Introduction to R

Using basic functions in R

Inspecting function documentation

There are many functions in base R that you can use (in a later tutorial, we will discuss how to create your own functions!). In order to view documentation for a function in R, you can type ?function_name or search for the function name in the help tab. As practice, let's explore the matrix() and mean() functions in R by typing ?matrix and ?mean into the console. Uncomment (by deleting the #s) the code below and run it in the console to inspect the documentation.

Under usage for the matrix()function, you should see the following:
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
The matrix function has 5 distinct parameters. All of them have default values. For example, if you don't put any data into the function, the resulting matrix will be made up of NA values (this is how missing data is generally coded in R). Additionally, the function will have 1 row and 1 column by default and data will be filled by columns because byrow=FALSE by default. The matrix will not have any row or column names because dimnames=NULL by default.

```
#?matrix
#?mean
```

Because all parameters of the matrix() function have defaults, we could call matrix() with no inputs and we would get a 1x1 matrix with NA values and no dimension names. We can also pick and choose whatever parameters we do want to fill in and ignore anything that we want to leave as defaults.

```
#call matrix() with no inputs
matrix()
```

```
##      [,1]
## [1,]   NA
```

```
#make a 2x3 matrix of NAs
matrix(nrow=2, ncol=3)
```

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

In contrast, the mean() function has some required parameters. When you type ?mean into the console, you should see the following:
mean(x, trim = 0, na.rm = FALSE, . . .)
The parameter x has no default value; you must specify the values that you want to take the mean of. The other paramters are given defaults: trim=0 and na.rm=FALSE. If you read the descriptions for these parameters, you will see that trim allows you to calculate a trimmed mean (i.e., eliminate some proportion of extreme values before calculating the mean) and na.rm allows you to remove missing (NA) values before calculating the mean. By default, R will not do any trimming and NAs will not be removed. This can cause issues (see below):

```
#create some data and save it as data1 and data2
data1 <- c(1,2,3,4,7,NA)
data2 <- c(1,2,3,4,7)

#calculate the mean of data1 and data2
#note that data1 has a mean of NA because there was an NA value that was not removed
mean(data1)
```

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

```r
mean(data2)
```

```
## [1] 3.4
```

```r
#now explicitly set na.rm=TRUE and recalculate the mean of data1. Now we get the same as data2
mean(data1, na.rm=TRUE)
```

```
## [1] 3.4
```

Calling functions in R

When using functions in R, you can make it clear which inputs refer to which parameters by either a) using the order of the parameters specified in the usage or b) naming them explicitly.

```r
#note that these two calls to the mean() function will return the same output
mean(data1, .2, TRUE)
```

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

```r
mean(x=data1, trim=.2, na.rm=TRUE)
```

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

```r
#however, the following code will give an error because "TRUE" is not a valid input to trim,
#which is the second parameter listed in the usage
#mean(data1,TRUE)

#to specify na.rm=TRUE but leave the trim=0 default as is, we simply do the following:
mean(data1, na.rm=TRUE)
```

```
## [1] 3.4
```

```r
#(note that, since data1 still matches to x, which is the first parameter in usage,
#we do not need to specify x=data1)
```

Data types and structures

There are four basic data types in R (that you need to know): character, numeric, integer, logical. There are five basic ways to store data in R: vectors, data frames, matrices, arrays, and lists.

You can learn about these data types below

Numerics and integers

```r
#to store data in some variable name, use either = or <-
#to save the number 5 as "number":
number <- 5
number = 5 #does the same thing

#print number:
print(number)
```

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

```r
#find out the class of number:
class(number)
```

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

```
#change number to an integer and re-save it as number2:
number2 <- as.integer(number)

#inspect the class of number2 to see that it is an integer:
class(number2)
```

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

Characters

```
#save a character string in a variable named message
message <- "welcome"

#print message:
print(message)
```

```
## [1] "welcome"
```

```
#inspect the class of message:
class(message)
```

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

Logicals

Logical data is either TRUE or FALSE
In R, TRUE=1 and FALSE=0

```
#save the logical TRUE as a variable called outcome:
outcome <- TRUE

#print outcome
print(outcome)
```

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

```
#inspect class of outcome
class(outcome)
```

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

```
#note that, weirdly, outcome+outcome=2
outcome+outcome
```

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

In R, you can test a statement to see if it is TRUE or FALSE. Note that R allows you to make comparisons accross variable types: integers may be compared to numerics and logicals may be compared to integers/numerics. For characters, comparatives are assessed using alphabetical order (letters earlier in the alphabet are “smaller”):
1) == means “is equal to”
2) != means “is not equal to”
3) > means “greater than”; >= means “greater than or equal to”
4) < means “less than; <= means”less than or equal to"

```
#is 5 equal to 3?
5==3
```

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

```
#is 5 not equal to 3?
5!=3
```

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

```
#is 5 less than 3?
5<3
```

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

```
#is 5 greater than 3?
5>3
```

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

```
#is 5 greater than 5?
5>5
```

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

```
#is 5 greater than or equal to 5?
5>=5
```

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

```
#is 5 equal to 5?
5==5
```

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

```
#is "hello" equal to "hello"?
"hello" == "hello"
```

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

```
#is "hello" equal to "goodbye"?
"hello" == "goodbye"
```

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

```
#is "hello" greater than "goodbye"? (in other words is "hello" after "goodbye" alphabetically?)
"hello">"goodbye"
```

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

```
#is TRUE == 1?
TRUE==1
```

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

```
#is FALSE==0?
FALSE==0
```

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

Vectors

```
# The easiest way to create a vector is by using the c() function
vec1 <- c(2,3,4,5)
print(vec1)
```

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

```
# Note that, if you include multiple data types in a vector, R will change all values to the same type
vec2 <- c(7,3)
vec2
```

```
## [1] 7 3
```

```
class(vec2)
```

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

```
vec3 <- c(7,3,"hello")
vec3
```

```
## [1] "7"     "3"     "hello"
```

```
class(vec3)
```

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

```
# For values in a row, we can also use a colon:
vec3 <- 2:5
print(vec3)
```

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

```
# We can use the c() function to combine pre-saved vectors:
vec4 <- c(vec1,vec3)
print(vec4)
```

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

Here are some other useful shortcuts for creating vectors in R:

```
#use the rep() function to repeat values. Inspect the following to see how it works!
rep(x=0,times=5) #create a vector of 5 zeros
```

```
## [1] 0 0 0 0 0
```

```
rep(x=c(1,2,3),times=5) #create a vector with five repeats of 1,2,3
```

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

```
rep(c(1,2,3),each=2,times=5) #repeat each value in 1,2,3 twice, then repeat that 5 times
```

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

```
#use the seq function to create a sequence of values
seq(from=1, to=5, by=.5) #create a vector with values from 1-5, incementing by .5
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
seq(from=1, to=5, length.out=17) #create a vector with 17 equally spaced values going from 1 to 5
```

```
##  [1] 1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00 3.25 3.50 3.75 4.00 4.25
## [15] 4.50 4.75 5.00
```

Matrices

As shown above, an m by n matrix can be created in R using the matrix() function
Here are some examples

```
A <- matrix(2, nrow=3, ncol=3)
print(A)
```

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

```
B <- matrix(c(1,2,5,3,4,0,2,1,5), nrow=3, ncol=3, byrow=TRUE)
print(B)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    5
## [2,]    3    4    0
## [3,]    2    1    5
```

```
#print the dimensions of a matrix (number of rows followed by number of columns)
dim(A)
```

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

```
#matrix multiplication
A %*% B
```

```
##      [,1] [,2] [,3]
## [1,]   12   14   20
## [2,]   12   14   20
## [3,]   12   14   20
```

```
#element-wise multiplication
A * B
```

```
##      [,1] [,2] [,3]
## [1,]    2    4   10
## [2,]    6    8    0
## [3,]    4    2   10
```

```
#element-wise addition
A+B
```

```
##      [,1] [,2] [,3]
## [1,]    3    4    7
## [2,]    5    6    2
## [3,]    4    3    7
```

```
#transpose of a matrix
t(A)
```

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

```
#inverse of a matrix
solve(B)
```

```
##            [,1]        [,2]        [,3]
## [1,] -0.5714286  0.14285714  0.57142857
## [2,]  0.4285714  0.14285714 -0.42857143
## [3,]  0.1428571 -0.08571429  0.05714286
```

Data Frames

Data frames are generally used to store tabular data. Data frames are composed of same-length vectors; these vectors can be of differing data types. In general, when you read a .csv data file into R, it will be saved as a data frame.
We can create a data frame in R as follows:

```
#create a fake dataset called example_data
example_data <- data.frame(ID_Num = c(1:10),
                           Age = rep(24:28, each=2),
                           State = c(rep("New Jersey", 5), rep("New York", 5)))

#change row names of the data frame (some made up names)
rownames(example_data) <- c("Sarah", "Mike", "Drew", "Eric", "Maria",
                            "Lindsey", "Mark", "Jenny", "Sophie", "Paul")

#print the data frame
example_data
```

```
##         ID_Num Age      State
## Sarah        1  24 New Jersey
## Mike         2  24 New Jersey
## Drew         3  25 New Jersey
## Eric         4  25 New Jersey
## Maria        5  26 New Jersey
## Lindsey      6  26   New York
## Mark         7  27   New York
## Jenny        8  27   New York
## Sophie       9  28   New York
## Paul        10  28   New York
```

The following R code outlines a few ways to inspect data in a data frame.

```
#get dimensions (same as matrices)
dim(example_data)
```

```
## [1] 10  3
```

```
#get summaries of the columns
summary(example_data)
```

```
##      ID_Num           Age            State
##  Min.   : 1.00   Min.   :24   New Jersey:5
##  1st Qu.: 3.25   1st Qu.:25   New York  :5
##  Median : 5.50   Median :26
##  Mean   : 5.50   Mean   :26
##  3rd Qu.: 7.75   3rd Qu.:27
##  Max.   :10.00   Max.   :28
```

```
#access a single column of the data frame using $
example_data$Age
```

```
##  [1] 24 24 25 25 26 26 27 27 28 28
```

```
#inspect row names
rownames(example_data)
```

```
##  [1] "Sarah"   "Mike"    "Drew"    "Eric"    "Maria"   "Lindsey" "Mark"
##  [8] "Jenny"   "Sophie"  "Paul"
```

```r
#inspect column names
colnames(example_data)
```

```
## [1] "ID_Num" "Age"    "State"
```