Accelerated Lecture 4: If statements and conditionals

Harris Coding Camp – Standard Track

Summer 2022

Review: Subsetting data

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

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

Remark: You may also see base R's `subset()`

Review: Sorting data

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

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

Review: Summarizing data

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

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

- We let `mean` stand in for any function that reduces the data to a single value (per group)
- *Remark* base R code can get more sophisticated

Review: Creating new data

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

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

Same functionality to change old data:

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

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

- something is length nrow(data) or 1

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`

Conditional statements

We often want to our code to do something depending on the context. We start with `if` statements.

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

We will cover:

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

if statements

The general syntax of an `if` statement is as follows:

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

For example:

```
x <- 100

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

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

if/else statements

Slightly more interesting, the syntax of an `if/else` statement is as follows:

```
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, 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.

```
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()))
}
```

Some common uses of `if`

In a function, you might want to adjust to different inputs.

example: How can we code up the absolute value function?

$$|x| = \begin{cases} x, & x >= 0 \\ -x, & x < 0 \end{cases}$$

Some common uses of `if`

In a function, you might want to adjust to different inputs.

```r
absolute_value <- function(x) {
  # x: a numeric of length 1

  if (x < 0) {

    x <- x * -1

  }

  return(x)
}
```

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
```

- This error is as of R 4.2.0

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

`if()` does not handle `NA`s

```r
x <- NA

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

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

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

You may need to write code to handle edge cases.

```r
x <- NA

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

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

```r
x <- c(123, 1)
out <- if (all(length(x) == 1 & !is.na(x) & x > 0)) {
  x
} else if (length(x) == 1) {
  -x
}
out
```

```
## NULL
```

Detour: NULL vs 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!

```
x <- 1309
if (all(length(x) == 1 & !is.na(x) & x > 0)) {
  x
} else if (length(x) == 1) {
  -x
}
```

```
## [1] 1309
```

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

```
## [1] 3.141593
```

Vectorized `ifelse` statements

In R, the `ifelse()` function is a shorthand vectorized alternative to the standard `if...else` statement.

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

```r
x <- 5
y <- 50
ifelse(x > y, "x is greater", "x is not greater")
```

```
## [1] "x is not greater"
```

```r
a <- 60
b <- 6
ifelse(a > b, "a is greater", "a is not greater")
```

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

What will the following statements return?

```r
ifelse(TRUE, 1, 2)
ifelse(FALSE, 1, 2)
```

What will the following statements return?

```r
ifelse(TRUE, 1, 2)
```

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

```r
ifelse(FALSE, 1, 2)
```

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

What will the following statements return?

```r
ifelse(c(TRUE, FALSE, FALSE, TRUE), 1, 2)

ifelse(1:4 > 3, 1, 2)
```

What will the following statements return?

```r
ifelse(c(TRUE, FALSE, FALSE, TRUE), 1, 2)
```

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

```r
ifelse(1:4 > 3, 1, 2)
```

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

`ifelse` handles NAs and missing data

What's going on in this `ifelse()` statement?

```
ifelse(NA, 1, 2)
```

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

```
ifelse(NULL, 1, 2)
```

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

- `NA`s are contagious.

ifelse statements in dataframes, base R

ifelse statements work well in dataframes when we need to create a new column.

- Let's add a column to the txhousing based on a conditional.

```
txhousing$in_january <-
  ifelse(txhousing$month == 1, TRUE, FALSE)
```

`ifelse` statements in dataframes for multiple categories

If we have a lot of categories, use **nested** `ifelse` statement.
Say we want to create a new variable called `median_ref` which value can be High, Medium or Low:

- If `median` is at least 70k: High
- If `median` is between 60k and 70k: Medium
- If `median` is lower than 60k: Low

```
txhousing$median_ref <-
  ifelse(txhousing$median >= 70000, 'High',
         ifelse(txhousing$median < 70000 &
                txhousing$median >= 60000,
                'Medium', 'Low'))
```

ifelse statements in dataframes, tidyverse

Use ifelse statements in mutate() function.

- Let's add a column called in_january to the txhousing based on a condition.

```
txhousing %>%
  mutate(in_january = ifelse(month == 1, TRUE, FALSE)) %>%
  select(city, year, month, sales, in_january)
```

```
## # A tibble: 8,602 x 5
##    city     year month sales in_january
##    <chr>   <int> <int> <dbl> <lgl>
##  1 Abilene  2000     1    72 TRUE
##  2 Abilene  2000     2    98 FALSE
##  3 Abilene  2000     3   130 FALSE
##  4 Abilene  2000     4    98 FALSE
##  5 Abilene  2000     5   141 FALSE
##  6 Abilene  2000     6   156 FALSE
##  7 Abilene  2000     7   152 FALSE
##  8 Abilene  2000     8   131 FALSE
##  9 Abilene  2000     9   104 FALSE
## 10 Abilene  2000    10   101 FALSE
## # ... with 8,592 more rows
```

ifelse statements in dataframes, tidyverse

As before, we can handle nested statements with `ifelse()`

```
txhousing %>%
  select(city, year, month, median) %>%
  mutate(housing_market =
      ifelse(median < 100000, "first quartile",
      ifelse(100000 <= median & median < 123800, "second quartile",
      ifelse(123800 <= median & median < 150000, "third quartile",
      ifelse(150000 <= median & 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 statements in dataframes, tidyverse

There's a cleaner way to handle multiple cases.

- Instead of nesting `ifelse` statements we can use `case_when()`

```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",
           100000 <= median & median < 123800 ~ "second quartile",
           123800 <= median & median < 150000 ~ "third quartile",
           150000 <= median & 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,
           100000 <= median & median < 123800 ~ "second quartile",
           123800 <= median & median < 150000 ~ "third quartile",
           150000 <= median & 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_`

Try it yourself

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

1. Create a new variable called `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. Hint: Use the `%in%` operator.

```
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.

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

Labs:

- Today: Practice with `ifelse`
- Tomorrow: Coding style, review and catch-up.

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

and

I'm gaining confidence doing basic data manipulation

Lecture:

- Making data visualizations