Tidyverse and documentation

Francis L. Yuen

2019-12-09

Learning goals:

1. Writing readable scripts using the pipe function
2. Properly document your script
3. Using conditional statements
4. Practice analyzing and visualizing data

Writing readable scripts

In the last workshop, we wrote a series of codes that analyzed a mock dataset. Open up that R script. It should look something like this:

Remember to check that your raw data is stored in the right place so that `here()` can find it. Adjust the directory to help `here()` locate your raw data if necessary.

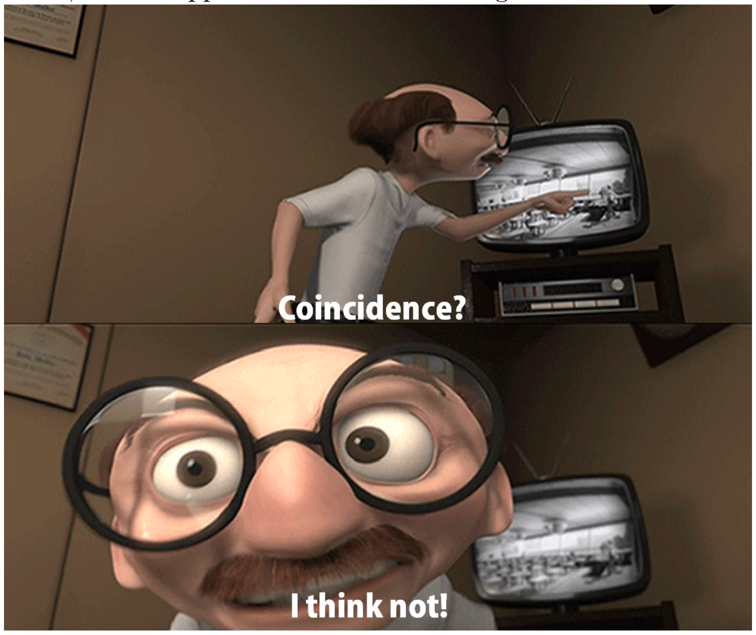
```
library(tidyverse)
library(janitor)
library(here)
data <- read_csv(here("data", "sample_data_1.csv"))
data <- clean_names(data)
data_tidy <- pivot_longer(data = data, cols = c(trial1:trial10),
    names_to = "trial_number", names_prefix = "trial",
    values_to = "looking_time")
data_tidy <- mutate(.data = data_tidy, corrected_looking_time = looking_time +
    3.4, log_looking_time = log(corrected_looking_time))
data_tidy <- filter(.data = data_tidy, corrected_looking_time <=
    30)
data_tidy <- group_by(.data = data_tidy, hypothesis)
lt_by_choice <- summarize(.data = data_tidy, mean = mean(log_looking_time),
    sd = sd(log_looking_time), n = length(log_looking_time),
    SEM = sd/sqrt(n))
```

Recall that one of the problems here is that you have to execute several separate lines, each one updating what 'data_tidy' is before finally arriving at your results. We can shorten the code by using a function called the pipe, which is written like this: `%>%`

What the pipe does is **take the object that comes before the pipe, and pass it on as the first argument of the function that comes after the pipe**. To illustrate, let's use the pipe to rewrite the first part of our code and store it as `data_tidy_pipe` so we can compare the results:

```
data_tidy_pipe <- data %>%
pivot_longer(cols = c(trial1:trial10), names_to = "trial_number",
    names_prefix = "trial", values_to = "looking_time")
```

Run this first line of your old code and compare `data_tidy_pipe` with `data_tidy`, they should be the same. What the pipe did was take the object 'data' and pass it as the first argument of the `pivot_longer()` function, which happens to be the `data =` argument.



The cool thing about tidyverse is that **all the functions in this package takes data as the first argument**, which is super useful for '%>%' because we can 'chain' several functions together as long as they all take 'data =' as the first argument. This is what that might look like if we chain the `mutate()` function next:

```
data_tidy_pipe <- data %>%
pivot_longer(cols = c(trial1:trial10), names_to = "trial_number",
    names_prefix = "trial", values_to = "looking_time") %>%

mutate(corrected_looking_time = looking_time +
    3.4, log_looking_time = log(corrected_looking_time))
```

Interpreting code: basically each time you see a pipe, you can view it as the word 'then'. In this example, you are essentially saying: first, call the object 'data', **then** pivot_longer() this data, **then** mutate() these two columns.

Exercise 1

Finish writing the piped version of this script. Instead of storing it as an object called `lt_by_choice`, you can keep the final result as `data_tidy_pipe` since you can even chain `summarize()` as part of the pipe.

Documentation

Now that our code looks neater, it's time to document and keep track of what is happening either in each section or each line. The '#' symbol tells R to "ignore" everything that comes after, which means you can utilize that to write notes for yourself/future self/other people. Here's what that might look like:

```
# Comparing log of looking time between babies
# who chose with vs against
comparison_by_choice <- data %>%
pivot_longer(cols = c(trial1:trial10), names_to = "trial_number",
    names_prefix = "trial", values_to = "looking_time") %>%

mutate(corrected_looking_time = looking_time +
    3.4, log_looking_time = log(corrected_looking_time)) %>%

filter(corrected_looking_time <= 30) %>%
group_by(hypothesis) %>%
summarize(mean = mean(log_looking_time), sd = sd(log_looking_time),
    n = length(log_looking_time), SEM = sd/sqrt(n))
```

You can make more elaborate documentation as long as it makes sense:

```
# Comparing log of looking time between babies who chose with vs against
comparison_by_choice <- data %>%
  pivot_longer(cols = c(trial1:trial10),                      # convert data to tidy form
               names_to = "trial_number",
               names_prefix = "trial",
               values_to = "looking_time") %>%
  mutate(corrected_looking_time = looking_time + 3.4,         # corrected looking time
         log_looking_time = log(corrected_looking_time)) %>%   # log transform looking time
  filter(corrected_looking_time <= 30.00) %>%                 # exclude trials where looking time > 30
  group_by(hypothesis) %>%                                    # group by choice: with vs against
  summarize(mean = mean(log_looking_time),                    # calculate mean, sd, n, and SEM
            sd = sd(log_looking_time),
            n = length(log_looking_time),
            SEM = sd/sqrt(n))
```

How much you document your steps is entirely up to you. Documentation takes a lot of time, but it is an important step to increase reproducibility and reduce error rates since it allows someone else who's looking at your script for the first time to easily understand your intentions and catch mistakes. This is especially important when you are collaborating with others and working on the same script.

Using Conditional Statements

Now that we know the basics, we can start doing slightly more complicated things. First, let's quickly review the tools we learned about data management:

Exercise 2
Download another **sample dataset**. Using what you know from the previous workshop, create a new Rproject called “Bullies_test”, create an Rscript for analysis, and read in the dataset. Inspect the dataset: Is it tidy? Are the column names clean? Clean up the column names and tidy the data.

Using ifelse()

In this hypothetical study, we expect the baby to pick the nice target (NT). Notice that three columns are missing. We can use `mutate()` to add values to an existing column instead of making a new column. If the baby picked NT, we know that the hypothesis is “With”; if they picked MT, it will be “Against”. We can utilize a simple `ifelse()` function to perform this `mutate()`.

Exercise 3
Write a line of code that fills in the `hypothesis_with_or_against` column appropriately by combining `mutate()` with `ifelse()`.

Hint 1: Remember how you can check the arguments of a function you have never used before?

Hint 2: logic tests use the double equal sign (==) instead of a single one.

Using case_when()

`ifelse()` is nice when you have very simple conditions like the one above. However, things can get more complicated, and it’s not always a good idea to nest several `ifelse()` statements within each other. To illustrate, let’s fill in the `choice_color` column. You can find out what color the mean target is by looking at the `mt_in_show` column. Here is a pretty straight forward way to think about this problem:

If baby chose MT -> choice_color = mt_in_show.

If baby chose NT -> choice_color = opposite of mt_in_show

Looks simple enough, right? But here’s the problem: R doesn’t know what the opposite of mt_in_show is, so the decision tree is actually a bit more complicated:

if baby chose NT -> if mt_in_show is yellow, choice_color = blue, otherwise it’s yellow (because mt_in_show is blue)

What does this look like as an `ifelse()`?

```r
bullies <- bullies %>%
  mutate(choice_color =
      ifelse(baby_choice_nt_or_mt == "MT", mt_in_show, # if baby chose MT, color = mt_in_show
      ifelse(mt_in_show == "yellow", "blue",     # otherwise, if mt_in_show = yellow, color = blue
      "yellow")))                                # otherwise, it's yellow
```

That doesn't look awful, but it's certainly not straight forward and immediately readable. Imagine having to nest another layer! This is where a different but similar function comes in handy: `case_when()`. This function takes a similar format, except it allws you to set one condition per line. The format is:

case_when(condition ~ output)

Here's how the same process looks like:

```
bullies <- bullies %>%
  mutate(choice_color =
           case_when(baby_choice_nt_or_mt == "MT" ~ mt_in_show, # if choice is MT, color = mt_in_show
                     mt_in_show == "yellow" ~ "blue",  # if mt is yellow, choice color = blue
                     mt_in_show == "blue" ~ "yellow")) # if mt is blue, choice color = yellow
```

Here, although you have more text in the code, it's much more readable because each line represents a condition, and at the end of each ~ is the output. You don't have to keep track of which argument you're on to figure out what is the condition and what is the output.

Notice that we don't need to specify baby_choice_nt_or_mt because case_when() conditions are mutually exclusive: after all the cases of choice = MT is considered, the following cases will assume that they should only be applied to the remaining cases. In our case, all other cases are NT since those are the only two options.

Exercise 4

There is one empty column remaining in the dataset. Using either `ifelse()` or `case_when()`, whichever one is more appropriate, mutate the last column.

Bonus: combine the generation of all three columns thus far into one `mutate()`.

Practice analyzing and visualizing data

The main hypothesis here is if babies significantly prefer the nice target over the mean target. However, notice that in the tidy format, you will double count the participants (in this case, octocount since each participant has 8 rows). How do we get around this?

```
hypothesis_sum <- bullies %>%
group_by(hypothesis_with_or_against) %>%
summarize(n = n()/8)
```

R has a lot of built in statistical tools already. Use `binom.test()` to see if our results are significant.

Bonus: Instead of manually entering the counts, think of a way to make it reproducible.

```
hypothesis_binom <- binom.test(hypothesis_sum$n[2],
    hypothesis_sum$n[1] + hypothesis_sum$n[2])
hypothesis_binom

##
##  Exact binomial test
##
```

```
## data:  hypothesis_sum$n[2] and hypothesis_sum$n[1] + hypothesis_sum$n[2]
## number of successes = 80, number of
## trials = 128, p-value = 0.005925
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.5351109 0.7089650
## sample estimates:
## probability of success
##                  0.625
```

Some questions to think about:

- Infant choice is often expressed as percentage or proportion of babies choosing whichever one is correct (in this case, the nice target). How can we mutate a column to include that number?
- What graph should we use to visualize this data?

You will be using google **a lot**. Being good at R is basically being good at googling and finding the right answer.

- Once you decide on a graph, google how to do that.
- What features should we include on the graph? How do we add those?
- What other analyses could we do with this dataset? What tests can we use?