R Basics & Data Cleaning

Kevin da Silva Castanheira

9/18/2018

# R Basics

#### Asking for Help

If you ever get stuck and need more information on any function you can type the following commands into the console. They are equivalent.

help(aggregate)   
?aggregate

#### Variable types

In R as with many other languages your variable can take on several different types: Character: a combination of letters

"Hi everyone"

## [1] "Hi everyone"

Numeric: Number (integer or decimal)

12

## [1] 12

123.232412

## [1] 123.2324

Factor: A set of variables with pre-defined values

factor(c("male", "female", "other","male", "female","male", "female","male", "female"))

## [1] male female other male female male female male female  
## Levels: female male other

factor(c("monday", "tuesday", "wednesday","thursday", "friday","saturday", "sunday"))

## [1] monday tuesday wednesday thursday friday saturday sunday   
## Levels: friday monday saturday sunday thursday tuesday wednesday

factor(c(1,0,1,1,0,1,0,1))

## [1] 1 0 1 1 0 1 0 1  
## Levels: 0 1

#### Vectors, Data-frames, Lists

R has a few basic data structures:

##### Vector: a combiantion of variables with the same type

c(1,2,3,4,5)

## [1] 1 2 3 4 5

5:10

## [1] 5 6 7 8 9 10

c("Dog", "Cat", "Bird", "Fish")

## [1] "Dog" "Cat" "Bird" "Fish"

##### List: a combiantion of variables with no restriction on type

list(c(1,2,3,4,5), c("Dog", "Cat", "Bird", "Fish"), c(TRUE, FALSE))

## [[1]]  
## [1] 1 2 3 4 5  
##   
## [[2]]  
## [1] "Dog" "Cat" "Bird" "Fish"  
##   
## [[3]]  
## [1] TRUE FALSE

##### Data Frame: a matrix organizing data in rows and columns

data.frame("PID"=1:3, "IQ"=c(100, 130, 90), "Age"=c(24, 23, 21))

## PID IQ Age  
## 1 1 100 24  
## 2 2 130 23  
## 3 3 90 21

#### Unless you save your variable by assigning it a “name”, the output will not be retrievable to manipulate

df = data.frame("PID"=1:3, "IQ"=c(100, 130, 90), "Age"=c(24, 23, 21))  
df[1,2] = 40  
df

## PID IQ Age  
## 1 1 40 24  
## 2 2 130 23  
## 3 3 90 21

## Boolean/Logical variables

In most programming languages, there exists boolean/logical variables. They take on the value of TRUE or FALSE. In R as with other languages you can apply operators on them.

1 > 3

## [1] FALSE

3 > 1

## [1] TRUE

3.1 >= 1

## [1] TRUE

3.1 <= 1

## [1] FALSE

You can test the equality of two things by using double equal signs.

3 == 1

## [1] FALSE

"Cat" == "Cat"

## [1] TRUE

"CAT" == "Cat"

## [1] FALSE

You can also test if a value or multiple values is/are present in a vactor:

1 %in% c(2,3,4,5)

## [1] FALSE

c(1,2,3) %in% c(2,3,4,5)

## [1] FALSE TRUE TRUE

##### Boolean Operators

In R, the ! symbolizes the NOT operator. It will return the opposite value of the boolean.

3 != 1

## [1] TRUE

!TRUE

## [1] FALSE

!FALSE

## [1] TRUE

You can combine boolean variables by using the AND/OR operators.

AND requires both variables to be TRUE to return TRUE

(1 > 4) & (1 < 4)

## [1] FALSE

(6 > 4) & (1 < 4)

## [1] TRUE

OR only requires that one variable equal to TRUE

(1 > 4) | (1 < 4)

## [1] TRUE

You can also use if and else statements to control the output of your code. #### If/Else statements

if(1 < 2){  
 print("TRUE")  
}

## [1] "TRUE"

if(3 < 1){  
 print("IMPOSSIBLE")  
} else{  
 print("3 is greater than 1")  
}

## [1] "3 is greater than 1"

if(3 < 1){  
 print("IMPOSSIBLE")  
} else if(3 > 1){  
 print("3 is greater than 1")  
} else{  
 print("WHAT?")  
}

## [1] "3 is greater than 1"

## For loops

If you need to do something multiple time you can use loops to accomplish this.

For loops work by repeating whatever is inside them until a pre-specified condition is met.

for(i in 1:7){  
 print(i)  
}

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

While loops are similar except function without needing to specify exactly how many time to repeat a process.

t = 0  
while(t<0.5){  
 t = runif(1,0,1) # change the value of t by sampling from a uniform distribution  
 # the first argument into the function specifies the number of sample  
 # the second and third arguments specify the min and max of the range respectively  
 print(t)  
}

## [1] 0.4035255  
## [1] 0.229425  
## [1] 0.3119886  
## [1] 0.9798656

# Loading Data

To load data you first need to know a few things about paths.

1. You do not need to provide a full path if the file you’re interested in is in the current working directory

getwd()

## [1] "/Users/kevin/Coding-Club/R\_Basics"

If this is not where you want your working directory to point you can change it

setwd("~/Desktop")

1. If you are on a Mac or a Linux OS paths look different than on a PC MAC/LINUX: “/Users/kevin/Desktop/File” OR “~/Desktop/File” PC: “C:” You can load data into R from a csv using the command read.csv.

NFC <- read.csv("~/Coding-Club/R\_Basics/NFC.csv")

If the first column contains row names, you can specify this to the function in order not read it in as a new column.

df <-read.csv("~/Coding-Club/R\_Basics/data.csv", sep = ',',  
 row.names = 1)

OR if your data is stored in some other format (separated by semi-colons) and does not contain headers, you can modify the arguments into the function read.csv to read the data properly.

Digit\_Symbol <- read.csv("~/Coding-Club/R\_Basics/DigitSymbol.txt", header=FALSE, sep=";")  
  
colnames(Digit\_Symbol) = c("PID", "Score")

#### Looking at your data

Get thr first 10 values

head(NFC, 10)

## PID Gender Age IQ Education Question Value  
## 1 1 F 31 127 BA Q1 5  
## 2 2 M 25 103 high school Q1 5  
## 3 3 M 36 119 Masters Q1 2  
## 4 4 F 34 115 Grad school Q1 1  
## 5 5 M 19 111 Grad Q1 2  
## 6 6 M 27 80 phd Q1 2  
## 7 7 F 25 103 undergrad Q1 4  
## 8 8 <NA> 42 114 high school Q1 2  
## 9 9 M 44 107 M.Sc Q1 3  
## 10 10 M 27 82 bachelors Q1 2

Or the last 10

tail(NFC,10)

## PID Gender Age IQ Education Question Value  
## 1791 91 O 34 105 Bsc Q18 5  
## 1792 92 M 23 126 undergrad Q18 2  
## 1793 93 M 35 91 phd Q18 4  
## 1794 94 F 37 117 BA Q18 1  
## 1795 95 F 28 110 Grad Q18 3  
## 1796 96 M 38 120 high school Q18 1  
## 1797 97 M 19 123 MA Q18 5  
## 1798 98 F 42 86 Masters Q18 5  
## 1799 99 F 21 96 bachelors Q18 3  
## 1800 100 F 44 120 B.Sc. Q18 4

Or a summary of all the columns

summary(NFC)

## PID Gender Age IQ   
## Min. : 1.00 F :756 Min. :18.00 Min. : 80.0   
## 1st Qu.: 25.75 M :846 1st Qu.:27.00 1st Qu.: 92.0   
## Median : 50.50 O : 90 Median :33.00 Median :110.5   
## Mean : 50.50 NA's:108 Mean :32.37 Mean :107.1   
## 3rd Qu.: 75.25 3rd Qu.:39.00 3rd Qu.:120.0   
## Max. :100.00 Max. :45.00 Max. :130.0   
## NA's :90 NA's :36   
## Education Question Value   
## bachelors :198 Q1 : 100 Min. :1.000   
## BS :144 Q10 : 100 1st Qu.:2.000   
## Grad school:144 Q11 : 100 Median :3.000   
## M.Sc :144 Q12 : 100 Mean :2.959   
## MSc :144 Q13 : 100 3rd Qu.:4.000   
## Grad :126 Q14 : 100 Max. :5.000   
## (Other) :900 (Other):1200

Or you can look at a single column by indexing into the data frame

NFC$PID  
NFC[,1] # format is DATAFRAME[ROW, COLUMN]

You can apply various functions to your columns. One useful function is unique which will return all the unique values of a column. I used it here to get a list of all Participant IDs.

unique(NFC$PID)

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17  
## [18] 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34  
## [35] 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51  
## [52] 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68  
## [69] 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85  
## [86] 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

# Data Cleaning

There are many useful function which can allow you to look at and manipulate you data. One useful feature of R is to remove trials or participants.

is.na is a function which checks if a value is NA. Here we are specifying we only want the rows in which Gender is not NA.

NFC = NFC[!is.na(NFC$Gender),]

Or we can exclude participants with an IQ score below a certain threshold

NFC = NFC[NFC$IQ > 80,]

You can index into a data frame to change the values. For example, here I am reverse coding the values of certain questions.

NFC\_reversal = c("Q3", "Q4", "Q5", "Q7", "Q8", "Q9", "Q12", "Q16", "Q17")  
NFC$Value[NFC$Question %in% NFC\_reversal] = 6 - NFC$Value[NFC$Question %in% NFC\_reversal]

#### GREP

You can use regular expressions to find patterns of characters and numbers. In our example, we can try to classify people based on their highest level of education…

unique(NFC$Education)

## [1] BA high school Masters Grad school Grad   
## [6] undergrad M.Sc bachelors BS MSc   
## [11] phd MA Bsc <NA> doctorate   
## [16] B.Sc.   
## 15 Levels: B.Sc. BA bachelors BS Bsc doctorate Grad ... undergrad

Some rules of GREP (see the link for more info: <https://stat.ethz.ch/R-manual/R-devel/library/base/html/regex.html>)

1. It is case sensitive Case so make sure everything is in the same case

NFC$Education = toupper(NFC$Education)  
unique(NFC$Education)

## [1] "BA" "HIGH SCHOOL" "MASTERS" "GRAD SCHOOL" "GRAD"   
## [6] "UNDERGRAD" "M.SC" "BACHELORS" "BS" "MSC"   
## [11] "PHD" "MA" "BSC" NA "DOCTORATE"   
## [16] "B.SC."

1. things preceeded by \* are matched 0 or more times
2. things preceeded by ? will be matched 0 or 1 times
3. ^ matches the start of the string of characters
4. $ matches the start of the end of characters

NFC$Bachelors = grepl("B[.]?S[C]?[.]?[A]?[ACHELOR]?[S]?", NFC$Education) | grepl("UNDERGRAD[UATE]?", NFC$Education)  
  
NFC$HS = NFC$Education == "HIGH SCHOOL"  
  
NFC$Grad = grepl("^GRAD", NFC$Education) | grepl("M[.]?[A]?[S]?[SC]?[.]?", NFC$Education) | grepl("PH[.]?D", NFC$Education)

# Pivot Tables in R

If you’re familiar with Excel, then the function aggregate will be very familiar. The function aggregate allows you to apply a function to groups you define. For example, if we want to calculate a score per participant you can sum all the value for every participant.

The variable on the left of the ~ is the “outcome variable” you are applying the function to. The variables on the right of the ~ are the grouping variables. The final two inputs are the data frame and the function respectively

Scores = aggregate(Value ~ PID, NFC, sum)  
head(Scores, 10)

## PID Value  
## 1 1 57  
## 2 2 65  
## 3 3 48  
## 4 4 70  
## 5 5 53  
## 6 7 66  
## 7 9 54  
## 8 10 60  
## 9 12 46  
## 10 13 47

You can add several columns to this.

aggregate(Age ~ Gender+Bachelors, NFC, mean)

## Gender Bachelors Age  
## 1 F FALSE 31.62500  
## 2 M FALSE 31.30000  
## 3 O FALSE 32.50000  
## 4 F TRUE 33.33333  
## 5 M TRUE 33.11111  
## 6 O TRUE 34.00000

And you can even define your own function! Here I’m defining my own function (SEM).

aggregate(Age ~ Gender+Bachelors, NFC, function(x) sd(x)/sqrt(length(x)))

## Gender Bachelors Age  
## 1 F FALSE 0.3238273  
## 2 M FALSE 0.3636464  
## 3 O FALSE 1.0631471  
## 4 F TRUE 0.5818294  
## 5 M TRUE 0.5937847  
## 6 O TRUE 0.0000000

# Mapping Values Between data-frames

Mapvalues allows you to map data from a different data frame and map those values to another data frame based on certain values. For example, mapping the scores of participants from one data frame of a questionnaire to another data frame.

mapvalues takes in 3 arguments

1. the factor to modify
2. the original values (which match those in 1)
3. the new values you want to map

since mapvalues is a function not availble in base R, it is from the library plyr, you either need to laod the library plyr or let R know the function is from that library by putting its name before the function name.

NFC$DigitSymbol = plyr::mapvalues(NFC$PID, Digit\_Symbol$PID, Digit\_Symbol$Score)

## The following `from` values were not present in `x`: 6, 8, 11, 27, 33, 36, 41, 53, 58, 67, 70, 84, 90