Introduction to R

cjfiscus

2020

Introduction

The R programming language is widely used for statistical computing in the biological sciences and beyond. This tutorial will highlight some of the capabilities of R and provide a rudimentary introduction to computer programming.

For the remainder of this tutorial, the text with a grey background is R code and the text that is prefaced with "##" is the output of the previous code statement.

Installation

To work through this tutorial, you must first have access to an R computing environment.

Method 1: Use RStudio in the Cloud
Make an account and setup a new project.

Method 2: Download and install R for your platform of choice here.

RStudio is an integrated development environment (IDE) for R and is highly recommended for new users. Download and install RStudio for your platform of choice here.

To follow along with this tutorial, first open RStudio. Type the code into the Console at the bottom of the screen and press Enter after each line.

Code and Comments

When you write code, it is useful to label pieces of code with human readable text that describes what is happening. This can be done using comments. Comments are specified using "#" at the beginning of the line and are ignored by the interpreter (i.e the code on those lines, if any, is not executed).

```
# this is a comment
```

Numbers, Variables, and Vectors

R can understand simple arthimetic operations (e.g. +, -, *, /).

```
# addition
5 + 4
```

```
## [1] 9
```

```
# subtraction
5 - 4
```

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

```r
# multiplication
5 * 4
```

```
## [1] 20
```

```r
# division
5 / 4
```

```
## [1] 1.25
```

However, simply doing arithmetic only has limited utility. It is more useful to be able to capture the output of an operation so that we can use it later. This is done using variables. We can assign the output of an operation to a variable using the assignment operator "<-" or "=". We can then use the variable as if it was a number. Finally, we can view the value of variable by typing the variable name.

```r
a <- 5 + 4
a
```

```
## [1] 9
```

```r
a * 2
```

```
## [1] 18
```

Variables can hold any type of information, not just numbers. For instance, we can assign a string (a bit of text) to a variable.

```r
a = "hello"
a
```

```
## [1] "hello"
```

Vectors are a collection of pieces of data. Vectors are specified with the syntax c("thing1", "thing2", etc.). R is particularly useful for statistics since it supports vector-based operations. This means that we can rapidly manipulate all of the values in a vector in the same way without the trouble of having to loop through each item in the vector as in other programming languages.

```r
a<-c(5, 10, 15, 20)
a
```

```
## [1]  5 10 15 20
```

```r
a/5
```

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

Data in a vector can be accessed using indexes. In R, the first item in a vector is index 1. Prefacing an index with "-" means to exclude this value. We can also subset a vector using indexes.

```r
# define vector a
a<-c(5, 10, 15, 20)

# first item in a
a[1]
```

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

```r
# a without the value at index 2
a[-2]
```

```
## [1]  5 15 20
```

```r
# subset
a[2:3]
```

```
## [1] 10 15
```

Conditional Statements

Conditionals are one of the most basic control flow statements used in computer programming and are used to execute a routine given a test of a condition. The most common conditional statement is the "if" statement. Here is how conditional statements are used in computer programming:

if a condition is true
do this

Conditional statements can also have alternate instructions to perform if a statement is not true.

if a condition is true
do this
else
do this instead

The syntax for conditional statements in R is as follows:

```r
# set x equal to 1
x <- 1

# if x is greater than 0 print x is greater than 0
if(x > 0){
print("x is greater than 0")
}
```

```
## [1] "x is greater than 0"
```

```r
# reassign x to -1
x <- -1

# if x is greater than 0, print x is greater than 0
# if x is not greater than 0, print x is less than 0
if(x > 0){
```

```
  print("x is greater than 0")
} else {
  print("x is less than 0")
}
```

```
## [1] "x is less than 0"
```

Loops

Loops are another type of control statement that are used when code is to be reused multiple times or when you want to iterate over a set of numbers or things. Imagine that you wanted to print all the numbers from 1 to 3.

This could be done like this:

```
print(1)
```

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

```
print(2)
```

```
## [1] 2
```

```
print(3)
```

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

Imagine if instead I wanted to print all of the numbers from 1 to 1 million. Written this way, this would take 1 million lines of code!

It is much more efficient to do this using a loop. In this case we are using a *for* loop to iterate over the numbers 1 to 3.

```
# for i (a number) in 1 through 3, print i
for (i in 1:3){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
```

```
# we can also iterate over vectors
# In this example length(vec) is 3 so
# this is the same as saying for 1:3

# define vec with three things
vec<-c("thing1", "thing2", "thing3")

# showing that length(vec) is 3
length(vec)
```

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

```
# loop over vector and print each item
for (i in 1:length(vec)){
  # print the item at index i in vec
  print(vec[i])
}
```

```
## [1] "thing1"
## [1] "thing2"
## [1] "thing3"
```

Functions

Functions are pieces of reusable code that perform a task. You can think of functions as factories. Factories take raw materials, process them, and export finished products. Similarily, functions take raw data, do operations on the data, and then export the result of the operations.

Functions are specified with the syntax "functionName()" where arguments (the inputs) can be supplied to the function within the parentheses. In the last code chunk, we are using the combine function c(). This function combines all of the arguments into a single vector. The arguments provided to the function are the things that are to be combined.

You can view the help page for any function using the syntax "?functionName". Help pages describe how functions operate and what arguments they can take. As you might have noticed from the previous code chunk, you can also assign the output of a function to a variable.

```
# combine these numbers into a vector
c(5,10,15,20)
```

```
## [1]  5 10 15 20
```

```
# functions can be given as the argument into another function
# the vector created by c() is being used as an argument provided
# to the sum function
sum(c(5,10,15,20))
```

```
## [1] 50
```

```
# view the help page for the c() function
?c
```

You can also write custom functions.
The syntax is:

"mySuperCoolFunction<-function(arguments){
code goes here
}"

```
# define a vector
v<-c(1, 2, 3, 4)
```

```r
# define a function to add 1
addOne<-function(vector){
  # add one to each value in vector
  vector<-vector + 1
  # return the value of vector from the function
  return(vector)
}

# run the function, providing the vector as an argument
addOne(v)
```

```
## [1] 2 3 4 5
```

Basic Statistics in R

R has some built-in functions for basic descriptive statistics. Here are some examples:

```r
numbers<-c(3,6,9,12,15,18)

# mean
mean(numbers)
```

```
## [1] 10.5
```

```r
# median
median(numbers)
```

```
## [1] 10.5
```

```r
# standard deviation
sd(numbers)
```

```
## [1] 5.612486
```

```r
# variance
var(numbers)
```

```
## [1] 31.5
```

R can also do some more advanced things too. Here we are calculating the correlation and building a linear model using two variables from the built-in iris dataset.

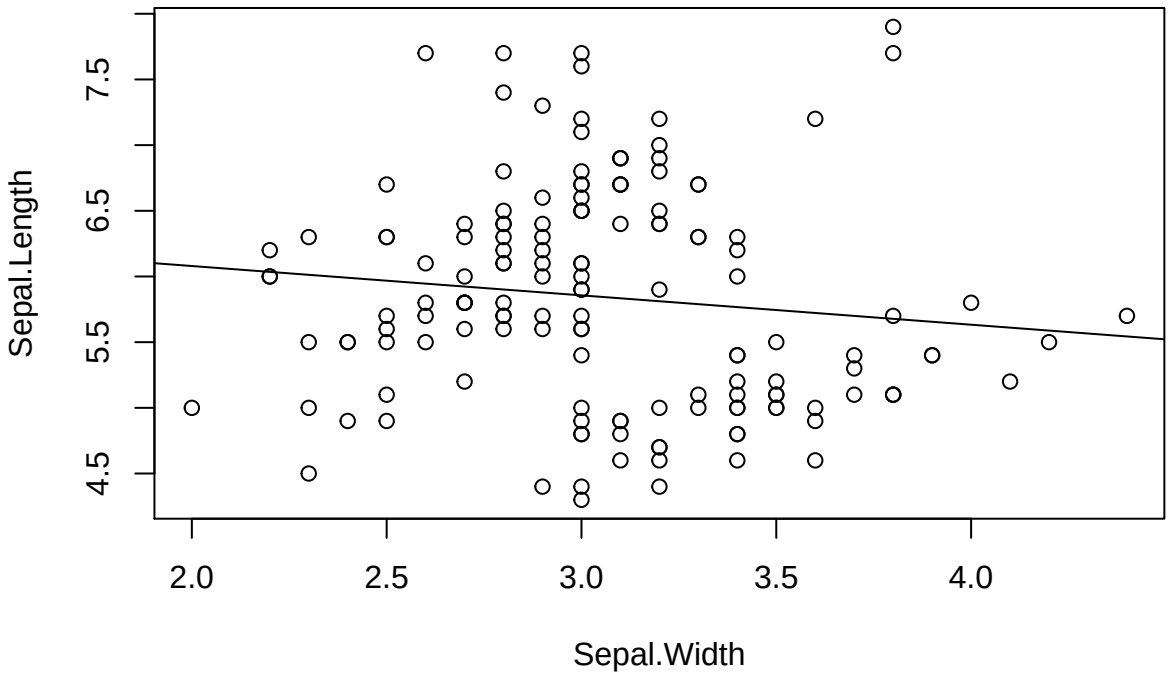
```r
# load iris dataset
data(iris)

# is there a correlation between Sepal Length and Sepal Width?
# "$" is used to subset columns in the dataset (more on this shortly)
cor(iris$Sepal.Length, iris$Sepal.Width)
```

```
## [1] -0.1175698
```

```r
# build a linear model
model<-lm(Sepal.Length ~ Sepal.Width, data=iris)
print(model)
```

```
##
## Call:
## lm(formula = Sepal.Length ~ Sepal.Width, data = iris)
##
## Coefficients:
## (Intercept)  Sepal.Width
##      6.5262      -0.2234
```

```r
# quick plot of model
plot(Sepal.Length ~ Sepal.Width, data=iris)
abline(lm(Sepal.Length ~ Sepal.Width, data=iris))
```



Data Frames

Data frames are R objects that store data in a table format. Data frames consist of rows (horizontal) and columns (vertical). R provides many functions to interact with data frames.
Here are some examples:

```r
# assign iris to df
df<-as.data.frame(iris)

# determine the dimensions of a data frame (rows, columns)
dim(df)
```

```
## [1] 150   5
```

```r
# what are the names of the columns in df
colnames(df)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

Similarly to vectors, data frames can be subsetted. The syntax is "df[rows,columns]". If the rows or columns are not specified then R assumes that you mean all rows or columns

```r
df<-as.data.frame(iris)

# the entire dataset
# df[,] or "df"

# subset the first two rows
df[1:2,]
```

```
##   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
```

```r
# subset the first two columns
df[,1:2]
```

```
##     Sepal.Length Sepal.Width
## 1            5.1         3.5
## 2            4.9         3.0
## 3            4.7         3.2
## 4            4.6         3.1
## 5            5.0         3.6
## 6            5.4         3.9
## 7            4.6         3.4
## 8            5.0         3.4
## 9            4.4         2.9
## 10           4.9         3.1
## 11           5.4         3.7
## 12           4.8         3.4
## 13           4.8         3.0
## 14           4.3         3.0
## 15           5.8         4.0
## 16           5.7         4.4
## 17           5.4         3.9
## 18           5.1         3.5
```

```
## 19          5.7         3.8
## 20          5.1         3.8
## 21          5.4         3.4
## 22          5.1         3.7
## 23          4.6         3.6
## 24          5.1         3.3
## 25          4.8         3.4
## 26          5.0         3.0
## 27          5.0         3.4
## 28          5.2         3.5
## 29          5.2         3.4
## 30          4.7         3.2
## 31          4.8         3.1
## 32          5.4         3.4
## 33          5.2         4.1
## 34          5.5         4.2
## 35          4.9         3.1
## 36          5.0         3.2
## 37          5.5         3.5
## 38          4.9         3.6
## 39          4.4         3.0
## 40          5.1         3.4
## 41          5.0         3.5
## 42          4.5         2.3
## 43          4.4         3.2
## 44          5.0         3.5
## 45          5.1         3.8
## 46          4.8         3.0
## 47          5.1         3.8
## 48          4.6         3.2
## 49          5.3         3.7
## 50          5.0         3.3
## 51          7.0         3.2
## 52          6.4         3.2
## 53          6.9         3.1
## 54          5.5         2.3
## 55          6.5         2.8
## 56          5.7         2.8
## 57          6.3         3.3
## 58          4.9         2.4
## 59          6.6         2.9
## 60          5.2         2.7
## 61          5.0         2.0
## 62          5.9         3.0
## 63          6.0         2.2
## 64          6.1         2.9
## 65          5.6         2.9
## 66          6.7         3.1
## 67          5.6         3.0
## 68          5.8         2.7
## 69          6.2         2.2
## 70          5.6         2.5
## 71          5.9         3.2
## 72          6.1         2.8
```

```
## 73          6.3         2.5
## 74          6.1         2.8
## 75          6.4         2.9
## 76          6.6         3.0
## 77          6.8         2.8
## 78          6.7         3.0
## 79          6.0         2.9
## 80          5.7         2.6
## 81          5.5         2.4
## 82          5.5         2.4
## 83          5.8         2.7
## 84          6.0         2.7
## 85          5.4         3.0
## 86          6.0         3.4
## 87          6.7         3.1
## 88          6.3         2.3
## 89          5.6         3.0
## 90          5.5         2.5
## 91          5.5         2.6
## 92          6.1         3.0
## 93          5.8         2.6
## 94          5.0         2.3
## 95          5.6         2.7
## 96          5.7         3.0
## 97          5.7         2.9
## 98          6.2         2.9
## 99          5.1         2.5
## 100         5.7         2.8
## 101         6.3         3.3
## 102         5.8         2.7
## 103         7.1         3.0
## 104         6.3         2.9
## 105         6.5         3.0
## 106         7.6         3.0
## 107         4.9         2.5
## 108         7.3         2.9
## 109         6.7         2.5
## 110         7.2         3.6
## 111         6.5         3.2
## 112         6.4         2.7
## 113         6.8         3.0
## 114         5.7         2.5
## 115         5.8         2.8
## 116         6.4         3.2
## 117         6.5         3.0
## 118         7.7         3.8
## 119         7.7         2.6
## 120         6.0         2.2
## 121         6.9         3.2
## 122         5.6         2.8
## 123         7.7         2.8
## 124         6.3         2.7
## 125         6.7         3.3
## 126         7.2         3.2
```

```
## 127          6.2         2.8
## 128          6.1         3.0
## 129          6.4         2.8
## 130          7.2         3.0
## 131          7.4         2.8
## 132          7.9         3.8
## 133          6.4         2.8
## 134          6.3         2.8
## 135          6.1         2.6
## 136          7.7         3.0
## 137          6.3         3.4
## 138          6.4         3.1
## 139          6.0         3.0
## 140          6.9         3.1
## 141          6.7         3.1
## 142          6.9         3.1
## 143          5.8         2.7
## 144          6.8         3.2
## 145          6.7         3.3
## 146          6.7         3.0
## 147          6.3         2.5
## 148          6.5         3.0
## 149          6.2         3.4
## 150          5.9         3.0
```

```
# subset the first two rows and first two columns
df[1:2,1:2]
```

```
##   Sepal.Length Sepal.Width
## 1          5.1         3.5
## 2          4.9         3.0
```

```
# subset the first and third column (use concatonate function) and only the first five rows
df[1:5,c(1,3)]
```

```
##   Sepal.Length Petal.Length
## 1          5.1          1.4
## 2          4.9          1.4
## 3          4.7          1.3
## 4          4.6          1.5
## 5          5.0          1.4
```

Columns can also be selected by name using "df$colname." Among other uses, this is a quick way to calculate summary statistics about a column.

```
df<-as.data.frame(iris)

mean(df$Petal.Length)
```

```
## [1] 3.758
```

Working with External Data

Working Directories

When running code, the working directory is the directory (or folder) where the code is being executed. This is the default location that data will be read and written to disk.

```
# check the working directory
getwd()
```

```
## [1] "/Users/cjfiscus/Google Drive/Teaching/BIOL119/2020/Drafts"
```

```
# set the working directory to the Desktop
# Note: this was done on MacOS, will be different for Windows
setwd("~/Desktop")
```

Reading and Writing Data

Data can be read and written to disk from R. The basic function to read data into R is "read.table". Similarly, the function to write data is "write.table".

```
# read data in
df<-read.table("reallycooldata.txt")

# write data out
write.table(df, "reallycooldata.txt")
```

Packages

One of the most useful facets of the R programming environment is the ability to extend the functionality of the language using packages. Packages are collections of code and/or data that are written and submitted by users to repositories, or databases of packages. To date, there are over 10,000 packages that are available to download from the Comprehensive R Archive Network (CRAN). To use a package, you must first download and install it. Once it is installed, the package must be loaded before use. Note, "supercoolpkg" is merely a placeholder and not the valid name of an R package so the code below will fail if you try to run it.

```
# install package
install.packages("supercoolpkg")

# load package
library("supercoolpkg")
```