setting as the value. The example above will show you some sample results, but you need to remember that each time you execute the function, you are going to get some different results. Let's take a look at how this works with the following code:

```
>sample(5:15, 5, replace=T)
[1] 10 7 11 7 15.
```

Working with all of these different functions is going to help to make the coding easier when you are working in R. There are many times when you would need to use some of these functions and putting them all together, or learning which one is going to be the best for you will help you to get the program to work in the manner that you wish.

Chapter 5: Working on a Directory

At some point, you will need to create a working directory. This is going to be the folder that R is going to save and load your files by default. You would be able to change the directory that the R language is using or it may just go to a predetermined spot on your system based on what is set up already. If you would like to find out the full path to the working directory, you are going to need to use the getwd() function.

If you would like to go through and make changes to which place is the working directory, you would need to use the setwd() function because this is going to make a shorter path in your system. So say you want to use the "tmp" named folder in the root of your system, you would just need to write out the following code to make it work:

setwd("c:/tmp")

After you have had some time to set up this new working directory, make sure to cop your myfile.csv to it so that you are able to use this working directory. Once you have bee able to copy the csv to the new working directory, you will be able to run the following command into it (you won't have to use the full path because R is going to use the working directory as the base)

read.csv("myfile.csv")

You should then get the following result on your R console:

>read.csv("myfile.csv")
col1 col2 col3 col4

1 a 100 200 500
2 b 300 700 8900
3 c 400 2300 8700
4 d 100 100 3400

5 e 500 600 4300
6 f 600 9800 5300

This may seem pretty simple to complete, but the myfile.csv is a sample data file that is inside of R. It was created for the sole purpose to allow you a chance to play around with the external files in R so you will just need to save this to your new working directory and type it into the system to get the table that is above.

You can also set the content of your file to a brand new variable inside of R that is called “mydata”. You would need to use the following script in order to make this happen:

mydata <- read.csv(“myfile.csv”)

Working with percentages

At some point, you will want to use the myfile.csv in order to work on some percentages. You will just need to use the prop.table function in order to get started on this. A good calculation to do this with would be the percentage of the col4 values to consider the total values.

The prop.table function is an interesting one to use because it is going to take a generic vector in your code and use it as the argument before returning another vector to you. Given that the object of interest in your col4 is the values, that is the parameter that you are going to use and you will be able to see and use these results in order to set up a new column inside of the mydata data frame.

So for this example, we are going to use the “$colunnname” approach in order to get the new column value as well as the new column setting. Let’s take a look at how this would look inside of your code:

>mydata$c4percent <- prop.table(mydata$col4)
>print(mydata)
col1 col2 col3 col4 c4percent

```
1       a 100 200 500 0.01607717
2       b 300 700 8900 1.28617363
3       c 400 2300 8700 0.27974277
4       d 100 100 3400 0.10932476
5       e 500 600 4300 0.13826367
6       f 600 9800 5300 0.17041801
```

One thing to note with this table is that the percentage is going to be placed in a decimal form rather than having it in the percentage. So for example, if you have an answer that is .50, this means that it is 50% and so on.

Creating a graph using the plot function

When you are trying to work with your data analysis, there are many times that you will want to create a graph of some kind with the information that you are using. Visual resources are the best way to do this because they are more interesting rather than just looking at a list of numbers and they are really great at showing some of the differences between the data points that you are using.

We are able to use the plot function, as well as some of the other functions that are in the R language in order to make your graphs. We are going to make a line graph in this section using the values from the col2 column in the Y axis.

The plot function is able to handle just one argument at a time as long as the argument is a proper data.frame. but the plat can also take another eight optional arguments. The most common are going to be the x and the y, which are called the two Cartesian coordinates. In order to achieve a goal of making our graphs, we are going to need two vectors for this example which would be the values in col2 and the values in col1.

The column vector is able to be retrieved with the references that we learned before. The two commands are going to give you the same results from the col2 columns so use the following to get this:

```
mydata$col2
mydata[,2]
```

This is a really simple thing that you are going to be able to use the plot function in order to use the x and y arguments. The code that you will need to use for the plot function with these two as the arguments include:

```
plot(mydata$col1,mydata$col2)
```

This command is going to generate a simple graph that will show up on the plot tab on your lower right panel. When you create this graph, you will probably notice that it doesn't have a line between the "y" axis values. If you would like to have this show up, you would need to add in a second part to this in order to add in the lines. You would just need to use the "lines" function to make this happen with the following code:

```
plot(mydata$col1,mydata$col2)
lines(mydata$col2)
```

The command that you put in above is gong to add in some of the lines that you need to connect things together and make the graph look a little bit better. Now that this looks a bit better, it is time to add in some captions for the value of y and x? some of the other parameters that you may need to use in your plot include:

Main: this is going to set a caption on the upper level of your graph
Xlab: this is going to add in a caption on the x axis.
Ylab: this is going to add in a caption on the y axis.

If you would like to set your upper caption to the example that we did above so that it says "Plot Example", the x axis caption to be "Row Description" and the y axis caption to say "Row Values" you would need to use the following code:

```
plot(mydata$col1, mydata$col2, main= "Plot Example", xlab= "Row Description", ylab
= "Values")
        lines(mydata$col2)
```

With the help of this code, you will have the right titles on the different parts of the graph so that it makes a bit more sense and it helps you to figure out what is in the graph and for others who may want to look over the graph to understand what is going on inside of it.

Saving your graph

Once you have created a graph with the help of R, you will want to make sure that you save the graph to use it again later. Inside of R, you will usually save it as either a jpeg file or a pdf file. If you would like to make this graph into a pdf file, you would need to use the pdf function in order to get it saved right. Use this code to make it happen:

```
pdf("PlotExample.pdf", width=7, height=5)
    plot(mydata$col1, mydata$col2, main="Plot Example", xlab="Row Description",
ylab="Values")
    lines(mydata$col2)
    dev.off()
```

After you run this script inside of your compiler, you will see that the file will now be named PlotExample.pdf and you will see it show up on the working directory. You will be able to go onto your working directory and open up this file any time that you would like to look at the graph again or make some change to the graph at a later time.

Writing data onto your file

Once you create one of your tables or your graphs, you will be able to use the write.table function in order to save data over to the file that you want. You will need to have two arguments present in order to get it to happen and these include the file name and the data frame that you want to export.

To get started on this task, you will need to create a data frame from mydata data frame, which is holding onto the myfile.csv in your memory, but you only need to use the col4 and col1 as your source. The way that you would be able to do this includes using the

following code:

```
myframe <- data.frame(maydata$col1, mydata$col4)
```

If you test out this myframe variable, the results that you see will include:

```
>myframe
        mydata.col1 maydata.col4
1        a       500
2        b       8900
3        c       8700
4        d       3400
5        e       4300
6        f       5300
```

Now if you would like to create this into a csv file, you will use the write.table as the function and the myframe as your argument with the code of write.table(myframe, file = “myfile.txt”).

With this code, you will be able to browse the working directory that is in your operational system and then you can go through and locate the file that you just created. In time, there is another way to load up the csv file with the read.table function. This is going to work with any of your text files that have been structured in the pattern of “data.frame” or a file that was originated using the “write.table” export method. If you would like to load your myfile.txt into your data frame object, while using the read.table function, use the following code:

```
#importing
newframe <- read.table(“myfile.txt”)
newframe
```

Creating some of your own working directories and learning how to make percentages, save and change some of your charts, and even how to create a chart and name all of the different parts is important if you would like a chance to analyze the data that you are

using the R programming language. Take some time to look through this chapter and try a few of the options to make sure that you get the hang of it all!

Chapter 6: Some of the Other Tasks You Can do in R

There are so many things that you will be able to do inside of your R programming language and we are going to use this chapter in order to get through a few of these to help you to make your code work even better with this language.

Loops

First we are going to take a look at loops inside of this language. Loops are basically sequences of instructions that will keep on repeating until a condition is met. In this language, these are handy if you would like to execute a task repeatedly in all elements of your vector. There are a few different types of loops that you can use in this program including the following:

For Loops

The For loop is one of the loop implementations that you can use and it is going to take as argument the vector that it will scrape as well as the name of the variable that is being used. Let's look at the example below:

```
f <- c(1,2,3,4,5)
for (I in f){
        print(past("Line", as.character(i)))
}
```

This code is starting out in the first line with initializing the "f" variable with the generic vector that has five elements, which is the (1,2,3,4,5). Then it moves on to declaring the "For" and the "I" variable is going to take the value from each member of the vector through each iteration of your loop. Inside of the loop, there will be a message printed that says "Line" combined with the position value of the number that is being used. This type of loop is set up to keep on going until all of the vector parts have been gone through and then it will end at that time.

While Loop

The while loop is another option that you will be able to use when running a loop. The while loop will repeat a certain part of the code until a condition that you set is fulfilled. The basic syntax that you would use for the while loop would include:

```
while (Boolean condition){
        your code here…
}
```

So in order to complete the logic for the "for" loop with the "while" loop, we would use the script that is below:

```
f <- c(1,2,3,4,5)
i<- 1

while (i <= length(f)){
        print(paste("Line", as.character(i)))
        i<-i+1
}
```

You will need to be careful when you are using the while loops because they do not have an increment control that is going to happen after each of the iterations, while the for loops will have this increment control. That is why the example above is going to have that placed into the code to make sure that you go up by the right increment with each loop.

Making a Function in R

Inside of a programming language, the functions are going to be the units that will perform a logical task and may, and sometimes may not, return a result. You have already used a lot of functions in this book already, but as you start to work with more complex parts of R, you may need to write out some of your own functions. Let's look at

a sample of writing your own function with the example below:

```
myfunction <- function(x){

print(x)
}
```

To execute this function, you would write out the following pieces of code:

```
#execute myfunction
myfunction("R for Beginners")
```

Since the R language is a script language, the function is not going to be named like you would with some of the other programming languages you may use. The function is going to be associated with the variable and then loaded into the memory of the system.

So let's take some time to do another example. For this one, we are going to take a look at a series of numbers and then figure out how many times a specific number will occur in your numeric vector sequence. Let's take a look at this generic vector and see how it will work:

```
c <- c(1,2,3,3,4,5,6,7)
```

Now you are able to look at this and see that there are 2 occurrences of the number three. But what would you need to type into the program in order to get this to show up as the result in the program. Creating a function would help to make this a possibility.

Note: when you are creating some of your own functions, a good idea to try out is to write the code without using any function declaration and then see what the results will look like. Once you have found one that floats, it is time to move on and create the function that you need.

To make the sure that the program is able to type out the amount of occurrences of the number that you pick, you will need to use the FOR function. A good example of how to

make the program work with counting out how many times a number occurs (in this situation we are going to use the number 3) you would need to use the following code:

```
x <-c(1,2,3,3,4,5,6,7)
c <- 0

for (n in x({
        if (n == 3)
        c <- c+1 #greater than v
}

print(c)
```

Now let's take a moment to see what is going on in this code. The first line is going to create a variable that is a generic vector. Then the c variable is going to be the counter and it is going to be initialized at zero, or 0. The loop will then be declared in order to run the elements in the "x" one by one, and then they will load at every iteration in the "n" variable. Then once the value of 3 is found out, the "c" variable will be incremented. Finally, the program is going to print how many times the number three shows up in your vector.

Working with packages in R

Inside of R, packages are going to be groups of functions that can be added into your environment to make it have extra functionality. To install a package into the R environment, you will be able to find that package from online and then run the command install.package.

You will be able to visit the CRAN, which stands for the Comprehensive R Archive Network, in order to see a list of packages that are available for R that is updated often. You can look through this list and then download the packages that you would like to use right from this area.

You will notice as you use the R language that their community is pretty active and there

is a growing number of packages that are out there that you are able to use. Here we are going to take a look at how to download one of these packages and you will be able to use the same idea to add in any of the packages that you want.

We are going to use the “Plyr” package because it contains a lot of tools that are great for breaking down large amounts of data so that it is more manageable. You can go to the CRAN site and then download the proper release of “plyr” package so that it works on your computer. We are going to concentrate on downloading the 1.8.1 version but you can pick the newer version if you like or whichever works the best for your computer.

Once the download is ready, you will want to make sure to copy this to your working directory and then use the command “install.package” using the zip file as your argument. The command that you will want to use includes:

install.packages (“~/plyr_1.8.1.zip”, repos = NULL)

Also, many of the packages that you choose will allow you to do this visually by clicking on the install button that is on the panel of the package. You can even save some hassle by having the R environment do the downloading for you. You just need to use the install.packages function and then name the package that you would like to load onto your computer.

There are many different types of packages that you will be able to use with your system and it is going to depend on what exactly you would like to do with the R program before you pick the right packages. Some people use one or two packages and get everything done and others may need to download quite a few before making it work the best for them!

Conclusion

Thank for making it through to the end of *R Programming for Beginners: For Data Science*, let's hope it was informative and able to provide you with all of the tools you need to achieve your goals whatever it may be.

The next step is to download the R programming language and get the editor all set up. Then you are ready to start writing out some of the codes that you want inside of this great program. You can try out a few of the codes that are in this guidebook to see what it is able to do and to make sure that you are used to working with this system before you start to use it on your own database and using it to do a data analysis.

This guidebook has spent some time talking about the important things that you will want to learn how to do with the R programming language. We will not only discuss some of the history and the benefits of using this coding language, but also about the different functions and operators that can be used in this language, how to set up your first code and some of the different parts of the R code, and even how to create directories, lists, and matrices inside of this language.

When you are ready to start learning how to work with the R language in order to work on your own data analysis in a way that makes sense for your needs, make sure to read through this guidebook and learn how to get started.

Finally, if you found this book useful in anyway, a review on Amazon is always appreciated!