

# **Intermediate dplyr**

**Anthony Chau**

**UCI Center for Statistical Consulting**

**7/24/2021 (updated: 2021-07-26)**

# Beyond core dplyr

- `dplyr` provides many useful functions beyond the core verbs (`select`, `filter`, `mutate`, etc)
- Functions:
  - `if_else()`
  - `case_when()`
  - `across()`, `if_any`, `if_all`
  - `*_join()` functions

# if\_else()

- `if_else()` allows you to create a new variable based on a condition
- Use `if_else()` within `mutate`
- General form: `ifelse(condition, value when condition is True, value when condition is False)`

# if\_else() example

```
starwars ← dplyr::starwars
starwars %>%
  # Each value in the films variable is a list.
  # Compute the length of the films variable for each character
  # If length is greater than 1, character is recurring.
  # Otherwise, they only have one appearance
  mutate(recurring = ifelse(length(films) > 1, "Yes", "No")) %>%
  select(name, recurring, films) %>%
  head(5)
#> # A tibble: 5 x 3
#>   name          recurring films
#>   <chr>         <chr>    <list>
#> 1 Luke Skywalker Yes      <chr [5]>
#> 2 C-3PO        Yes      <chr [6]>
#> 3 R2-D2         Yes      <chr [7]>
#> 4 Darth Vader  Yes      <chr [4]>
#> 5 Leia Organa  Yes      <chr [5]>
```

# case\_when()

- `case_when()` allows you to create a new variable based on many conditions
- Think of `case_when()` as a generalized version of `if_else`
- Use `case_when()` within `mutate`
- The conditions of `case_when()` are sequential - ie: if your data is matched by condition 1 and 2, the value of the new variable will be the value specified for condition 1 since condition 1 was first

```
case_when(new_variable =  
  condition 1 ~ value when condition 1 is True,  
  condition 2 ~ value when condition 2 is True,  
  more conditions .....,  
  # if data doesn't match any of the above conditions,  
  assign the value to NA (missing)  
  TRUE ~ NA)
```

# case\_when() example

```
starwars %>%
  mutate(tallness = case_when(
    height > 180 ~ "Tall",
    height ≥ 150 & height < 180 ~ "Average",
    height < 150 ~ "Short",
    TRUE ~ NA_character_)
  ) %>%
  select(name, height, tallness) %>%
  head(5)
#> # A tibble: 5 x 3
#>   name          height tallness
#>   <chr>         <int> <chr>
#> 1 Luke Skywalker    172 Average
#> 2 C-3PO             167 Average
#> 3 R2-D2              96 Short
#> 4 Darth Vader      202 Tall
#> 5 Leia Organa      150 Average
```

# across()

- `across()` allows you to apply a function to multiple variables
- `across()` uses select helper functions to help you select variables efficiently
- `across()` reduces redundancy for similar operations performed to multiple variables

# across() example

- How to add a curve to all three test scores in one line of code? Use across

```
# test scores
tests <- data.frame(test1 = round(runif(n=50, min = 0, max = 100)),
                    test2 = round(runif(n=50, min = 0, max = 100)),
                    test3 = round(runif(n=50, min = 0, max = 100))
)

tests %>% head(10)
```

	test1	test2	test3
#> 1	33	29	32
#> 2	45	51	67
#> 3	11	29	84
#> 4	82	21	82
#> 5	78	56	92
#> 6	50	83	94
#> 7	50	19	79
#> 8	2	89	11
#> 9	95	31	72
#> 10	78	25	48



# across() example

```
tests %>%  
  # everything() selects all columns  
  # .x refers to each individual column  
  # for each column, add 10 (points)  
  mutate(across(.cols = everything(),  
                .fns = ~ .x + 10)) %>%  
  head(10)  
#>      test1 test2 test3  
#> 1      43    39    42  
#> 2      55    61    77  
#> 3      21    39    94  
#> 4      92    31    92  
#> 5      88    66   102  
#> 6      60    93   104  
#> 7      60    29    89  
#> 8      12    99    21  
#> 9     105    41    82  
#> 10     88    35    58
```

# across() example

```
starwars %>%  
  # use logical expressions inside where()  
  # .x refers to each individual character column  
  # for each column, capitalize the text  
  mutate(across(.cols = where(is.character),  
                 .fns = ~ toupper(.x))) %>%  
  select(where(is.character)) %>%  
  head(10)  
#> # A tibble: 10 x 8  
#>   name          hair_color skin_color eye_color sex      gender  
#>   <chr>         <chr>      <chr>    <chr>    <chr>  <chr>  
#> 1 LUKE SKYWALKER BLOND      FAIR     BLUE    MALE   MASCULI~  
#> 2 C-3PO         <NA>      GOLD     YELLOW  NONE   MASCULI~  
#> 3 R2-D2         <NA>      WHITE, BLUE RED     NONE   MASCULI~  
#> 4 DARTH VADER   NONE      WHITE     YELLOW  MALE   MASCULI~  
#> 5 LEIA ORGANA   BROWN     LIGHT     BROWN   FEMALE FEMININE  
#> 6 OWEN LARS     BROWN, GREY LIGHT     BLUE    MALE   MASCULI~  
#> 7 BERU WHITESUN LARS BROWN     LIGHT     BLUE    FEMALE FEMININE  
#> 8 R5-D4         <NA>      WHITE, RED RED     NONE   MASCULI~  
#> 9 BIGGS DARKLIGHTER BLACK      LIGHT     BROWN   MALE   MASCULI~  
#> 10 OBI-WAN KENOBI AUBURN, WHITE FAIR     BLUE-GRAY MALE   MASCULI~
```

# **if\_any(), if\_all()**

- `if_any()` allows you to filter data if at least of the selected columns selected meet a condition
- `if_all()` allows you to filter data if all the selected columns meet a condition

# if\_any() example

```
tests %>%  
  # only show rows with students who scored above 50  
  # on ANY of the three tests  
  filter(if_any(.cols = starts_with("test"), .funs = ~ .x > 50)) %>%  
  head(10)  
#>      test1 test2 test3  
#> 1      33    29    32  
#> 2      45    51    67  
#> 3      11    29    84  
#> 4      82    21    82  
#> 5      78    56    92  
#> 6      50    83    94  
#> 7      50    19    79  
#> 8       2    89    11  
#> 9      95    31    72  
#> 10     78    25    48
```

# if\_all() example

```
tests %>%  
  # only show rows with students who scored above 50  
  # on ALL three tests  
  filter(if_all(.cols = starts_with("test"), .fns = ~ . > 50))  
#>   test1 test2 test3  
#> 1    78    56    92  
#> 2    95    88    81  
#> 3    66    81    64  
#> 4    93    60    98  
#> 5    90    88    89  
#> 6    88    99    67
```

# join() functions

- \*\_join() functions allows you to combine multiple data frames (tables) together
- There are many different types of joins: left, right, inner, full, semi, anti
- We can link tables together with **key variables**. **Key variables** uniquely identify an observation.

# Join examples

```
students <- data.frame(
  id = c(1, 3, 4, 5, 8),
  name = c("Anthony", "Beatrice", "Claudia", "Dan", "Eliza")
)
classes <- data.frame(
  student_id = c(1, 1, 2, 3, 4, 4),
  class = c("English", "Math", "Science", "History", "Math", "History")
)
students
#>   id    name
#> 1  1 Anthony
#> 2  3 Beatrice
#> 3  4  Claudia
#> 4  5      Dan
#> 5  8     Eliza
classes
#>   student_id  class
#> 1           1 English
#> 2           1   Math
#> 3           2 Science
#> 4           3 History
#> 5           4   Math
#> 6           4 History
```

# Left Join

- Keep all rows in the left table regardless of a match

```
students %>%  
  # supply key variables in the by argument  
  # notice how the names of the key variables do not have to be the same  
  left_join(classes, by = c("id" = "student_id"), keep=TRUE)  
#>   id   name student_id  class  
#> 1  1 Anthony         1 English  
#> 2  1 Anthony         1    Math  
#> 3  3 Beatrice        3  History  
#> 4  4 Claudia         4    Math  
#> 5  4 Claudia         4  History  
#> 6  5      Dan        NA    <NA>  
#> 7  8     Eliza        NA    <NA>  
  
# keep all rows in the "left" table (students)  
  
# the class variable from the "right" table (classes) is appended  
# rows are repeated for every match from the "right" table  
# when there is no match in id, a missing value is returned
```



# Right Join

- Keep all rows in the right table regardless of matches

```
students %>%  
  right_join(classes, by = c("id" = "student_id"), keep=TRUE)  
#>   id      name student_id  class  
#> 1  1 Anthony           1 English  
#> 2  1 Anthony           1    Math  
#> 3  3 Beatrice          3 History  
#> 4  4 Claudia           4    Math  
#> 5  4 Claudia           4 History  
#> 6 NA      <NA>          2 Science  
  
# keep all rows in the "right" table (classes) are kept  
  
# left_join(students, classes, by = c("id" = "student_id")) is equivalent to  
# right_join(classes, students, by = c("student_id" = "id"))
```

# Inner Join

- Keep only rows with matches

```
students %>%  
  inner_join(classes, by = c("id" = "student_id"), keep=TRUE)  
#>   id    name student_id  class  
#> 1  1 Anthony         1 English  
#> 2  1 Anthony         1   Math  
#> 3  3 Beatrice        3 History  
#> 4  4 Claudia         4   Math  
#> 5  4 Claudia         4 History  
  
# only keep rows with matches in both tables  
# in other words, only ids 1, 3, and 4 are found in both tables
```

# Full Join

- Keep all rows from both tables regardless of matches

```
students %>%  
  full_join(classes, by = c("id" = "student_id"), keep=TRUE)  
#>   id      name student_id  class  
#> 1 1 Anthony         1 English  
#> 2 1 Anthony         1   Math  
#> 3 3 Beatrice        3 History  
#> 4 4 Claudia         4   Math  
#> 5 4 Claudia         4 History  
#> 6 5      Dan        NA    <NA>  
#> 7 8     Eliza        NA    <NA>  
#> 8 NA    <NA>         2 Science  
  
# return all rows from both tables  
# fill values with NA where there are no matches
```

# Semi Join

- Keep all rows in left table that have a match in right table
- Use the semi join to see which rows will match
- If a lot of rows are returned from the semi join, then be happy!

```
students %>%  
  semi_join(classes, by = c("id" = "student_id"), keep=TRUE)  
#>   id    name  
#> 1  1 Anthony  
#> 2  3 Beatrice  
#> 3  4 Claudia  
  
# notice that the all ids are in the classes table
```

# Anti Join

- Drop all rows in left table that have a match in right table
- Use the anti join to see which rows will not match
- If a lot of rows are returned from the anti join, investigate the cause for low matching.

```
students %>%  
  anti_join(classes, by = c("id" = "student_id"), keep=TRUE)  
#>   id  name  
#> 1  5   Dan  
#> 2  8 Eliza  
  
# notice that the ids are NOT in the classes table
```

# Issues with joining

- When joining works, you are rewarded with rich datasets.
- Be wary of data issues that could affect your join
  - Missing values in key variables
  - Data entry issues - errors, misspellings, inconsistencies

# Missing values in key variables

- id variable in df2 has a missing value.
- If we can't identify the observation, we can't join it!

```
df1 <- data.frame(id = c(1, 2, 3, 4),  
                  date = c("Jan 2020", "Jan 2020", "Mar 2020", "Mar 2020"))  
  
df2 <- data.frame(id = c(1, 2, NA, 5),  
                  sex = c("male", "female", "male", "female"))
```

# Data entry issues

- Key variables need to be identical for accurate row matching
- Data manipulation is required to make the name variable in covid\_test consistent with the name variable in patients

```
patients ←  
  data.frame(id = c(1:5),  
             name = c("Anthony", "Rick",  
                      "Claudia", "Dan", "John"))  
  
covid_tests ← data.frame(name = c("Anthony",  
                                  "RICKY",  
                                  "Johnny",  
                                  "Claudia C",  
                                  "Daniel"),  
                          covid_tested = c("Yes", "No", "Yes",  
                                             "Yes", "No"))
```



# Summary

- Create new variables based on conditions with `if_else()` and `case_when()`
- Apply functions to multiple variables with `across()`
- Filter data if all or any selected variables meet the conditions with `if_any()` and `if_all()`
- Combine data from different tables with `*_join()` functions