Introduction to R for Data Analysis in the Health Sciences: Lecture 9

Amy Willis, Biostatistics, UW

05 December, 2019

Today: Advanced programming

... we will discuss cool miscellanea!

- Making nice tables for publication
  - By popular demand!
- Lists
- Scaling up: `apply`, `mutate_all`, `mutate_if`, `mutate_at`...

As always...

```r
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------ tidy
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

By popular demand – making beautiful tables

Let's look at how to make publication-worthy tables with `R` Markdown!

See `9-kable.pdf` and `9-kable.Rmd` under `Module 9` on Canvas

We're going to go through the components of the R Markdown file in detail now.

Making beautiful tables

Let's investigate building a professional looking report on the FEV dataset:

```
fev <- read_csv("datasets/fev.csv")
```

```
## Parsed with column specification:
## cols(
##   seqnbr = col_double(),
##   subjid = col_double(),
##   age = col_double(),
##   fev = col_double(),
##   height = col_double(),
##   sex = col_double(),
##   smoke = col_double()
## )
```

Making beautiful tables

There are two packages that we're going to look at in detail today: `kable` and `kableExtra`:

```
library(knitr)
library(kableExtra)
```

```
##
## Attaching package: 'kableExtra'

## The following object is masked from 'package:dplyr':
##
##     group_rows
```

I have them previously installed on my computer – does anyone remember how to install them for the first time?

Making beautiful tables

Here are two presentations of the same data: Which do you like better?

Making beautiful tables

```
## # A tibble: 8 x 5
## # Groups:   age, sex [6]
##     age   sex smoke `n()` `mean(fev)`
##   <dbl> <dbl> <dbl> <int>       <dbl>
## 1    17     0     0     1        3.5
## 2    17     1     0     5        4.88
## 3    17     1     1     2        3.24
## 4    18     0     0     3        2.95
## 5    18     1     0     1        4.22
## 6    18     1     1     2        4.24
## 7    19     0     1     2        3.43
## 8    19     1     0     1        5.10
```

Making beautiful tables

Age (years)	Sex	Smoker	Sample size	Mean FEV (L)
17	Female	No	1	3.50
17	Male	No	5	4.88
17	Male	Yes	2	3.24
18	Female	No	3	2.95
18	Male	No	1	4.22
18	Male	Yes	2	4.25
19	Female	Yes	2	3.43
19	Male	No	1	5.10

Making beautiful tables

Two steps to making beautiful tables

1. Organize your data into the structure that you want shown
2. Format the data that you structured in Step 1

I encourage you to separate these two steps – don't attempt them concurrently!

Step 1: format the data

Here are some things that we didn't like about the first table:

- Superfluous # signs
- Unnecessary tibble title
- Variable types listed
- No units on variable names
- Not nicely formatted column names
- No borders...

Step 1: format the data

Remember that we can use `rename` and `mutate` to rename the columns:

```
summary_table <- fev %>%
  filter(age > 16) %>%
  rename(`Age (years)` = age) %>%
  mutate(Sex = ifelse(sex == 1, "Male", "Female")) %>%
  mutate(Smoker = ifelse(smoke == 1, "Yes", "No")) %>%
  group_by(`Age (years)`, Sex, Smoker) %>%
  summarise("Sample size" = n(),
            "Mean FEV (L)" = mean(fev) %>% round(2))
```

Notice also that we used `round(2)` to ask us to show us only two decimal places.

Step 1: format the data

The above code gives us the following:

```
summary_table
```

```
## # A tibble: 8 x 5
## # Groups:   Age (years), Sex [6]
##   `Age (years)` Sex    Smoker `Sample size` `Mean FEV (l
##           <dbl> <chr>  <chr>          <int>           <db
## 1            17 Female No                 1             3.
## 2            17 Male   No                 5             4.
## 3            17 Male   Yes                2             3.
## 4            18 Female No                 3             2.
## 5            18 Male   No                 1             4.
## 6            18 Male   Yes                2             4.
## 7            19 Female Yes                2             3.
## 8            19 Male   No                 1             5.
```

Now we can move on to step 2. . . .

Step 2: formatting the table

```
summary_table %>%
  kable
```

Age (years)	Sex	Smoker	Sample size	Mean FEV (L)
17	Female	No	1	3.50
17	Male	No	5	4.88
17	Male	Yes	2	3.24
18	Female	No	3	2.95
18	Male	No	1	4.22
18	Male	Yes	2	4.25
19	Female	Yes	2	3.43
19	Male	No	1	5.10

Step 2: formatting the table

There are a number of different functions in `kable` to change the styling (this doesn't show up well in the slides but shows up in a PDF document)

```
summary_table %>%
  kable %>%
  kable_styling(full_width = F,
                bootstrap_options = c("bordered"))
```

Age (years)	Sex	Smoker	Sample size	Mean FEV (L)
17	Female	No	1	3.50
17	Male	No	5	4.88
17	Male	Yes	2	3.24
18	Female	No	3	2.95
18	Male	No	1	4.22
18	Male	Yes	2	4.25
19	Female	Yes	2	3.43
19	Male	No	1	5.10

Summary: Formatting tables

- First, build the table, e.g.:
  - Rename columns, providing units
  - Reformat the categories
  - Round numeric data to sensible numbers of digits
- Then, use `kable` to format it into a table
- Use `echo = FALSE` in a R Markdown chunk to suppress the code that you used to make the table
  - i.e. You can do your data analysis without all of the steps showing up in the report. See `9-kable.Rmd` and `9-kable.pdf` for an example

Questions?

Lists

There is a type of object in `R` called a *list*

```
mylist <- list() # an empty list
mylist
```

```
## list()
```

```
mylist[["estimate"]] <- 12 ## add in a component called "es
mylist
```

```
## $estimate
## [1] 12
```

Lists

Lists are useful for storing multiple different datatypes

```
mylist3 <- list("estimate" = 12,
                "type" = "parametric")
mylist3
```

```
## $estimate
## [1] 12
##
## $type
## [1] "parametric"
```

Lists can store different datatypes

```r
lm1 <- lm(fev ~ age, data = fev)
mylist4 <- list("estimate" = 12,
                "type" = "semiparametric",
                "lm" = lm1)
mylist4
```

```
## $estimate
## [1] 12
##
## $type
## [1] "semiparametric"
##
## $lm
##
## Call:
## lm(formula = fev ~ age, data = fev)
##
## Coefficients:
## (Intercept)          age
##      0.4316       0.2220
```

Lists can store different datatypes

`list(5, 6)` creates a list with two elements, which are 5 and 6

`list(c(5, 6))` creates a list with one element, which is `c(5, 6)`

```r
list(5, 6)
```

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

```r
list(c(5, 6))
```

```
## [[1]]
## [1] 5 6
```

Lists can store different datatypes

```r
amy <- list(office_num = 657, pets = TRUE,
            pets_names = c("Princess Jaws", "Friendly",
                           "USA", "Regina George"),
            is_cat = c(TRUE, rep(FALSE, 3)))
amy
```

```
## $office_num
## [1] 657
##
## $pets
## [1] TRUE
##
## $pets_names
## [1] "Princess Jaws" "Friendly"      "USA"           "Regina George"
##
## $is_cat
## [1]  TRUE FALSE FALSE FALSE
```

Accessing elements of lists

Double square brackets pull out individual elements. Single square brackets pull out subsets of the list.

```
amy[[3]] # subset third element
```

```
## [1] "Princess Jaws" "Friendly"      "USA"             "Regina George"
```

```
amy[3] # third element -- a list!
```

```
## $pets_names
## [1] "Princess Jaws" "Friendly"      "USA"             "Regina George"
```

Accessing elements of lists

Single square brackets pull out subsets of the list. You can also refer to elements by name

```
amy[2:3] # second and third elements -- a list!
```

```
## $pets
## [1] TRUE
##
## $pets_names
## [1] "Princess Jaws" "Friendly"      "USA"           "Regina George"
```

```
amy$office # can also refer by name
```

```
## [1] 657
```

Lists can store different datatypes

Lists are most useful for storing multiple different data types. In contrast, data frames and tibbles are most useful for storing data that is organized with observations in rows and variables in columns.

I have deemphasised lists in this class because they are not typically needed for data analysis... But they can be useful and I want you to know how to use them when you see them!

Lists in functions

We have seen that functions in `R` can only return one thing...

```
my_chao1 <- function(my_data) {
  c <- sum(my_data > 0)
  f1 <- sum(my_data == 1)
  f2 <- sum(my_data == 2)

  if (f2 == 0) {
    chat <- ifelse(f1 == 0, c, NA)
  } else {
    chat <- c + f1^2 / (2*f2)
  }
  chat
}
```

Lists

... but you can use lists to return multiple things!

```
chao1_list <- function(my_data) {
  c <- sum(my_data > 0)
  f1 <- sum(my_data == 1)
  f2 <- sum(my_data == 2)

  if (f2 == 0) {
    chat <- ifelse(f1 == 0, c, NA)
  } else {
    chat <- c + f1^2 / (2*f2)
  }
  list("chat" = chat,
       "worked" = !is.na(chat))
}
```

Lists

```r
my_counts <- c(5, 1, 1, 7, 20, 2, 1, 1550, 1, 2)
chao1_list(my_counts)
```

```
## $chat
## [1] 14
##
## $worked
## [1] TRUE
```

Questions about lists? (Or anything else?)

Scaling up

A number of you have asked me how to do the same thing to **many** columns – more than you want to type out. Let's check it out!

```
library(tidyverse)
PII <- read_csv(file="datasets/hw8/PII_Jan1.csv")
```

```
## Parsed with column specification:
## cols(
##   ID = col_double(),
##   FirstName = col_character(),
##   LastName = col_character(),
##   Phone = col_character(),
##   Postcode = col_double()
## )
```

Scaling up

Let's start by looking at the data:

```
PII
```

```
## # A tibble: 500 x 5
##       ID FirstName LastName   Phone          Postcode
##    <dbl> <chr>     <chr>      <chr>             <dbl>
##  1   196 Cassidy   Camp       04 4036 8910       2615
##  2   367 JOSHUA    .          02 0510 0633       2612
##  3   481 William   banson     03 6870 2659      29001
##  4    77 <NA>      SHERESTON 03 5667 8971       2617
##  5     4 .         .          08 0183 9097       2904
##  6   482 NONAME    <NA>       03 6870 2659       2900
##  7   444 NONAME    <NA>       03 9886 1216       2905
##  8   299 <NA>      Dukic      04 0167 8131       2605
##  9   483 .         NONAME     03 6870 2659       2900
## 10   466 Ruben     <NA>       04 9124 1348       2904
## # ... with 490 more rows
```

Scaling up

One way to do the last homework was to create a function like the following, and then use it to `mutate` the columns

```
name_standardizer <- function(column) {
    column[column %in% c("NONAME", "", ".")] <- NA
    column <- as.character(column)
    column <- tolower(column)
    column
}
```

Scaling up

You can use this function in conjunction with `mutate` as follows:

```
PII %>%
  mutate(FirstName = name_standardizer(FirstName),
         LastName = name_standardizer(LastName))
```

```
## # A tibble: 500 x 5
##        ID FirstName LastName  Phone            Postcode
##     <dbl> <chr>     <chr>     <chr>               <dbl>
##  1    196 cassidy   camp      04 4036 8910         2615
##  2    367 joshua    <NA>      02 0510 0633         2612
##  3    481 william   banson    03 6870 2659        29001
##  4     77 <NA>      shereston 03 5667 8971         2617
##  5      4 <NA>      <NA>      08 0183 9097         2904
##  6    482 <NA>      <NA>      03 6870 2659         2900
##  7    444 <NA>      <NA>      03 9886 1216         2905
##  8    299 <NA>      dukic     04 0167 8131         2605
##  9    483 <NA>      <NA>      03 6870 2659         2900
```

Scaling up

This was easy because they were only two columns – how do we do multiple columns?

One way is to “apply” the same function to many columns.

```
apply(PII[, 2:3], 2, name_standardizer)
```

```
##       FirstName   LastName
##  [1,] "cassidy"   "camp"
##  [2,] "joshua"    NA
##  [3,] "william"   "banson"
##  [4,] NA          "shereston"
##  [5,] NA          NA
##  [6,] NA          NA
##  [7,] NA          NA
##  [8,] NA          "dukic"
##  [9,] NA          NA
## [10,] "ruben"     NA
## [11,] "charlotte" NA
```

Scaling up

How does apply work? `apply(df, index, func)`

- `df`: A dataset (data frame, tibble or matrix)
- `index`: The direction you want to apply across
  - 1: apply horizontally (to the rows)
  - 2: apply vertically (to the columns)
- `func`: The function that you want to apply to the rows or columns

So `apply(PII[, 2:3], 2, name_standardizer)` says apply `name_standardizer` to the 2nd and 3rd columns of `PII`

Scaling up

To use this to overwrite columns, create a copy of the dataset and then overwrite the columns that you want to change

```
PII_v2 <- PII
PII_v2[, 2:3] <- apply(PII_v2[, 2:3], 2, name_standardizer)
PII_v2
```

```
## # A tibble: 500 x 5
##        ID FirstName LastName   Phone          Postcode
##     <dbl> <chr>     <chr>      <chr>             <dbl>
##  1    196 cassidy   camp       04 4036 8910       2615
##  2    367 joshua    <NA>       02 0510 0633       2612
##  3    481 william   banson     03 6870 2659      29001
##  4     77 <NA>      shereston  03 5667 8971       2617
##  5      4 <NA>      <NA>       08 0183 9097       2904
##  6    482 <NA>      <NA>       03 6870 2659       2900
##  7    444 <NA>      <NA>       03 9886 1216       2905
##  8    299 <NA>      dukic      04 0167 8131       2605
```

Scaling up

Here are three other ways that use the `tidyverse`:

- `mutate_all`: mutate all the columns
- `mutate_if`: mutate columns that satisfy a simple condition
- `mutate_at`: mutate columns that satisfy a (perhaps more complex) condition

Scaling up: mutate_all

Apply the function to all of the columns:

```
PII[, 2:3] %>%
  mutate_all(.funs=name_standardizer)
```

```
## # A tibble: 500 x 2
##    FirstName LastName
##    <chr>     <chr>
##  1 cassidy   camp
##  2 joshua    <NA>
##  3 william   banson
##  4 <NA>      shereston
##  5 <NA>      <NA>
##  6 <NA>      <NA>
##  7 <NA>      <NA>
##  8 <NA>      dukic
##  9 <NA>      <NA>
## 10 ruben     <NA>
```

Scaling up: mutate_if

Apply the function to all of the columns that are characters:

```
PII %>%
  mutate_if(is.character, name_standardizer)
```

```
## # A tibble: 500 x 5
##       ID FirstName LastName   Phone          Postcode
##    <dbl> <chr>     <chr>      <chr>             <dbl>
##  1   196 cassidy   camp       04 4036 8910       2615
##  2   367 joshua    <NA>       02 0510 0633       2612
##  3   481 william   banson     03 6870 2659      29001
##  4    77 <NA>      shereston  03 5667 8971       2617
##  5     4 <NA>      <NA>       08 0183 9097       2904
##  6   482 <NA>      <NA>       03 6870 2659       2900
##  7   444 <NA>      <NA>       03 9886 1216       2905
##  8   299 <NA>      dukic      04 0167 8131       2605
##  9   483 <NA>      <NA>       03 6870 2659       2900
## 10   466 ruben     <NA>       04 9124 1348       2904
```

Scaling up: `mutate_at`

Apply the function to all of the columns that contain the phrase "Name" in their column name (e.g. `FirstName`, `LastName`, but not `Postcode`)

```
PII %>%
  mutate_at(vars(contains("Name")), name_standardizer)
```

```
## # A tibble: 500 x 5
##        ID FirstName LastName   Phone           Postcode
##     <dbl> <chr>     <chr>      <chr>              <dbl>
##  1    196 cassidy   camp       04 4036 8910        2615
##  2    367 joshua    <NA>       02 0510 0633        2612
##  3    481 william   banson     03 6870 2659       29001
##  4     77 <NA>      shereston  03 5667 8971        2617
##  5      4 <NA>      <NA>       08 0183 9097        2904
##  6    482 <NA>      <NA>       03 6870 2659        2900
##  7    444 <NA>      <NA>       03 9886 1216        2905
##  8    299 <NA>      dukic      04 0167 8131        2605
```

Wrap up: Questions

Are there any questions/comments that folks would like to share with the class?

Wrap up: Congratulations

We covered an enormous amount of material this quarter! This wouldn't have been possible without your *hard work* and *commitment*! Congratulations!

The objective of this course was to give you the **confidence** and **skills** to perform data analysis in `R` via

- Learning the syntax of `R`
- Gaining experience loading, transforming, summarising and plotting data
- Developing a repertoire of strategies to troubleshoot and expand your understanding of `R`
  - Both *within* and *beyond* the scope of this class

I hope you feel that we achieved our objective!

Wrap up

- Our materials will continue to be publicly available at `https://github.com/adw96/biost509`
  - Feel free to distribute this link to your friends and colleagues!
- Keep in touch by e-mail (`adwillis@uw.edu`) or Twitter (@AmyDWillis)
  - I always enjoy hearing from my students about their `R` and statistics adventures
- I look forward to seeing around!

Wrap up: Into the Unknown

Now you have skills and strategies to go forth and analyze your data in R!

Best wishes from the BIOST 509 team! It has been a pleasure to work with you throughout this quarter!

The plan

- An optional exercise is available to give you practice with some concepts from today
  - It won't be graded, but are here to help you with questions
  - I will post solutions at 3:30pm
- Please please please please please submit your course evaluation
  - I really do take your feedback on board!
  - `https://uw.iasystem.org/survey/214294`

Thank you again for your effort and achievement in this course! I have loved working with you all so much!