Lesson 3: Control flow with `if` and `ifelse`

Ari Anisfeld

8/31/2020

We expect you to watch the `class 3` material, here prior to lab. Download the data before lab.

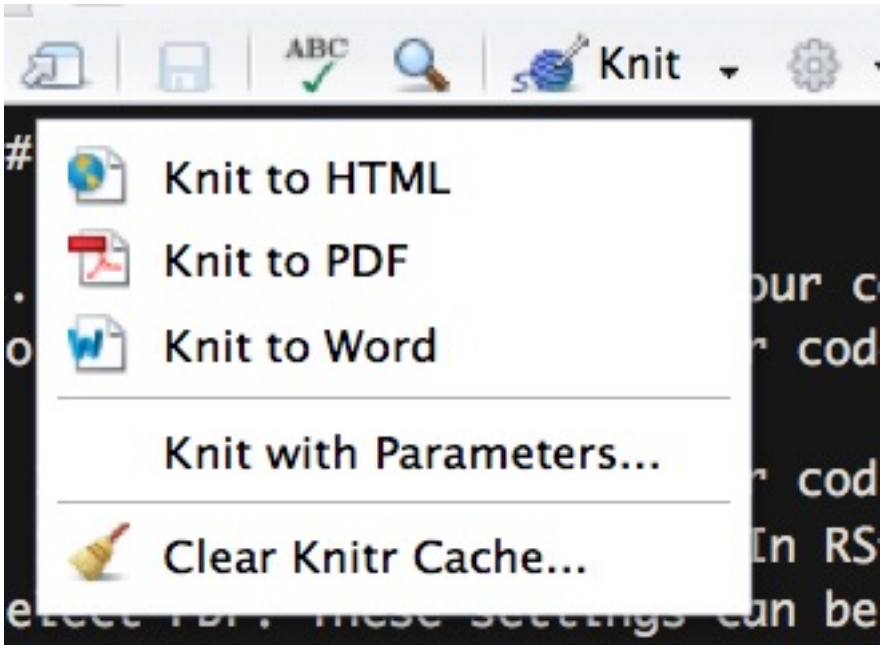
Data:

For this lab, we will use data from Opportunity Insights: https://opportunityinsights.org/data/

1. Download the Stata file and Readme for "College Level Characteristics from the IPEDS Database and the College Scorecard".

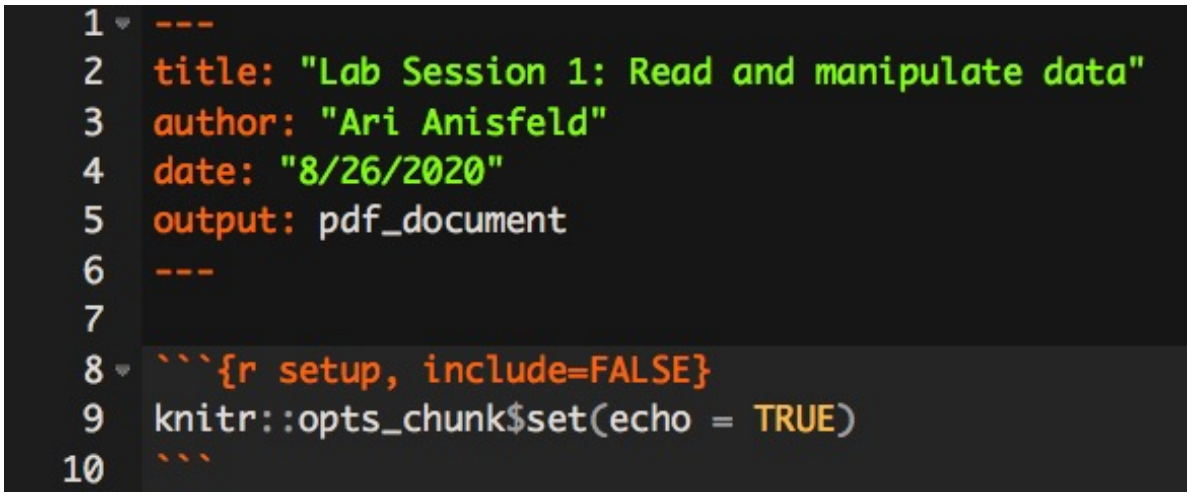
Intro to `Rmd`s

1. Create an `Rmd` file. (In RStudio's menu `File > New File > R Markdown`). Name the document and select PDF. These settings can be changed later.
2. Save the `Rmd` in your coding lab folder as `lab_3.Rmd`.
3. New `Rmd`s comes with some example code. Read the document and then knit to pdf by clicking the knit button. If `pdf` doesn't work, try `html`.



Pay attention to the syntax and ask your group or TA about anything you don't understand. `Rmd`s start with meta information which provides instructions to `knitr` on how to knit. After that, there's a

normal code chunk which runs, but you won't see because they of the `include=FALSE` bit at start of the code chunk.



Keep the part shown in the image above and erase the rest of the code and text in the document. This is how we start our own `Rmd` code.

4. Make a new code chunk by pressing Ctrl + Alt + I (Cmd + Option + I on macOS).[1] Then, add code to load the `tidyverse` in that chunk.

5. `Rmd`s keep track of their own working directories. Try `getwd()` in your `Rmd`[2] and then run it again in the console. You should notice that `R` in the console uses your default working directory. On the other hand, the `lab_3.Rmd` knows which folder it is in and uses that folder as the working directory!

6. If you didn't already, put the data you downloaded in your preferred data location.

If you followed the set-up from above, you should be able to read the data in your `Rmd` with no error.

We provide three options dependening on how you have structured your folders. Option 1 is the easiest for a first time user. Keep the data in the same folder as the Rmd. Option 2 and 3 are what you're more likely to see in a professional environment.[3]

```
library(haven)
# Option 1:
# data is in the same folder as Rmd
mrc_data <- read_dta("mrc_table10.dta")

# Option 2:
# data is in a data folder inside the folder with the Rmd
mrc_data <- read_dta("data/mrc_table10.dta")

# Option 3:
# data is in a data folder that is at the same level
# as the folder with the Rmd.
mrc_data <- read_dta("../data/mrc_table10.dta")
```

Once you got the data to load with code in the Rmd. Try the same code in the console. It will throw an error. Briefly discuss why with your group.

[1] You can also type three tick marks with `{r}` and then another three tick marks.

[2] By which, I mean in a code chunk in your `Rmd`.

[3] We did a poll among the TAs about their preferred directory structure. We were split between Option 2, Option 3, and a feeling like the choice was "Too personal" to make a cohort wide

Warm-up: Conditional statements and `ifelse`:

1. Without running the code, predict what the output will be. Then, see if you were right by running the code in the console.

True or False

a. `TRUE | FALSE`
b. `TRUE | (FALSE & FALSE)`
c. `TRUE | (10 < 4)`
d. `TRUE | (10 < 4 & )`
e. `TRUE | (4 > pi & 3 < pi & exp(1) >= 3 & 1e6 < 2^30)`
f. `4 > 2 | 2 > 4`
g. What rule do these problems demonstrate?

True and False

a. `TRUE & FALSE`
b. `TRUE & (FALSE & FALSE)`
c. `TRUE & (10 < 4)`
d. `TRUE & (10 < 4 & )`
e. `TRUE & (4 > pi & 3 < pi & exp(1) >= 3 & 1e6 < 2^30)`
f. `4 > 2 & 2 > 4`
g. What rule do these problems demonstrate?

True and NA

h. There are a few times when `NA` are not contangious. Run the code and think about how this relates to your findings above.

```
TRUE & NA
FALSE & NA
TRUE | NA
FALSE | NA
```

1. Without running the code, predict what the output will be. Then, see if you were right by running the code in the console.

```
ifelse(TRUE, "yes", "no")
ifelse(FALSE, "yes", "no")
ifelse(c(TRUE, FALSE, TRUE, FALSE), "yes", "no")
ifelse(c(TRUE & FALSE,
         FALSE | TRUE,
         TRUE | TRUE,
         FALSE & FALSE),
       "yes", "no")
ifelse(c(NA, TRUE, FALSE), "yes", "no")
ifelse(c(NA, NA, TRUE, FALSE), "yes", "no")
```

Common uses of `ifelse`

1. Run the following code and you will see the distinct `tier_names` available in the dataset.

```
mrc_data %>% distinct(tier_name)
```

   a. `ifelse` can be used to adjust entries in the `tier_name` column. Change "Two-year (public and private not-for-profit)" to "Two-year (public and private)".[4]

[4]Hint: In the first position, put a condition testing if `tier_name` matches the string. If it does, we replace the string with

```
# Fill in the ... with the appropriate code
mrc_data %>%
  mutate(tier_name = ifelse( ... , ..., tier_name))
```

   a. `ifelse` is often used to collapse tiers. Redefine `tier_name` so that "Nonselective four-year public" and "Nonselective four-year private not-for-profit" are grouped together as "Nonselective four-year (public and private)".[5]

2. As you can see, there are 1466 colleges missing average SAT scores. Sometimes we want to replace `NA`s with a value. For example, linear regressions will drop any row with `NA`s, and we might not want that.[6]

```
mrc_data %>%
  summarise(missing_sat_2013 = sum(is.na(sat_avg_2013)))
```

```
## # A tibble: 1 x 1
##   missing_sat_2013
##              <int>
## 1             1466
```

   To avoid dropping rows, sometimes people replace the `NA` with the mean and add a new column that is an indicator of missingness. Using `mutate()` and `ifelse()`, fill `NA` in `sat_avg_2013` with the average SAT score of the other colleges and create a column called `missing_sat_avg_2013` that is 1 if `NA` and 0 otherwise.[7]

Here's a small example of what we expect. Try reproducing this example and then applying your code to `mrc_data`.

```
before <- tibble(fake_data = c(1, 2, NA))
before
```

```
## # A tibble: 3 x 1
##   fake_data
##       <dbl>
## 1         1
## 2         2
## 3        NA
```

```
after
```

```
## # A tibble: 3 x 2
##   fake_data missing_fake_data
##       <dbl>             <dbl>
## 1       1                   0
## 2       2                   0
## 3       1.5                 1
```

Extension: College choice:

This part is admittedly silly! Imagine the situation: It's 2014 and a group of high school friends want to go to college together. They need to find a college that meets all their preferences. Your job is to find the perfect college.

"Two-year (public and private)", otherwise keep the same data.

[5] Hint: The code will be very similar to the previous problem.

[6] I believe you'll discuss missing data problems in stats I.

[7] Hint: First, make the indicator column. Hint 2: When replacing `NA` in the example, I used the following code to find the mean `mean(fake_data, na.rm = TRUE)`.

Name	SAT Score	Preferences
A-plus Abdul	1430	Either ivy plus tier or a flagship school
Snooty Stephen	1450	not a public school
Nourishing Nancy	1590	school in the midwest so she can be near her grandma
Radical Rei	1490	strong social studies (as measured by the percentage of students majoring in social studies $>$ 30 percent)
Cost-conscious Casey	1600	wants a public school in CA or a school where students from homes in the bottom 20th percentile of incomes pay less than 10000 per year

Here are the rules. They want to go to school where their test scores are within 100 points of the school average SAT score. To match their preferences, use the most recent data. You will need a few tools.

1. First, in order to understand what a column contains you can use `distinct()`[8]. For example, say you are trying to figure out how to identify "ivy plus" schools (or what that specifically means). Using `names()` you see their is a columns called `tier_name`.

```
mrc_data %>% distinct(tier_name)
```

```
## # A tibble: 12 x 1
##    tier_name
##    <chr>
##  1 Two-year for-profit
##  2 Selective private
##  3 Nonselective four-year public
##  4 Four-year for-profit
##  5 Selective public
##  6 Two-year (public and private not-for-profit)
##  7 Nonselective four-year private not-for-profit
##  8 Highly selective private
##  9 Less than two-year schools of any type
## 10 Other elite schools (public and private)
## 11 Highly selective public
## 12 Ivy Plus
```

   We see there are 12 tiers and one is "Ivy Plus"! Note the capitalization.

2. Second, we're going to have to find schools that match ranges of SAT scores. Welcome `between()`.[9]

```
mrc_data %>% filter(1330 <= sat_avg_2013, sat_avg_2013 <= 1530)
mrc_data %>% filter(between(sat_avg_2013, 1330, 1530))
```

   a. Figure out whether `between()` use `<` or `<=`?

3. The final thing is a concept. You're probably about to write code that looks like the following pseudo code.[10]

```
# This is pseudo code
mrc_data %>%
  mutate(abdul_choices = ifelse(CONDITIONS, yes, no),
         stephens_choices = ifelse(CONDITIONS, yes, no),
```

[8] from `dplyr`. The codebook is also useful.
[9] from `dplyr`
[10] pseudo code is a term for fake code that captures the logic of some coding idea without being actual code.

```
        ...) %>%
  filter(abdul_choices == yes, stephens_choices == yes, ...)
```

We can avoid the extra `== yes` by making `abdul_choices` a logical vector. In other words, write code like so:

```
# This is pseudo code
mrc_data %>%
  mutate(abdul_choices = ifelse(CONDITIONS, TRUE, FALSE),
         stephens_choices = ifelse(CONDITIONS, TRUE, FALSE),
         ...) %>%
  filter(abdul_choices, stephens_choices, ...)
```

a. Test out the concept with a simple example.[11]

4. Now you're ready to find the college for the five friends.

```
# fill in the  ... with appropriate code

# We'll give this a name so we can use it later.
bff_super_awesome_college_list <-
mrc %>%
  mutate(abdul_choices = ifelse(between(sat_avg_2013, 1330, 1530) &
                                  (tier_name == "Ivy Plus" | ... ), TRUE, FALSE),
         sam_choices = ifelse(..., ..., ...),
         nancy_choices = ifelse(..., ..., ...),
         rei_choices = ifelse(..., ..., ...),
         casey_choices = ifelse(..., ..., ...)
         )


bff_super_awesome_college_list %>%
  filter(abdul_choices, sam_choices, nancy_choices, rei_choices, cary_choices)
```

a. What school(s) are acceptable to all five
b. How many school(s) are available to any of the five. Adjust `filter` statement slightly.[12]

1. The five friends have `NA` in their choice sets. Do the the school list change if we replace all the `NA`s with `TRUE`? Without coding, argue why the list will not change if we replace the `NA`s with `FALSE`.

2. **Challenge** Create a "Five friends college ranking". A college is ranked 1 if it is acceptable to all 5 friends. 2 if it is acceptable to any 4 friends and so on.[13] Colleges that are not acceptable to any friend should be marked "Unranked".

[11] For example, try it with Abdul's only condition being Ivy Plus.
[12] Hint: Think about the warm-up you did for this lab
[13] 3 if it is acceptable to 3 friends. 4 if acceptable to 2 friends and 5 if acceptable to 1 friend