Accelerated Lecture 4: If statements and conditionals

Harris Coding Camp – Standard Track

Summer 2022

Review: Subsetting data

```r
# tidyverse
data |>
  filter(row_condition) |>
  select(columns, we, want)

# base R
data[row_condition, c("columns", "we", "want")]
```

Review: Do these `filter()` calls give the same result?

```
south_africa_data |>
  filter(percentile == "p0p50",
         value == max(value, na.rm = TRUE))

south_africa_data |>
  filter(percentile == "p0p50") |>
  filter(value == max(value, na.rm = TRUE))
```

Review: Why not?

```
identical(
south_africa_data |>
  filter(percentile == "p0p50",
         value == max(value, na.rm = TRUE)),
south_africa_data |>
  filter(percentile == "p0p50") |>
  filter(value == max(value, na.rm = TRUE))
)
```

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

Review: Subsetting data

```r
south_africa_data |>
  filter(percentile == "p0p50",
         value == max(value, na.rm = TRUE))

# this is the max for ALL income groups.
max(south_africa_data$value, na.rm = TRUE)
```

Review: Subsetting data

Here, we use the max() on the bottom half only.

```
south_africa_data |>
  filter(percentile == "p0p50") |>
  filter(value == max(value, na.rm = TRUE))
```

We could rewrite the code like so.

```
bottom_half <- south_africa_data |>
  filter(percentile == "p0p50")

# this is the max for the bottom half income groups.
bottom_half |>
  filter(value == max(value, na.rm = TRUE))
```

```
## # A tibble: 1 x 4
##   country      percentile  year  value
##   <chr>        <chr>      <dbl>  <dbl>
## 1 South Africa p0p50       2004 0.0018
```

Review: Sorting data

```r
# tidyverse
arrange(data, col, desc(col2))

# base R
data[order(data$col, -data$col2),]
```

Review: Summarizing data

```r
# tidyverse
# results in a tibble with 1 row
summarize(data, mean = mean(col))

# base R
# results in a vector of length 1
mean(data$col)
```

- Let `mean` stand in for any function that *reduces* a vector to a **single value**

Review: Creating new data

```
# tidyverse
data <- data |> mutate(new_column = something)

# base R
data$new_column <- something
```

Same functionality to change old data:

```
# tidyverse
data <- data |> mutate(old_column = something)

# base R
data$old_column <- something
```

- `something` is a vector length nrow(data) or 1

Quiz time – no stakes!!

Do your best! Do it by yourself!

- Only resource is R and RStudio

Go to Canvas

- Click on `Gradescope`
- Click on `Accelerated Quiz`
- 10 minute timer starts when you open it

How would we make a column dependent on other data?

```
## # A tibble: 4 x 2
##       x     y
##   <int> <dbl>
## 1     1  -1.5
## 2     2   1.6
## 3     3  -1
## 4     4  -0.9
```

Add column dependent on `y`

```
## # A tibble: 4 x 3
##       x     y set_neg_y_to_0
##   <int> <dbl>          <dbl>
## 1     1  -1.5            0
## 2     2   1.6            1.6
## 3     3  -1              0
## 4     4  -0.9            0
```

Call in `if` and `ifelse`

When we want code to do something depending on the context

```
ifelse(test, yes, no)



if (test is TRUE) {
  do this
} else {  # test is FALSE
  do this other thing
}
```

We will cover:

- introduce vectorized `ifelse` and `case_when()` statements
- introduce `if` and `else` statements

Syntax: `ifelse(test, yes, no)`

- **if**`test` is TRUE return `yes`
- **else** `test` is FALSE return `no`

```r
x <- c(5, 50)

ifelse(x > 10, "x is big", "x is small")
```

```
## [1] "x is small" "x is big"
```

`ifelse(test, yes, no)` is vectorized

```
x <- c(5, 50, log(1e11), -1)
# 5 > 10 ?              ... no
# 50 > 10 ?             ... yes
# log(1e11) > 10 ?      ... yes
# -1 > 10 ?             ... no
ifelse(x > 10, "x is big", "x is small")
```

```
## [1] "x is small" "x is big"   "x is big"   "x is small"
```

What will the following statements return?

```r
ifelse(is.na(NA), 1, 2)



ifelse(is.na("a"), 1, 2)
```

`test` typically evaluates to a boolean vector

Think of:

- conditional operators
- is.() tests

```r
# ifelse(is.na(NA), 1, 2)
ifelse(TRUE, 1, 2)
```

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

```r
# ifelse(is.na("a"), 1, 2)
ifelse(FALSE, 1, 2)
```

```
## [1] 2
```

TRUE gives option 1, FALSE gives option 2

```r
# ifelse(TRUE, 1, 2)
ifelse(is.na(NA), 1, 2)
```

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

```r
# ifelse(FALSE, 1, 2)
ifelse(is.na("a"), 1, 2)
```

```
## [1] 2
```

What will the following statements return?

```r
ifelse(c(TRUE, FALSE, FALSE, TRUE), "a", "b")

ifelse(1:4 > 3, "a", "b")
```

What will the following statements return?

```
ifelse(c(TRUE, FALSE, FALSE, TRUE), "a", "b")
```

```
## [1] "a" "b" "b" "a"
```

```
ifelse(1:4 > 3, "a", "b")
```

```
## [1] "b" "b" "b" "a"
```

Another example

```r
trial_1 <- c(98, 20, 100, 18, 40)
trial_2 <- c(30, 41, 64, 8, 70)

ifelse(trial_1 > trial_2, trial_1, trial_2)
```

```
## [1]  98  41 100  18  70
```

What should the following code returns?

```r
states <- c("IL", NA, "IA", "NM")

ifelse(states == "IL", "home", "elsewhere")
```

NA still contagious

```r
states <- c("IL", NA, "IA", "NM")

ifelse(states == "IL", "home", "elsewhere")
```

```
## [1] "home"      NA          "elsewhere" "elsewhere"
```

Using ifelse with data

Add a column called vowel which is 1 for “a”, “e”, “i”, “o” and “u” and 0 otherwise.[1]

```
alphabet <- tibble(letters = letters)
head(alphabet)
```

```
## # A tibble: 6 x 1
##   letters
##   <chr>
## 1 a
## 2 b
## 3 c
## 4 d
## 5 e
## 6 f
```

[1] Sorry “y”!

Call ifelse() inside mutate()!

```
alphabet |>
  mutate(vowel =
          ifelse(letters %in% c("a", "e", "i", "o","u"),
                 1, 0)) |>
  head()
```

```
## # A tibble: 6 x 2
##   letters vowel
##   <chr>   <dbl>
## 1 a           1
## 2 b           0
## 3 c           0
## 4 d           0
## 5 e           1
## 6 f           0
```

Code that works on vectors, will work on columns. After all, they're vectors!

Of course, you'll see baseR do this too

```r
alphabet$vowel <-
  ifelse(alphabet$letters %in% c("a", "e", "i", "o","u"),
         1, 0)
```

Are we stuck with two outcomes?

`ifelse` statements with multiple categories

Let's make the `vowel` column

- "yes" for "aeiou"
- "sometimes" for "y"
- "no" for everything else

```
tail(alphabet)
```

```
## # A tibble: 6 x 1
##   letters
##   <chr>
## 1 u
## 2 v
## 3 w
## 4 x
## 5 y
## 6 z
```

Option 1: call ifelse multiple times

```
alphabet |>
  mutate(
    vowel = "no",
    vowel = ifelse(letters == "y", "sometimes", vowel),
    vowel = ifelse(letters %in%  c("a", "e", "i", "o", "u")
                   "yes", vowel)
  ) |>
  tail()
```

```
## # A tibble: 6 x 2
##   letters vowel
##   <chr>   <chr>
## 1 u       yes
## 2 v       no
## 3 w       no
## 4 x       no
## 5 y       sometimes
## 6 z       no
```

Option 1: call `ifelse` multiple times

```
alphabet |>
  mutate(
    vowel = "no"
  ) |>
  tail()
```

```
## # A tibble: 6 x 2
##   letters vowel
##   <chr>   <chr>
## 1 u       no
## 2 v       no
## 3 w       no
## 4 x       no
## 5 y       no
## 6 z       no
```

Option 1: call ifelse multiple times

```
alphabet |>
  mutate(
    vowel = "no",
    vowel = ifelse(letters == "y", "sometimes", vowel)
  ) |>
  tail()
```

```
## # A tibble: 6 x 2
##   letters vowel
##   <chr>   <chr>
## 1 u       no
## 2 v       no
## 3 w       no
## 4 x       no
## 5 y       sometimes
## 6 z       no
```

Option 1: call `ifelse` multiple times

```
alphabet |>
  mutate(
    vowel = "no",
    vowel = ifelse(letters == "y", "sometimes", vowel),
    vowel = ifelse(letters %in%  c("a", "e", "i", "o", "u")
                 "yes", vowel)
  ) |>
  tail()
```

```
## # A tibble: 6 x 2
##   letters vowel
##   <chr>   <chr>
## 1 u       yes
## 2 v       no
## 3 w       no
## 4 x       no
## 5 y       sometimes
## 6 z       no
```

Option 2: nest the ifelse

```
alphabet |>
  mutate(
    vowel = ifelse(letters == "y",
                   "sometimes",
                   ifelse(letters %in%  c("a", "e", "i", "o", "u"
                          "yes", "no"))
  ) |>
  tail()
```

```
## # A tibble: 6 x 2
##   letters vowel
##   <chr>   <chr>
## 1 u       yes
## 2 v       no
## 3 w       no
## 4 x       no
## 5 y       sometimes
## 6 z       no
```

option 3: case_when()

```
alphabet |>
  mutate(
    vowel =
      case_when(
        letters == "y" ~ "sometimes",
        letters %in%  c("a", "e", "i", "o", "u") ~ "yes",
        TRUE ~  "no")
  ) |>
  tail()
```

```
## # A tibble: 6 x 2
##   letters vowel
##   <chr>   <chr>
## 1 u       yes
## 2 v       no
## 3 w       no
## 4 x       no
## 5 y       sometimes
```

Another nested `ifelse` example

```r
txhousing |>
  select(city, year, month, median) |>
  mutate(housing_market =
      ifelse(median < 100000, "first quartile",
      ifelse(median < 123800, "second quartile",
      ifelse(median < 150000, "third quartile",
      ifelse(median < 350000, "fourth quartile",
             NA))))
           ) |>
  head(3)
```

```
## # A tibble: 3 x 5
##   city     year month median housing_market
##   <chr>   <int> <int>  <dbl> <chr>
## 1 Abilene  2000     1  71400 first quartile
## 2 Abilene  2000     2  58700 first quartile
## 3 Abilene  2000     3  58100 first quartile
```

case_when again

```r
# add a column called `housing_market` to the `txhousing`
txhousing |>
  select(city, year, month, median) |>
  mutate(housing_market =
         case_when(
           median < 100000 ~ "first quartile",
           median < 123800 ~ "second quartile",
           median < 150000 ~ "third quartile",
           median < 350000 ~ "fourth quartile"
         ))  |>
  head(3)
```

case_when statements are a bit “surly”

`case_when` will not do type coercion.

```
txhousing |>
  mutate(housing_market =
        case_when(
           median < 100000 ~ 1,
           median < 123800 ~ "second quartile",
           median < 150000 ~ "third quartile",
           median < 350000 ~ "fourth quartile"
         )) |>
  select(city, median, housing_market)

Error: must be a double vector, not a character vector
Run `rlang::last_error()` to see where the error occurred.
```

Here we try to include *both* doubles and characters in the `housing_market` column, but atomic vectors can only have one type!

- Rather than coerce and provide a warning, the developers decided to make this an error
- If using `NA` as an output, you have to specify `NA` types e.g. `NA_integer_`, `NA_character_`

case_when “else”

You might wonder how to approximate else.

- Use TRUE as a catch all.

```
example <- tibble(a = 1:12)

example |>
  mutate(category = case_when(a %in% c(2, 3, 5, 7, 11) ~ "prime",
                              sqrt(a) == round(sqrt(a))  ~ "square",
                              TRUE ~ "other"))
```

Try it yourself

We will use `midwest` here, which is a dataset built into `tidyverse`.

1. Create a new variable, `poverty_designation`, that is “High Poverty” if `percbelowpoverty` is above 10 and is “Low Poverty” otherwise.
2. Create a new variable called `ohio` that is “Ohio Counties” for observations from Ohio and “Other Midwestern Counties” for the rest of the observations.
3. Create a new variable called `populous_counties` that is `TRUE` for the observations from the counties listed in `big_counties` and `FALSE` otherwise.

```r
big_counties <- c("COOK", "WAYNE", "CUYAHOGA", "OAKLAND", "FRANKLIN")
```

4. Create a new variable called `pop_index` that is “High” for the observations with `poptotal` greater than 100000, is “Medium” for the observations with `poptotal` between 30000 and 100000, and “Low” otherwise.

if statements

if statements

```
if (condition is TRUE) {
  do this
  ...
  ...
  ...
}
```

if statements

For example:

```r
x <- 100

if (x > 0) {
  print("x is positive")
}
```

```
## [1] "x is positive"
```

if/else statements

```
if (condition is TRUE) {
  do this
} else {
  do this other thing
}
```

if/else statements, example

```r
x <- -5
if (x > 0) {
  print("Non-negative number")
} else {
  print("Negative number")
}
```

```
## [1] "Negative number"
```

if and else versus ifelse

ifelse

- often used in a data setting
- handy for quick yes, no type alternatives
- vectorized and accepts NA

if and else

- often used in a "programming" setting
- handle complicated chunks of code and more complex alternatives
- if() only accepts TRUE or FALSE (not vectorized, no NA)

if, else if and else statements

If we have more than 2 conditions, use if, else if and else:

```
if (condition is TRUE) {
  do this
} else if (second condition is TRUE) {
  do this other thing
} else if (third condition is TRUE) {
  do this third thing
} else {
  do a default behavior
}
```

Note: a default behavior with else is not necessary.

if, else if and else statements, example

```r
x <- sample(1:100, 1)
x
```

```
## [1] 92
```

```r
y <- sample(1:100, 1)
y
```

```
## [1] 34
```

```r
if (x > y) {
  print("x is greater")
} else if (x < y) {
  print("y is greater")
} else {
  print("x and y are equal")
}
```

```
## [1] "x is greater"
```

`if`, `else if` and `else` can take a compound condition

```r
x <- sample(1:100, 1)
x
```

```
## [1] 8
```

```r
y <- sample(1:100, 1)
y
```

```
## [1] 91
```

```r
z <- sample(1:100, 1)
z
```

```
## [1] 82
```

if, else if and else can take a compound condition

```r
if (x >= y & x >= z) {
  print("x is the greatest")
} else if (y >= z) {
  print("y is the greatest")
} else {
  print("z is the greatest")
}
```

```
## [1] "y is the greatest"
```

Try it yourself

Let's develop a small dice game.

1. Fill in the ... so the code says "You win" if the dice add up to 7 and "You lose" otherwise.

```r
dice <- sample(c(1:6), 2)

if (...) {
  print("You win")
} else {
  print("You lose")
}
```

2. Add an `else if()` block to the code above that says "try again" if the dice add up to 6 or 8.

Try it yourself

2. Add an `else if()` block to the code above that says “Try again” if the dice add up to 6 or 8.

```r
dice <- sample(c(1:6), 2)

if (...) {
  print("You win")
} else if (...) {
  print("Try again")
} else {
  print("You lose")
}
```

Some common uses of `if`

Sharing code among various people.

- `Sys.getenv("USER")` returns the name of the USER fr

```
if (Sys.getenv("USER") == "arianisfeld") {
  setwd("~/repo/dir")
} else if (Sys.getenv("USER") == "yunjoo") {
  setwd("C://repo/dir")
} else {
  print(paste0("WARNING: Unknown user.
               Working directory is ", getwd()))
}
```

if() the condition must return TRUE or FALSE

if() is not vectorized

```r
x <- c(1, -4)

if (x > 0) {
   x
} else {
  -x
}
```

```
Error in if (x > 0) { : the condition has length > 1
```

if() the condition must return TRUE or FALSE

if() does not handle NAs

```
x <- NA

if (x > 0) {
  x
} else {
  -x
}
```

```
Error in if (x > 0) { : missing value where
TRUE/FALSE needed
```

If you can't afford errors . . .

write code to handle edge cases

```r
x <- NA
if (length(x) == 1 & all(!is.na(x) & x > 0)) {
  x
} else if (length(x) == 1) {
  -x
}
```

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

```r
x <- c(pi, 2)
out <- if(length(x) == 1 & all(!is.na(x) & x > 0)) {
  x
} else if(length(x) == 1) {
  -x
}
out
```

Detour: Why NULL? Why not NA?

NULL stands in for an *object* that is undefined.

```
length(NULL)
```

```
## [1] 0
```

```
NULL > 1
```

```
## logical(0)
```

∼∼∼

NA stands in for a *value* that is undefined.

```
length(NA)
```

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

```
NA > 1
```

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

`if()` the condition must return TRUE or FALSE

Good idea to make sure it still works for valid input!

```r
x <- exp(1)
if (length(x) == 1 & all(!is.na(x) & x > 0)) {
  x
} else if (length(x) == 1) {
  -x
}
```

```
## [1] 2.718282
```

```r
x <- -1000
if (length(x) == 1 & all(!is.na(x) & x > 0)) {
  x
} else if (length(x) == 1) {
  -x
}
```

```
## [1] 1000
```

Recap

Today we learned how to:

- use control flow with `if` and `ifelse` statements
- use `ifelse()` and `case_when()` statements in conjunction with `mutate()` or `$<-` to create columns based on conditional statements

Next up

Lab:

- Today: Practice with `ifelse`

I can use `ifelse` to create columns conditional on data

Lecture:

- Making data visualizations