5. UPDATING DATA

Mutate, Nested ifelse

Mutating the imported dataframe

Mutate can be used to change existing variables in a dataframe and add new variables that are mathematically related to existing variables. For both scenarios, mutate can be included in a pipe chain during the initial data import, which reduces typing because the dataframe need only be specified once.

Using mutate to change a dataframe during the initial import can also reduce errors and knitting failures. This strategy results in a single version of a dataframe that is used throughout the analysis. Consider that rarely do we write a script perfectly the first time. We don't write essays, speeches or short stories perfectly the first time, so why should coding be different? Often we return to earlier sections of a document to add or change code that we didn't realize was necessary in our first pass. Jumping around within a script to write code is dangerous though, because we can introduce changes that mean code written near the top relies upon code that appears later in the script. The code may initially run fine because the R environment stores the output of any code execution, regardless of the where the code exists in the script. Unfortunately, the script will fail to knit and throw errors for new sessions of R when we attempt to run the code sequentially, top to bottom.

One way to minimize these problems is by creating and updating a single version of our dataframe during the initial import. If we later realize the need for additional variables, we simply return to the import pipe chain and add what is missing. A second strategy is to periodically clear the R environment using the broom icon, or by executing rm(list = ls()), and then running the code chunks sequentially to flush out sequence errors.

The code below imports our data, recodes the sex and group variables and then adds 6 more variables all in one execution. The code looks intimidating at first, but consider that it does not need to be written all at once, and probably shouldn't be. An easier and safer approach is to start with just the read_csv command and only the file path to import data. After the data are imported, view the variable types and then add the col_types argument to make necessary corrections. After the updated code produces the desired result, add the rename(gender) component with a pipe operator, and again execute the code and verify the results. Continue the cycle of adding components, executing and verifying the results until the dataframe reaches a "mature" state and is ready for analysis. If you later discover the need to create more variables, simply return to the import pipe chain and make the addition.

```
df <-
    read_csv(file = "data/Friends_Cholesterol.csv",
             col_types = "cnfnfnnnnnn") %>%
    rename(sex = gender) %>%
    mutate(sex = fct_recode(sex, 'male' = '0', 'female' = '1'),
           group = fct_recode(group, 'control' = '0', 'statin' = '1'),

           # while still inside the mutate command, we create and add 6 more variables

           tc_i = hdl_i + ldl_i,
           tc_f = hdl_f + ldl_f,
           bmi_i = (weight_i/(height)^2) * 703,
           bmi_f = (weight_f/(height)^2) * 703,
           weight_min = weight_i * 18.5 / bmi_i,
```

```
           weight_max = weight_i * 24.9 / bmi_i)
```

Adding variables using base R extraction operator

While I recommend using mutate during import for adding variables (for reasons mentioned above), the extraction operator from base R can also be used to create and add new variables to a dataframe. This approach is not "wrong", but it is tedious and opens the door to errors down the road that can be difficult to identify.

Create 2 new variables using base r

```
df$tc_i <- df$hdl_i + df$ldl_i
df$tc_f <- df$hdl_f + df$ldl_f
```

Then maybe our analysis continues until we realize we need more variables...

```
df$bmi_i = (df$weight_i/(df$height)^2) * 703
df$bmi_f = (df$weight_f/(df$height)^2) * 703
```

More analysis and then whoops, we need 2 more variables...

```
df$weight_min = df$weight_i * 18.5 / df$bmi_i
df$weight_max = df$weight_i * 24.9 / df$bmi_i
```

Again, changing a dataframe throughout a script will work, but you must vigilantly avoid returning to earlier sections in the script and making changes that depend on a dataframe modified by code introduced later in the script

ifelse with mutate

In the example below, I use an iflse statement to create an obese_i variable, and then show the first 6 rows of a selection of variables. Note that after writing and testing this code chunk, we could copy the arguments of mutate into the mutate command used in the import pipe chain. We could then remove this entire code chunk and avoid working with multiple versions of df.

```
df %>%
    mutate(obese_i = ifelse(test = bmi_i >= 30,
                            yes = "obese",
                            no = "not obese"),
           obese_i = factor(obese_i)) %>%
    select(1:5, obese_i) %>%
    head
```

```
## # A tibble: 6 x 6
##   name       age sex    height group   obese_i
##   <chr>    <dbl> <fct>   <dbl> <fct>   <fct>
## 1 William     26 male       69 control not obese
## 2 Richard     37 male       67 control not obese
## 3 Joseph      45 male       66 control obese
## 4 Daniel      35 male       72 control not obese
## 5 Jennifer    44 female     64 control obese
## 6 Barbara     23 female     61 control obese
```

Nested ifelse statements

Nested ifelse statements can be used to create factor variables with more than two levels. The third argument of the first ifelse statement is simply replaced with a new ifelse statement that is then used to identify additional factor levels.

```
df <-
  df %>%
  mutate(weight_chg_rec = ifelse(test = weight_i >= weight_max,
                                 yes = "decrease weight",
                                 no = ifelse(test = weight_i <= weight_min,
                                             yes = "increase weight",
                                             no = "no change rec")),

# We can also use nested ifelse statements to create quantitative variables as shown below

         weight_chg_val = ifelse(test = weight_i >= weight_max,
                                 yes = weight_max - weight_i,
                                 no = ifelse(test = weight_i <= weight_min,
                                             yes = weight_min - weight_i,
                                             no = 0)))

# Show 6 rows of the patient identifiers and new variables.
df %>% select(1:4, weight_chg_rec, weight_chg_val)
```

```
## # A tibble: 40 x 6
##    name       age sex    height weight_chg_rec  weight_chg_val
##    <chr>    <dbl> <fct>   <dbl> <chr>                    <dbl>
##  1 William     26 male       69 decrease weight          -3.37
##  2 Richard     37 male       67 decrease weight         -25.0
##  3 Joseph      45 male       66 decrease weight         -36.7
##  4 Daniel      35 male       72 no change rec             0
##  5 Jennifer    44 female     64 decrease weight         -57.9
##  6 Barbara     23 female     61 decrease weight         -61.2
##  7 Susan       31 female     61 decrease weight         -60.2
##  8 Jessica     44 female     66 decrease weight         -37.7
##  9 Kimberly    35 female     62 decrease weight         -63.8
## 10 Emily       41 female     61 decrease weight         -71.2
## # ... with 30 more rows
```

Updating the import pipe chain

At this point we have made a lot of changes to our dataframe. We did so iteratively though, spread out over several code chunks. If this were part of a larger analysis, we might choose to combine all of these dataframe modifications into the import pipe chain, such that we produce a "mature" dataframe from the outset, and before additional code uses the dataframe for exploratory data analysis, inferential statistics and graphing. Such an import pipe chain might look intimidating, but recall that we built it up one step at a time, testing each component before adding another. Below is how such an import pipe chain might look, with comments added for readability.

```
df <-
    read_csv(file = "data/Friends_Cholesterol.csv",
             col_types = "cnfnfnnnnnn") %>%
    rename(sex = gender) %>%
```

```
    # recode the sex and group variables

    mutate(sex = fct_recode(sex, 'male' = '0', 'female' = '1'),
           group = fct_recode(group, 'control' = '0', 'statin' = '1'),

           # add 6 more variables within mutate

           tc_i = hdl_i + ldl_i,
           tc_f = hdl_f + ldl_f,
           bmi_i = (weight_i/(height)^2) * 703,
           bmi_f = (weight_f/(height)^2) * 703,
           weight_min = weight_i * 18.5 / bmi_i,
           weight_max = weight_i * 24.9 / bmi_i,

           # add categorical weight change recommendation variable within mutate

           weight_chg_rec = ifelse(test = weight_i >= weight_max,
                                   yes = "decrease weight",
                                   no = ifelse(test = weight_i <= weight_min,
                                               yes = "increase weight",
                                               no = "no change rec")),

           # add quantitative weight change recommendation variable within mutate

           weight_chg_val = ifelse(test = weight_i >= weight_max,
                                   yes = weight_max - weight_i,
                                   no = ifelse(test = weight_i <= weight_min,
                                               yes = weight_min - weight_i,
                                               no = 0))
           )
```

Adding rows & columns to dataframes

in progress

rbind

in progress

cbind

in progress

Merging dataframes

in progress