Introduction to R for Epidemiologists

Jenna Krall, PhD

Thursday, February 5, 2015

Outline

1. Conditional statements
2. If/then statements
3. For loops
4. Apply statements

Conditional statements

Recall that we can use logical operators to make logical vectors

- Logical vectors are vectors with TRUE/FALSE entries

```
vector_1 <- c(3.3, 5.7, 5.8, 3.3, 3.1)
vector_2 <- c(4, 4.5, 3, 5.5, 3.3)

vector_1 < vector_2
```

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

```
vector_1 == vector_1
```

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

```
all.equal(vector_1, vector_1)
```

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

Conditional statements

We can also determine whether some object is in a vector

```
3.3 %in% vector_1
```

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

```
!(3.3 %in% vector_1)
```

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

Conditional statements

Using & and |

```
(2 < 3) & (2 > 3)
```

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

```
(2 < 3) | (2 < 3)
```

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

Conditional statements

If we have vectors, performs element wise

- If the vectors are not the same length, will recycle (and give a warning)

```
c(TRUE, TRUE) & c(TRUE, FALSE)
```

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

```
c(2 < 3, 2 < 4) & c(2 < 3, 2 > 4)
```

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

```
c(2 < c(3, 4)) | c(2 < 3, 2 > 4)
```

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

```
c(2 < 3, 2 < 4) & c(2 < 3, 2 > 4, 4 < 5)
```

```
## Warning in c(2 < 3, 2 < 4) & c(2 < 3, 2 > 4, 4 < 5): longer object length
## is not a multiple of shorter object length
```

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

Conditional statements

Use functions `any` or `all` to summarize `TRUE`/`FALSE` across a Boolean vector

```
any(c(TRUE, TRUE, TRUE, FALSE))
```

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

```
all(c(TRUE, TRUE, TRUE, FALSE))
```

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

Conditional statements

Andand and Oror

- Expect one argument
- &&, || execute sequentially (short-circuit evaluation)

```
FALSE & "does it matter what this is?"
```

```
## [1] "Error in FALSE & \"does it matter what this is?\" : \n  operations are
```

```
FALSE && "does it matter what this is?"
```

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

- Because the first item is FALSE and we used andand, R does not bother to check the second item
- Useful if you only care about second condition if first condition holds

Conditional statements

What about oror?

```
TRUE | "does it matter what this is?"
```

```
## [1] "Error in TRUE | \"does it matter what this is?\" : \n  operations are p
```

```
TRUE || "does it matter what this is?"
```

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

- Because the first item is TRUE and we used oror, R does not bother to check the second item
- Useful if you only care about second condition if first condition does not hold

Control structures

"Controlling" how R performs

- Usually, code in R is executed sequentially (line 1, line 2, etc.)
- What if we only want R to execute code if a certain condition is met?
- What if we want R to do the same thing multiple times?

In SAS

- `if/then` statements
- `do` loops

In R

- `if/then` statements
- `for`, `while` loops

If/then/else

Evaluate code based on whether a condition is true

- If flu activity is under 6000, it is a low flu activity day
- If the species is setosa, make the color blue

Syntax:

```
if ( conditional statement ) do this
```

Example 1:

```
age_emory <- 2015 - 1836
age_emory > 100
```

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

```
if (age_emory > 100) print("Emory is old!")
```

```
## [1] "Emory is old!"
```

If/then/else

What if we wanted to do multiple things?

Example 2: use brackets

```
if (age_emory > 100) {
    print("Emory is old!")
    cat(c("Emory is", age_emory, "years old"))
    }
```

```
## [1] "Emory is old!"
## Emory is 179 years old
```

If/then/else

What if we also want something to print if the statement is false?

Example 3 (nothing happens!):

```
if (age_emory < 100) print("Emory is young!")
```

Example 4: if/then/else

```
if (age_emory < 100) print("Emory is young!") else print("Emory is old!")
```

```
## [1] "Emory is old!"
```

If/then/else

Example 5: Assign result

```
is_emory_young <- if (age_emory < 100) "Emory is young!" else "Emory is old!"
is_emory_young
```

```
## [1] "Emory is old!"
```

Example 6: Alternative function `ifelse`

```
ifelse(age_emory < 100, yes = "Emory is young!", no = "Emory is old!")
```

```
## [1] "Emory is old!"
```

If/then/else

Sometimes we want to have multiple `else`'s

Syntax:

```
if (conditional statement) {
    do thing 1
    do thing 2
} else if (second conditional statement) {
    do thing 3
    do thing 4
} else if () {
    do...
} else if () {
    do...
} else {
    catchall...
    last thing!
}
```

If/then/else

```
if (age_emory < 100) {
    print("Emory is under 100")
} else if (age_emory > 100 & age_emory < 200) {
    print("Emory is over 100 and less than 200")
} else {
    print("Emory is over 200")
}
```

```
## [1] "Emory is over 100 and less than 200"
```

If/then/else

Be careful with `else`!

```
age_emory <- 100
if (age_emory < 100) {
    print("Emory is under 100")
} else if (age_emory > 100 & age_emory < 200) {
    print("Emory is over 100 and less than 200")
} else {
    print("Emory is over 200")
}
```

```
## [1] "Emory is over 200"
```

If/then/else

Be careful with `else`!

```
age_emory <- 100
if (age_emory < 100) {
    print("Emory is under 100")
} else if (age_emory > 100 & age_emory < 200) {
    print("Emory is over 100 and less than 200")
} else if (age_emory > 200 ){
    print("Emory is over 200")
} else {
    print("ERROR: case not defined!")
}
```

```
## [1] "ERROR: case not defined!"
```

If/then/else

```r
rep(c("+", "-"), each = 2)
```

```
## [1] "+" "+" "-" "-"
```

```r
blood_abo <- "A"
blood_rh <- "+"
if (blood_abo == "A" && blood_rh == "+") {
    donors_1 <- paste0(c("O", "A"), rep(c("+", "-"), each = 2))
    cat("Potential blood donors include", donors_1)
} else if (blood_abo == "A" && blood_rh == "-") {
    donors_1 <- paste0(c("O", "A"), "-")
    cat("Potential blood donors include", donors_1)
} else {
    cat("Blood type not documented")
}
```

```
## Potential blood donors include O+ A+ O- A-
```

If/then/else

```
blood_abo <- "B"
blood_rh <- "-"
if (blood_abo == "A" && blood_rh == "+") {
    donors_1 <- paste0(c("O", "A"), rep(c("+", "-"), each = 2))
    cat("Potential blood donors include", donors_1)
} else if (blood_abo == "A" && blood_rh == "-") {
    donors_1 <- paste0(c("O", "A"), "-")
    cat("Potential blood donors include", donors_1)
} else {
    cat("Blood type not documented")
}
```

```
## Blood type not documented
```

```
donors_1
```

```
## [1] "O+" "A+" "O-" "A-"
```

If/then/else

```
flu <- read.csv("googleflu.csv", stringsAsFactors = F)
if (mean(flu$Georgia) > 1000){
    flu_high <- TRUE
} else if (mean(flu$Georgia) <=  1000){
    flu_high <- FALSE
} else {
    print("ERROR: case not defined!")
}
flu_high
```

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

For loops

Do you have to perform the same task for multiple variables?

- Take the mean of age, years of education, income, other variables
- Make scatterplots of CD4 count against a series of risk factors

Syntax:

```
for (how will we loop?) do this
for (how will we loop?) {
    do these things
}
```

For loops

"How will we loop?"

- We need an index (frequently `i`)
- We need to say how that index will change over the loop
- e.g. `i in 1 : 4`

For loops

Example 1:

```
for (i in 1 : 4) {
    print(i)
}
```

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

```
for (i in c(1, 2, 5)) {
    print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 5
```

For loops

Example 2:

```
seq(1, 10, by = 2)
```

```
## [1] 1 3 5 7 9
```

```
for (i in seq(1, 10, by = 2)) {
    print(i)
}
```

```
## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
```

For loops

Example 3:

```
matrix_1
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] 5.04 5.59 1.75 4.52 3.09
## [2,] 4.63 5.78 4.49 6.97 4.61
## [3,] 2.26 2.58 7.20 6.48 6.85
## [4,] 3.80 4.27 6.51 5.18 5.97
```

```
for (i in c(1, 3)) {
    print(matrix_1[i, ])
}
```

```
## [1] 5.04 5.59 1.75 4.52 3.09
## [1] 2.26 2.58 7.20 6.48 6.85
```

For loops

Example 4: column means

```
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
```

```
for (i in 1 : 4) {
    print(head(iris[, i]))
    mean(iris[, i])
}
```

```
## [1] 5.1 4.9 4.7 4.6 5.0 5.4
## [1] 3.5 3.0 3.2 3.1 3.6 3.9
## [1] 1.4 1.4 1.3 1.5 1.4 1.7
## [1] 0.2 0.2 0.2 0.2 0.2 0.4
```

For loops

But where are our means?

Example 5: saving output

```
mean_iris <- vector(length = 4)
mean_iris
```

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

```
for (i in 1 : 4) {
    mean_iris[i] <- mean(iris[, i])
}
mean_iris
```

```
## [1] 5.843333 3.057333 3.758000 1.199333
```

For loops

Can also do the same thing using column names:

```
mean_iris <- vector()
for (i in c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")) {
    mean_iris[i] <- mean(iris[, i])
}
mean_iris
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.843333     3.057333     3.758000     1.199333
```

Final vector is named!

For loops

```
flu_high <- vector(length = length(flu$Georgia))
head(flu_high)
```

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

```
for(i in 1 : length(flu$Georgia)) {
    if (flu$Georgia[i] > 3500) {
        flu_high[i] <- "High"
    } else if (flu$Georgia[i] <= 3500) {
        flu_high[i] <- "Low"
    }
}
table(flu_high, exclude = NULL)
```

```
## flu_high
## High  Low <NA>
##   45  538    0
```

For loops

```
flu <- read.table("googleflumissing.txt", sep = "\t", stringsAsFactors = F,
  header = T)
flu_high <- vector(length = length(flu$Georgia))
head(flu$Georgia)
```

```
## [1]    NA    NA    NA    NA    NA 10380
```

For loops

```
head(flu_high)
```

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

```
for(i in 1 : length(flu$Georgia)) {
    flui <- flu$Georgia[i]
    if (!is.na(flui) && flui > 3500) {
        flu_high[i] <- "High"
    } else if (!is.na(flui) && flui <= 3500) {
        flu_high[i] <- "Low"
    }
}
table(flu_high, exclude = NULL)
```

```
## flu_high
## FALSE  High   Low  <NA>
##   313    14    38     0
```

For loops

```
flu_high <- vector(length = length(flu$Georgia))
for(i in 1 : length(flu$Georgia)) {
    flui <- flu$Georgia[i]
    if (!is.na(flui) && flui > 3500) {
        flu_high[i] <- "High"
    } else if (!is.na(flui) && flui <= 3500) {
        flu_high[i] <- "Low"
    } else if (is.na(flui)) {
        flu_high[i] <- NA
    } else {
        cat("ERROR: case not defined")
    }
}
table(flu_high, exclude = NULL)
```

```
## flu_high
## High  Low <NA>
##   14   38  313
```

Apply

Series of R functions that automate loops

- apply: apply a function to margins of matrix (rows or columns)
  - e.g. Means of variables (columns)
  - e.g. Standard deviation of observations (rows)
- tapply: apply a function separately to groups of an indexing variable
  - e.g. Mean heart rate by sex
  - e.g. Median CD4 count by income group
- lapply: apply a function to each element of a list
- sapply: same as lapply, but tries to simplify

Apply

Syntax for apply:

```
apply( X = mymatrix, MARGIN = 1 or 2, FUN = function of choice)
```

- `MARGIN = 1` corresponds to rows
- `MARGIN = 2` corresponds to columns

```
matrix_1
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] 5.04 5.59 1.75 4.52 3.09
## [2,] 4.63 5.78 4.49 6.97 4.61
## [3,] 2.26 2.58 7.20 6.48 6.85
## [4,] 3.80 4.27 6.51 5.18 5.97
```

Apply

Finding standard deviation of each row:

```
sd_out <- vector(length = 4)
sd_out[1] <- sd(matrix_1[1, ])
sd_out[2] <- sd(matrix_1[2, ])
sd_out[3] <- sd(matrix_1[3, ])
sd_out[4] <- sd(matrix_1[4, ])
sd_out
```

```
## [1] 1.562904 1.072418 2.438725 1.131340
```

Apply

```
sd_out <- vector(length = 4)
for(i in 1: 4) {
    sd_out[i] <- sd(matrix_1[i, ])
}
sd_out
```

```
## [1] 1.562904 1.072418 2.438725 1.131340
```

```
apply(matrix_1, 1, sd)
```

```
## [1] 1.562904 1.072418 2.438725 1.131340
```

Apply

Finding median of each column:

```
matrix_1
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] 5.04 5.59 1.75 4.52 3.09
## [2,] 4.63 5.78 4.49 6.97 4.61
## [3,] 2.26 2.58 7.20 6.48 6.85
## [4,] 3.80 4.27 6.51 5.18 5.97
```

```
apply(matrix_1, 2, median)
```

```
## [1] 4.215 4.930 5.500 5.830 5.290
```

Apply

Recall how we explicitly used for loops for finding column means:

```
mean_iris <- vector()
for (i in c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")) {
    mean_iris[i] <- mean(iris[, i])
}
mean_iris
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.843333     3.057333     3.758000     1.199333
```

One line of code, same result

```
apply(iris[, 1 : 4], 2, mean)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.843333     3.057333     3.758000     1.199333
```

Apply

Can pass other arguments to mean

```
apply(iris[, 1 : 4], 2, mean)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.843333     3.057333     3.758000     1.199333
```

```
apply(iris[, 1 : 4], 2, mean, trim = 0.1, na.rm = T)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.808333     3.043333     3.760000     1.184167
```

Apply

`tapply` allows us to apply a function by a factor

- e.g. What is the mean sepal length by species?
- e.g. What is the mean FEV by age category?

Syntax:

```
tapply(thing to take FUN of, factor, FUN)
```

Apply

Summarizing irises by species

```
levels(iris$Species)
```

```
## [1] "setosa"     "versicolor" "virginica"
```

```
m_seplen <- vector()
m_seplen["setosa"] <- mean(iris$Sepal.Length[iris$Species ==
  "setosa"])
m_seplen["versicolor"] <- mean(iris$Sepal.Length[iris$Species ==
  "versicolor"])
m_seplen["virginica"] <- mean(iris$Sepal.Length[iris$Species ==
  "virginica"])
m_seplen
```

```
##     setosa versicolor  virginica
##      5.006      5.936      6.588
```

Apply

Summarizing irises by species

```
un_species <- unique(iris$Species)
un_species
```

```
## [1] setosa     versicolor virginica
## Levels: setosa versicolor virginica
```

```
m_seplen <- vector()
for(i in un_species) {
  m_seplen[i] <- mean(iris$Sepal.Length[iris$Species == i])
}
m_seplen
```

```
##     setosa versicolor  virginica
##      5.006      5.936      6.588
```

Apply

Summarizing irises by species

```
tapply(iris$Sepal.Length, iris$Species, mean)
```

```
##     setosa versicolor  virginica
##      5.006      5.936      6.588
```

```
tapply(iris$Sepal.Width, iris$Species, sd)
```

```
##     setosa versicolor  virginica
##  0.3790644  0.3137983  0.3224966
```

```
tapply(iris$Sepal.Width, iris$Species, length)
```

```
##     setosa versicolor  virginica
##         50         50         50
```

Apply

The sample function in R selects objects from a vector

```
sample(x = seq(1, 10), size = 1)
```

```
## [1] 6
```

```
sample(x = seq(1, 10), size = 1)
```

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

Apply

lapply and sapply for lists

```
mylist <- list()
for(i in 1 : 4) {
    nrows <- sample(seq(1, 10), size = 1)
    mat1 <- sample(seq(1, 10), size = nrows * 2, replace = T)
    mylist[[i]] <- matrix(mat1, nrow = nrows, ncol = 2)
}
mylist
```

```
## [[1]]
##      [,1] [,2]
## [1,]    7    7
## [2,]    1    5
## [3,]    3    7
## [4,]    3    6
## [5,]    3    2
##
## [[2]]
##      [,1] [,2]
## [1,]    4    9
## [2,]    5    7
## [3,]    1    8
## [4,]    3    4
## [5,]    4    5
## [6,]    9    8
```

Apply

lapply and sapply for lists

```
lapply(mylist, nrow)
```

```
## [[1]]
## [1] 5
##
## [[2]]
## [1] 6
##
## [[3]]
## [1] 9
##
## [[4]]
## [1] 2
```

```
sapply(mylist, nrow)
```

```
## [1] 5 6 9 2
```

Apply

```
lapply(mylist, FUN = apply, 1, mean)
```

```
## [[1]]
## [1] 7.0 3.0 5.0 4.5 2.5
##
## [[2]]
## [1] 6.5 6.0 4.5 3.5 4.5 8.5
##
## [[3]]
## [1] 6.0 9.0 5.5 6.0 2.0 2.5 5.0 6.5 5.5
##
## [[4]]
## [1] 1.5 7.0
```

Apply

```
sapply(mylist, FUN = apply, 1, mean)
```

```
## [[1]]
## [1] 7.0 3.0 5.0 4.5 2.5
##
## [[2]]
## [1] 6.5 6.0 4.5 3.5 4.5 8.5
##
## [[3]]
## [1] 6.0 9.0 5.5 6.0 2.0 2.5 5.0 6.5 5.5
##
## [[4]]
## [1] 1.5 7.0
```

While loops

Mostly useful when interested in convergence

- Iterate until get desired result

Syntax:

```
while ( condition ) {
    do this
}
```

While loops

```
diceroll <- 0
while (diceroll != 3) {
    #Roll a dice!
    diceroll <- sample(seq(1, 6), 1)
    print(diceroll)
}
```

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

```
diceroll
```

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

While loops

Set exit condition

```
diceroll <- 0
iteration <- 0
#Roll until get a 10!
while (diceroll != 10 && iteration < 10) {
    diceroll <- sample(seq(1, 6), 1)
    iteration <- iteration + 1
    cat("Iteration = ", iteration, "\n")
}
```

```
## Iteration =  1
## Iteration =  2
## Iteration =  3
## Iteration =  4
## Iteration =  5
## Iteration =  6
## Iteration =  7
## Iteration =  8
## Iteration =  9
## Iteration =  10
```

```
diceroll
```

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

The R inferno by Patrick Burns

`http://www.burns-stat.com/pages/Tutor/R_inferno.pdf`

WANDERED through

http://www.r-project.org.

To state the good I found there, I'll also say what else I saw.

Circle 8: Believing it does as intended - Section 8.2.17 and and andand