How-To-R: A tutorial series on coding, and data analysis for Biologists

Part 3: Manipulating Tables

OKR

2025-08-13

Loosely adapted from "Introductory R: A Beginner's Guide to Data Visualisation, Statistical Analysis and Programming in R" by Robert Knell

(3.0) Working with tables

(3.05) starting with a clean workspace

It's good practice to periodically declutter your workspace, getting rid of old or irrelevant objects from the session. You can always generate them again using your scripts if needed. If you are starting this tutorial immediately from part 2, then your RStudio session already has some objects and values in the *environment window*, in part 1 we used *rm()* to remove a single object, we can use the following commands to **remove all of them**...

you can skip this if your R environment is already empty.

```
rm(list = ls(all.names = TRUE)) #will clear all objects including hidden objects.
gc() #free up memory and report the memory usage.
```

```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  766789 41.0    1319095 70.5  1319095 70.5
## Vcells 1371045 10.5    8388608 64.0  1973674 15.1
```

(3.1) importing data

Let's import the *.csv* file we saved from the previous tutorial, turn it into an object, and name it *DataStart*. Use *setwd()* to change the working directory to where you saved it.

```
DataStart <- read.csv("TutorialPt2Data.csv", header = TRUE)
```

(3.2) subsetting data

When working with complex datasets, you may want to reshape the data, making subsets to look at specific trends (e.g. comparing males vs females). We will learn two methods of subsetting, using *square brackets* '[]', and using *subset()*.

Bracket subsetting makes use of positionals.

```
vecset0 <- round(rnorm(n=10, mean =20, sd =5), digits = 1) # we create a vector
vecset0 # your values may be different to these, pay attention to the positions of the numbers
```

```
##  [1] 19.8 23.1 12.8  9.6 26.1 28.3 28.5 19.9 16.7 15.3
```

```
vecset0[c(1,4)] # return the values in positions 1 and 4
```

```
## [1] 19.8  9.6
```

```
vecset0[5:8] # return the values in positions 5 through 8
```

```
## [1] 26.1 28.3 28.5 19.9
```

```
vecset0[seq(from=1, to=10, by =2)] # odd numbered positions (e.g. position 1,3,5, etc..)
```

```
## [1] 19.8 12.8 26.1 28.5 16.7
```

```
vecset0[-seq(from=1, to=10, by =2)] #  even numbered positions (e.g. position 2,4,6, etc..)
```

```
## [1] 23.1  9.6 28.3 19.9 15.3
```

```
vecset0[vecset0 <= 20] # return values less than or equal to 20
```

```
## [1] 19.8 12.8  9.6 19.9 16.7 15.3
```

Let's apply this to our *DataStart*. Unlike the 1-dimensional vector *vecset0*, tables can have many rows and columns, when using '[]', we must specify the range of rows and columns (i.e. like coordinates).

```
DataStart[1:3,3] # the first three rows of the third column
```

```
## [1] "Aisyah" "Sarah"  "Peter"
```

If you only need to subset either rows, or columns, then leave the respective part of the 'coordinate' blank.

```
DataStart[1:3,] # the first three rows
```

```
##   X PatientNo FirstName Sex Height_cm Weight_kg  Exercise Married
## 1 1         1    Aisyah   F       156        40    rarely    TRUE
## 2 2         2     Sarah   F       169        50     often   FALSE
## 3 3         3     Peter   M       180        50 sometimes   FALSE
```

```
DataStart[,3] # the third column
```

```
## [1] "Aisyah"  "Sarah"   "Peter"   "Xi Yin"  "Kamarul" "John"
```

Let's subset with conditional statements, for example if we only want to look at Female data. We need to specify the relevant column to apply the condtion. we can use the **dollar/peso sign** '$' followed by the column name to be more specific with our command targets. Try the following commmands.

1. DataStart$Sex #returns the column as a vector
2. DataStart$Sex == "F" #checks the statement : which elements in the vector is 'true' for the condition 'equals to "F" '
3. which(DataStart$Sex == "F") #find the positions for the elements where the statement is true

```
DataStart[which(DataStart$Sex == "F"),] # returns rows where statement '"Sex" == "F"' is true
```

```
##   X PatientNo FirstName Sex Height_cm Weight_kg Exercise Married
## 1 1         1    Aisyah   F       156        40   rarely    TRUE
## 2 2         2     Sarah   F       169        50    often   FALSE
## 4 4         4    Xi Yin   F       175        80    often    TRUE
```

Coding Tip: Copy-paste repeats

Alot of code is repetitive. Instead of typing in full every time, you can copy previous parts of your script and paste when needed. Even if it is just the object name, this saves time.

(3.3) Conditional subsetting

We can combine multiple subset criteria in a single command by adding certain symbols between them. A pipe/vertical bar (|) between statements means *EITHER/OR*, an ampersand (&) between statements means *AND*.

```
# subset for married women
DataStart[which(DataStart$Sex == "F" & DataStart$Married == "TRUE"),]
```

```
##   X PatientNo FirstName Sex Height_cm Weight_kg Exercise Married
## 1 1         1    Aisyah   F       156        40   rarely    TRUE
## 4 4         4    Xi Yin   F       175        80    often    TRUE
```

```
# subset for at least 170cm tall OR lighter than average
DataStart[which(DataStart$Height_cm >= 170|DataStart$Weight_kg < (mean(DataStart$Weight_kg))),]
```

```
##   X PatientNo FirstName Sex Height_cm Weight_kg  Exercise Married
## 1 1         1    Aisyah   F       156        40    rarely    TRUE
## 2 2         2     Sarah   F       169        50     often   FALSE
## 3 3         3     Peter   M       180        50 sometimes   FALSE
## 4 4         4    Xi Yin   F       175        80     often    TRUE
## 6 6         6      John   M       200        80 sometimes   FALSE
```

```
# same as above, but just the third column
DataStart[which(DataStart$Height_cm >= 170|DataStart$Weight_kg < (mean(DataStart$Weight_kg))),3]
```

```
## [1] "Aisyah" "Sarah"  "Peter"  "Xi Yin" "John"
```

Coding Tip: Multi-line code

If you type a long command, R will keep extending to the right of the script line, moving the window as it goes. instead, if your code has commas (","), you can press *ENTER* to move to the next line. This indents your code and keeps everything in the same viewing space. Like below:

```
subset(DataStart,
       Height_cm >= 170)
```

```
##   X PatientNo FirstName Sex Height_cm Weight_kg  Exercise Married
## 3 3         3     Peter   M       180        50 sometimes   FALSE
## 4 4         4    Xi Yin   F       175        80     often    TRUE
## 6 6         6      John   M       200        80 sometimes   FALSE
```

you can subset using the *subset()* function. For dataframes, it can use the *select* argument to specify which columns to return.

```
subset(DataStart,
       DataStart$Height_cm >= 170 | DataStart$Weight_kg < (mean(DataStart$Weight_kg)),
       select = c(FirstName, Height_cm,Weight_kg))
```

```
##   FirstName Height_cm Weight_kg
## 1    Aisyah       156        40
## 2     Sarah       169        50
## 3     Peter       180        50
## 4    Xi Yin       175        80
## 6      John       200        80
```

Use whichever subsetting method you find easier to understand going forward. There are use cases for each, so we are learning both now.

You should start thinking of R as a language, in that there are many ways to say the same thing. There isn't always one 'best way' to phrase a command. Do whatever works for you.

For example, let's try and calculate the Body Mass Index (BMI) for the people in *DataStart*, and insert it as a new column 'BMI'.

BMI is defined as the body mass divided by the square of the body height, expressed as kg/(mˆ2)

If you want to use multiple commands and objects:

```
heights <- DataStart$Height_cm # the heights in cm
heights_m <- heights/100 # heights in m
weights <- DataStart$Weight_kg # the weights in kg
h2 <- (heights_m)^2 # the square of heights in m^2
bmi_longway <- weights/h2 # body mass index
```

Or you can do it in a single command:

```
bmi_short <- (DataStart$Weight_kg)/((DataStart$Height_cm)/100)^2
```

Which way do you prefer? Which way is easier to read, or learn and process mentally? Write your code in a way that you can understand it, even if you found it again 10 years later.

we now have vectors for BMI, we just need to pick one and insert it into *DataStart*. we can use '$'

Currently, *DataStart* does **NOT** have a column called *BMI*, if you tried to find it would return **NULL**

```
DataStart$BMI
```

```
## NULL
```

We can create and fill column 'BMI' it with one of our vectors like this :

```
DataStart$BMI <- bmi_longway
```

Exercises 2: Vectors and subscripts

0. run the command *set.seed(18111992)* and then create the following objects:

i) use *seq()* to create vector 'v1' which contains every 10th number from 20 to 100
ii) use *rnorm()* and *round()* to create object 'v2' with 10 numbers, a mean of 10, and standard deviation of 2; rounded to 1 decimal point. (you may use more than one line of command for this, make sure to run the *set.seed()* command beforehand as suggested above)

1. use a subscript to find the value of the 3rd number in vector v1
2. use a subscript to find the value of numbers in vector v2 that **are not** in the 3rd position
3. add the 1st number in v1 to the 4th number in v2
4. create a vector v3 that consists of the first 4 numbers of v1, followed by the last 4 numbers of v2
5. calculate the sum of all numbers in v3 that is less than 10
6. calculate the mean of all numbers in v2 that are greater or equal to 10.30

(3.39) attaching and detaching objects

We can struggle to read sentences with repeating words, and fewer words also when fewer chances for typos/errors. We have been typing *DataStart* alot recently to access the columns inside in. One way to resolve this is to **attach** the object. Try submitting "BMI" before and after the attaching *DataStart*

```
attach(DataStart) # you may get a warning about masked objects, this is fine
BMI
```

```
## [1] 16.43655 17.50639 15.43210 26.12245 31.11111 20.00000
```

you can now just treat the columns in the attached dataframe as if they are independent objects. This is just an optional tool to use, if you find this confusing or problematic, then you can **detach** the object using *detach()*. I will not be attaching objects in future tutorials to be as clear as possible where objects come from.

(3.4) conditional action, ifelse statements

the ifelse statement is a logic tool in computing to assign values based on the outcome of a test/statement. You provide a statement to check, and two possible actions, **if** the statement is true, the first value is returned, and (**else**), the second action is returned (if the statement is false)

example 1: if the recorded *Sex* is "F", return 'yes', otherwise 'no'

```r
ifelse(Sex == "F", "yes", "no")
```

```
## [1] "yes" "yes" "no"  "yes" "no"  "no"
```

example 2: if the recorded *Sex* is "F", return 'woman', otherwise 'man'

```r
ifelse(Sex == "F", "woman", "man")
```

```
## [1] "woman" "woman" "man"   "woman" "man"   "man"
```

example 3: if exercise frequency is NOT 'rarely', reduce weight by 20, otherwise increase by 10

```r
ifelse(Exercise != "rarely", Weight_kg-20, Weight_kg+10)
```

```
## [1] 50 30 30 60 80 60
```

Since the output is a vector, this can also be added to the dataframe just like *bmi_longway* before. try it with the next exercise

Exercises 3: ifelse statements

Let's add another column to *DataStart* to label patients which are outside of the normal weight. A BMI between 18.5 and 24.9 is considered 'normal weight', below that is 'underweight' and above that is 'overweight'.

1. Add the column 'IsOverW' to *DataStart* which has two possible values: "overW" for patients who are overweight, and "not overW" for patients not overweight.
2. Add the column 'IsNormal' to *DataStart* which has two possible values: "yes" for patients who are normal weight, and "no" for patients outside normal weight. (hint: linguistically, this is: "**If** patient is above **OR** below normal weight, label as 'no', **Else** label as 'yes')

(**HARD EXERCISE**)

3. Add the column 'Category' which has 3 possible values: "underweight", "normal", and "overweight" (hint: you can use a nested ifelse statement; an *ifelse* function inside another *ifelse* function)

Solutions to Exercises 2: Vectors and subscripts i) use *seq()* to create vector 'v1' which contains every 10th number from 20 to 100 ii) use *rnorm()* and *round()* to create object 'v2' with 10 numbers, a mean of 10, and standard deviation of 2; rounded to 1 decimal point. (you may use more than one line of command for this)

```
set.seed(18111992)
v1 <- seq(from = 20, to = 100, by = 10)
v2 <- round(rnorm(n = 10, mean = 10, sd = 2), digits = 2)
```

1. use a subscript to find the value of the 3rd number in vector v1

```
v1[3]
```

```
## [1] 40
```

2. use a subscript to find the value of numbers in vector v2 that **are not** in the 3rd position

```
v2[-3]
```

```
## [1] 10.30  7.22 10.99 11.44  8.67 11.18 10.60  8.39  8.48
```

3. add the 1st number in v1 to the 4th number in v2

```
v1[1] + v2[4]
```

```
## [1] 30.99
```

4. create a vector v3 that consists of the first 4 numbers of v1, followed by the last 4 numbers of v2

```
v3 <- c(v1[1:4],v2[7:10]) # method 1
v3 <- c(head(v1,4), tail(v2,4)) # method 2, look up head() and tail()
```

5. calculate the sum of all numbers in v3 that is less than 10

```
sum(v3[v3<10])
```

```
## [1] 16.87
```

6. calculate the mean of all numbers in v2 that are greater or equal to 10.30

```
mean(v2[v2>=10.3])
```

```
## [1] 11.60833
```

Solutions to Exercises 3: ifelse statements Let's add another column to *DataStart* to label patients which are outside of the normal weight. A BMI between 18.5 and 24.9 is considered 'normal weight', below that is 'underweight' and above that is 'overweight'.

1. Add the column 'IsOverW' to *DataStart* which has two possible values: "overW" for patients who are overweight, and "not overW" for patients not overweight.

```
DataStart$IsOverW<- ifelse(BMI > 24.9 , "overw", "not overw")
```

2. Add the column 'IsNormal' to *DataStart* which has two possible values: "yes" for patients who are normal weight, and "no" for patients outside normal weight. (hint: linguistically, this is: "**If** patient is above **OR** below normal weight, label as 'no', **Else** label as 'yes')

```
DataStart$IsNormal <- ifelse(BMI > 24.9 | BMI < 18.5 , "no", "yes")

# ifelse(BMI >= 18.5 & BMI <= 24.9 , "yes", "no") # alternative phrase using '&'
```

(**HARD EXERCISE**) 3. Add the column 'Category' which has 3 possible values: "underweight", "normal", and "overweight" (hint: you can use a nested ifelse statement; *ifelse* inside another *ifelse*)

```
DataStart$Category <- ifelse(BMI > 24.9 , "overweight",
                             ifelse(BMI < 18.5 , "underweight", "normal"))
```

```
DataStart
```

```
##   X PatientNo FirstName Sex Height_cm Weight_kg  Exercise Married      BMI
## 1 1         1    Aisyah   F       156        40    rarely    TRUE 16.43655
## 2 2         2     Sarah   F       169        50     often   FALSE 17.50639
## 3 3         3     Peter   M       180        50 sometimes   FALSE 15.43210
## 4 4         4    Xi Yin   F       175        80     often    TRUE 26.12245
## 5 5         5   Kamarul   M       150        70    rarely   FALSE 31.11111
## 6 6         6      John   M       200        80 sometimes   FALSE 20.00000
##     IsOverW IsNormal    Category
## 1 not overw       no underweight
## 2 not overw       no underweight
## 3 not overw       no underweight
## 4     overw       no  overweight
## 5     overw       no  overweight
## 6 not overw      yes      normal
```

END of tutorial 3