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

Suggested Solutions

9/10/2020

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

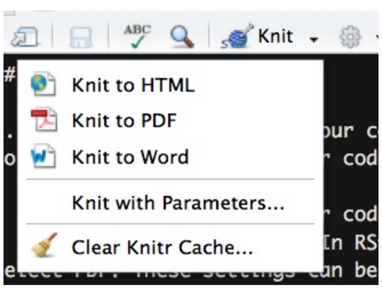
Data:

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

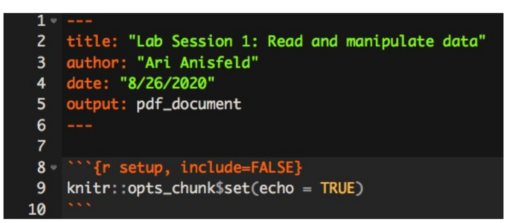
1. Download the Stata file and Readme for “College Level Characteristics from the IPEDS Database and the College Scorecard”.

Intro to `Rmds`

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 `Rmds` 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. `Rmds` 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]

```r
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.

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?

Solution a. `TRUE` b. `TRUE` c. `TRUE` d. `Error, (10 < 4 & ) is an incomplete statement` e. `TRUE` f. `TRUE` g. OR (`|`) returns `TRUE` if at least one term is `TRUE`

True and False

a. `TRUE & FALSE`
b. `TRUE & (FALSE & FALSE)`
c. `TRUE & (10 < 4)`

[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

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?

Solution a. `FALSE` b. `FALSE` c. `FALSE` d. `Error, (10 < 4 & ) is an incomplete statement` e. `FALSE` f. `FALSE` g. AND (`&`) returns `TRUE` *only* when *both* operands are `TRUE`

True and NA

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

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

Solution

```r
TRUE & NA
```

```
## [1] NA
```

```r
FALSE & NA
```

```
## [1] FALSE
```

```r
TRUE | NA
```

```
## [1] TRUE
```

```r
FALSE | NA
```

```
## [1] NA
```

`NA` is valid logical object, and the results of statements including `NA` will evaluate to `NA` when the outcome is ambiguous. Think of `NA` as an *unknown*.

`TRUE & NA` evaluates to `NA` because we do not know if `NA` represents a `TRUE` or `FALSE` statements, so the outcome is ambiguous. If `NA` were a `TRUE` statement, then this example would evaluate to `TRUE`. If `NA` were a `FALSE` statement, then this example would evaluate to `FALSE`.

`FALSE & NA` evaluates to `FALSE` because at least one of the statements is `FALSE`.

`TRUE | NA` evaluates to `TRUE` because at least one of the statements is `TRUE`.

`FALSE | NA` evaluates to `NA` again because we do not know if `NA` represents a `TRUE` or `FALSE` statements, so the outcome is ambiguous. If `NA` were a `TRUE` statement, then this example would evaluate to `TRUE`. If `NA` were a `FALSE` statement, then this example would evaluate to `FALSE`.

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

```r
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")
```

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

Solution

```r
ifelse(TRUE, "yes", "no")
```

```
## [1] "yes"
```

```r
ifelse(FALSE, "yes", "no")
```

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

```r
ifelse(c(TRUE, FALSE, TRUE, FALSE), "yes", "no")
```

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

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

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

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

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

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

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

Recall that `ifelse()` asks, "Does the statement in the first position evaluate to `TRUE`?" If it does, then return what sits in the second position. If it does not, then return what sits in the third position. As always, `NA` will be returned if the outcome of a statement is ambiguous.

If what sits in the first position is a vector, then repeat this process for every element of the vector one by one.

Common uses of `ifelse`

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

```r
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]

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

Solution

```r
mrc_data %>%
  mutate(tier_name = ifelse(
          tier_name == "Two-year (public and private not-for-profit)",
          "Two-year (public and private)",
```

[4]Hint: In the first position, put a condition testing if `tier_name` matches the string. If it does, we replace the string with "Two-year (public and private)", otherwise keep the same data.

```r
            tier_name)) %>%
  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)
##  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
```

Notice that a good way to check if you've executed this correctly, is to call `distinct()` once again and see for yourself if "Two-year (public and private not-for-profit)" is still a value for `tier_name`.

```
b. `ifelse` is often used to collapse tiers. Redefine `tier_name` so that "Nonselective four-year publi
```

Solution

```r
mrc_data %>%
  mutate(tier_name = ifelse(
            tier_name == "Nonselective four-year public" |
            tier_name == "Nonselective four-year private not-for-profit",
            "Nonselective four-year (public and private)",
            tier_name)) %>%
  distinct(tier_name)
```

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

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.[5]

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

   ```
   ## # A tibble: 1 x 1
   ```

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

```
##   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.[6]

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
```

Solution

```
mrc_data <- mrc_data %>%
  mutate(missing_sat_avg_2013 = ifelse(
           is.na(sat_avg_2013), 1, 0),
         sat_avg_2013 = ifelse(
           is.na(sat_avg_2013),
           mean(sat_avg_2013, na.rm = TRUE),
           sat_avg_2013))

mrc_data %>%
  summarise(sum(is.na(sat_avg_2013)),
            sum(missing_sat_avg_2013),
            sum(sat_avg_2013 == mean(sat_avg_2013, na.rm = TRUE)))
```

```
## # A tibble: 1 x 3
##   `sum(is.na(sat_avg_2~ `sum(missing_sat_avg~ `sum(sat_avg_2013 == mean(sat_avg~
##                   <int>                 <dbl>                              <int>
## 1                     0                  1466                               1466
```

To check your work, you can use summarize() again. We see the following. There are no more `NA` values in the `sat_avg_2013` column. There are 1466 instances of a value `1` in `missing_sat_avg_2013`, which matches the number of `NA` values were originally present in `sat_avg_2013`. There at 1466 instances of `sat_avg_2013` equalling the mean of `sat_avg_2013`, so we can feel fairly confident that `NA` values were replaced by the mean.

[6] 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)`.

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

```
## # A tibble: 1 x 1
##   missing_sat_2013
##              <int>
## 1                0
```

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.

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()`[7]. 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`.

   ```r
   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.

[7] from `dplyr`. The codebook is also useful.

2. Second, we're going to have to find schools that match ranges of SAT scores. Welcome `between()`.[8]

   ```
   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 `<=`?

Solution

As always, `?between` is the first thing to do. In the R Documentation, you should see "This is a shortcut for x >= left & x <= right, implemented efficiently in . . . "

```
?between
```

You can also test things out in R. In this case, we see that the lower bound (1330) is included in the final output. However, this way would not be helpful to tell whether `between()` use `<` or `<=` if there is no 1330 entry in the data.

```
mrc_data %>%
  filter(between(sat_avg_2013, 1330, 1530)) %>%
  summarize(min(sat_avg_2013),
            max(sat_avg_2013))
```

```
## # A tibble: 1 x 2
##   `min(sat_avg_2013)` `max(sat_avg_2013)`
##                 <dbl>               <dbl>
## 1                1330               1518.
```

3. The final thing is a concept. You're probably about to write code that looks like the following pseudo code.[9]

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

   b. Test out the concept with a simple example.[10]

Solution

```
mrc_data %>%
  mutate(abdul_choices = ifelse(
            tier_name == "Ivy Plus", TRUE, FALSE)) %>%
  filter(abdul_choices) %>%
  select(name,tier_name,abdul_choices,sat_avg_2013,flagship)
```

[8] from `dplyr`

[9] pseudo code is a term for fake code that captures the logic of some coding idea without being actual code.

[10] For example, try it with Abdul's only condition being Ivy Plus.

```
## # A tibble: 12 x 5
##    name                             tier_name abdul_choices sat_avg_2013 flagship
##    <chr>                            <chr>     <lgl>                <dbl>    <dbl>
##  1 Brown University                 Ivy Plus  TRUE                  1440        0
##  2 Columbia University In The Cit~ Ivy Plus  TRUE                  1480        0
##  3 Cornell University               Ivy Plus  TRUE                  1420        0
##  4 Dartmouth College                Ivy Plus  TRUE                  1455        0
##  5 Duke University                  Ivy Plus  TRUE                  1460        0
##  6 Harvard University               Ivy Plus  TRUE                  1505        0
##  7 Massachusetts Institute Of Tec~ Ivy Plus  TRUE                  1495        0
##  8 Princeton University             Ivy Plus  TRUE                  1500        0
##  9 Stanford University              Ivy Plus  TRUE                  1475        0
## 10 University Of Chicago            Ivy Plus  TRUE                  1518.       0
## 11 University Of Pennsylvania       Ivy Plus  TRUE                  1455        0
## 12 Yale University                  Ivy Plus  TRUE                  1505        0
```

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),
         stephen_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, casey_choices)
```

Solution

```
bff_super_awesome_college_list <- mrc_data %>%
  mutate(abdul   = ifelse(between(sat_avg_2013, 1330, 1530) &
                           (tier_name == "Ivy Plus" | flagship == 1),
                          TRUE, FALSE),
         stephen = ifelse(between(sat_avg_2013, 1350, 1550) & public == 0,
                          TRUE, FALSE),
         nancy   = ifelse(between(sat_avg_2013, 1490, 1690) & region == 2,
                          TRUE, FALSE),
         rei     = ifelse(between(sat_avg_2013, 1390, 1590) & pct_socialscience_2000 > 30,
                          TRUE, FALSE),
         casey   = ifelse(between(sat_avg_2013, 1500, 1600) &
                           (public == 1 & state == "CA" | scorecard_netprice_2013 < 10000),
                          TRUE, FALSE))
```

c. What school(s) are acceptable to all five

Solution

```
bff_super_awesome_college_list %>%
  filter(abdul, stephen, nancy, rei, casey) %>%
```

```
  select(name,abdul, stephen, nancy, rei, casey)
```

```
## # A tibble: 1 x 6
##   name                    abdul stephen nancy rei   casey
##   <chr>                   <lgl> <lgl>   <lgl> <lgl> <lgl>
## 1 University Of Chicago TRUE  TRUE    TRUE  TRUE  TRUE
```

```
# Alternative (notice it yields the same result)
bff_super_awesome_college_list %>%
  filter(abdul & stephen & nancy & rei & casey) %>%
  select(name,abdul, stephen, nancy, rei, casey)
```

```
## # A tibble: 1 x 6
##   name                    abdul stephen nancy rei   casey
##   <chr>                   <lgl> <lgl>   <lgl> <lgl> <lgl>
## 1 University Of Chicago TRUE  TRUE    TRUE  TRUE  TRUE
```

d. How many school(s) are available to any of the five. Adjust `filter` statement slightly.[11]

Solution

```
bff_super_awesome_college_list %>%
  filter(abdul | stephen | nancy | rei | casey) %>%
  nrow()
```

```
## [1] 52
```

5. 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`.

Solution Yes, the school list will change if we replace all `NA`s with `TRUE`. The list will not change if we replace `NA`s with `FALSE` since we are using the OR (|) operator, and this operator evaluates to `TRUE` so long as at least one statement is `TRUE`.

6. **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.[12] Colleges that are not acceptable to any friend should be marked "Unranked".

Solution 1

We take advantage of the fact that the arithmetic operation of booleans (`TRUE` and `FALSE`) works like 1's and 0's. Since `abdul`, `stephen`, `nancy`, `rei`, and `casey` are logical vectors (vectors full of `TRUE`s and `FALSE`s), we get the number of friends who would accept going to that school when we add across the rows which represent schools. Finally, we use `case_when` to assign the rankings according to how many friends would accept going to that school.

```
bff_super_awesome_college_list %>%
  mutate(num_friends_accept = abdul + stephen + nancy + rei + casey,
         rank = case_when(num_friends_accept == 5 ~ "1",
                          num_friends_accept == 4 ~ "2",
                          num_friends_accept == 3 ~ "3",
                          num_friends_accept == 2 ~ "4",
                          num_friends_accept == 1 ~ "5",
                          num_friends_accept == 0 ~ "Unranked")) %>%
  arrange(rank) %>%
  select(name, num_friends_accept, rank)
```

[11] Hint: Think about the warm-up you did for this lab

[12] 3 if it is acceptable to 3 friends. 4 if acceptable to 2 friends and 5 if acceptable to 1 friend

```
## # A tibble: 2,463 x 3
##    name                                    num_friends_accept rank
##    <chr>                                                <int> <chr>
##  1 University Of Chicago                                    5 1
##  2 Harvard University                                       4 2
##  3 Princeton University                                     4 2
##  4 Yale University                                          4 2
##  5 Brown University                                         3 3
##  6 Columbia University In The City Of New York              3 3
##  7 Dartmouth College                                        3 3
##  8 Duke University                                          3 3
##  9 Stanford University                                      3 3
## 10 University Of Pennsylvania                               3 3
## # ... with 2,453 more rows
```

Solution 2

Listing all the possible cases can be mundane when the number of cases is very large. There are two functions that can be very helpful for this exercise: `recode()` and `dense_rank()`. If you are not sure what is going on in the `mutate()` step, please see the simple example provided below.

```
bff_super_awesome_college_list %>%
  mutate(num_friends_accept = abdul + stephen + nancy + rei + casey,
         rank = recode(as.character(dense_rank(-num_friends_accept)),"6" = "Unranked")) %>%
  arrange(rank) %>%
  select(name, num_friends_accept, rank)
```

```
## # A tibble: 2,463 x 3
##    name                                    num_friends_accept rank
##    <chr>                                                <int> <chr>
##  1 University Of Chicago                                    5 1
##  2 Harvard University                                       4 2
##  3 Princeton University                                     4 2
##  4 Yale University                                          4 2
##  5 Brown University                                         3 3
##  6 Columbia University In The City Of New York              3 3
##  7 Dartmouth College                                        3 3
##  8 Duke University                                          3 3
##  9 Stanford University                                      3 3
## 10 University Of Pennsylvania                               3 3
## # ... with 2,453 more rows
```

Simple example

`dense_rank()` in dplyr can give you continuous ranks.

```
simple.example <- c(5,2,1,4,3)
dense_rank(simple.example)
```

```
## [1] 5 2 1 4 3
```

This is a common trick to obtain the descending ranks.

```
dense_rank(-simple.example)
```

```
## [1] 1 4 5 2 3
```

What if we want to label the last rank (which is 5 in this case) with "Last"? In the lecture, Ari has showed your an example where R cannot handle "numeric" and "character" type entries in the same column. To update the label with characters, let's first convert the numeric ranks into characters.

```
simple.rank <- as.character(dense_rank(-simple.example))
```

Finally, let's relabel `5` with `"Last"`. N.B. There are many other ways to do so.

```
recode(simple.rank, "5"="Last")
```

```
## [1] "1"    "4"    "Last" "2"    "3"
```