11 How to filter rows based on multiple columns

> **What will this tutorial cover?**
>
> In this tutorial you will learn how to use the `if_any` and `if_all` functions to filter rows based on conditions across multiple columns. We will deal with three use cases: Filtering rows based on specific conditions, filtering rows with missing values and creating new columns with `case_when` and `if_any` / `if_all`.

> **Who do I have to thank?**
>
> I would like to thank the authors of the book R for Epidemiology, from whom I took the example of showing `filter` only with logical vectors. I would also like to thank Romain Francois for his great Tidyverse blog post about the `if_any` and `if_all` functions.

Suppose you want to filter all rows from your data frame that contain a missing value. If you have only a few columns, you can solve this problem as follows:

```
df <- tibble(
  a = c(1, 2, 3),
  b = c(NA, 4, 8),
  c = c(1, 4, 1)
)

df %>%
  filter(!is.na(a) & !is.na(b) & !is.na(c))
```

```
# A tibble: 2 x 3
      a     b     c
  <dbl> <dbl> <dbl>
1     2     4     4
2     3     8     1
```

The function `filter` basically says the following: Find the rows where neither a, nor b, nor c is a missing value. To accomplish this, we repeat the code `!is.na` three times. It doesn't take much to imagine how error-prone this approach would be when you have dozens of rows.

To scale the solution to this problem, we need a way to automatically check a condition across multiple columns. It's no surprise that I say `across` here, because the functions we'll learn in a minute are basically a derivative of the `across` function from the previous tutorial. The two functions you will get to know in this tutorial are `if_any` and `if_all`.

What both functions have in common is that they produce `TRUE` or `FALSE` values. Let us have another look at `filter`. `filter` basically works with logical vectors. For example, suppose we want to remove the first and third rows of your data frame with logical values only:

```
df %>%
  filter(c(FALSE, TRUE, FALSE))
```

```
# A tibble: 1 x 3
      a     b     c
  <dbl> <dbl> <dbl>
1     2     4     4
```

When working with `filter`, these logical values are usually generated by checking a condition for a single column:

```
df %>%
  filter(a == 2)
```

```
# A tibble: 1 x 3
      a     b     c
  <dbl> <dbl> <dbl>
1     2     4     4
```

The trick with `if_any` and `if_all` is that they check a condition across multiple columns and return a `TRUE` or `FALSE` value for each row. Here is the difference between them:

- `if_any` indicates whether one of the selected columns fulfills a condition
- `if_all` indicates whether all selected columns satisfy a condition

The structure of these two functions is very similar to the `across` function. The following structure applies to both `if_any` and `if_all`:

```
<DFRAME> %>%
  filter(
    if_any(
      .cols  = <SELECTION OF COLUMNS>,
      .fns   = <CONDITION TO BE CHECKED FOR EACH COLUMN>,
    )
  )
```

Both functions were introduced in dplyr in February 2021. One reason for their introduction is that `across` was not feasible with `filter`.

Next, we will dive into three use cases of `if_any` and `if_all`.

11.1 How to filter rows based on a condition across mulitple columns

Suppose you are working with the `billboard` data frame, which contains the rankings of songs over a period of 76 weeks. The columns "wk1" to "wk76" contain the rankings for each week. Let's further assume that you want to filter out the songs that made it to #1 for at least one week. Here is how you would do this with `if_any`:

```
billboard %>%
  filter(
    if_any(
      .cols = contains("wk"),
      .fns  = ~ . == 1
      )
  )
```

```
# A tibble: 17 x 79
   artist     track date.entered   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8
   <chr>      <chr> <date>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 Aaliyah    Try ~ 2000-03-18      59    53    38    28    21    18    16    14
 2 Aguilera,~ Come~ 2000-08-05      57    47    45    29    23    18    11     9
 3 Aguilera,~ What~ 1999-11-27      71    51    28    18    13    13    11     1
 4 Carey, Ma~ Than~ 1999-12-11      82    68    50    50    41    37    26    22
 5 Creed      With~ 2000-05-13      84    78    76    74    70    68    74    75
 6 Destiny's~ Inde~ 2000-09-23      78    63    49    33    23    15     7     5
 7 Destiny's~ Say ~ 1999-12-25      83    83    44    38    16    13    16    16
 8 Iglesias,~ Be W~ 2000-04-01      63    45    34    23    17    12     9     8
```

```
 9 Janet      Does~ 2000-06-17      59    52    43    30    29    22    15    10
10 Lonestar   Amaz~ 1999-06-05      81    54    44    39    38    33    29    29
11 Madonna    Music 2000-08-12      41    23    18    14     2     1     1     1
12 N'Sync     It's~ 2000-05-06      82    70    51    39    26    19    15     9
13 Santana    Mari~ 2000-02-12      15     8     6     5     2     3     2     2
14 Savage Ga~ I Kn~ 1999-10-23      71    48    43    31    20    13     7     6
15 Sisqo      Inco~ 2000-06-24      77    66    61    61    61    55     2     1
16 Vertical ~ Ever~ 2000-01-22      70    61    53    46    40    33    31    26
17 matchbox ~ Bent  2000-04-29      60    37    29    24    22    21    18    16
# ... with 68 more variables: wk9 <dbl>, wk10 <dbl>, wk11 <dbl>, wk12 <dbl>,
#   wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>, wk18 <dbl>,
#   wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>, wk24 <dbl>,
#   wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>, wk30 <dbl>,
#   wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>, wk35 <dbl>, wk36 <dbl>,
#   wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>, wk41 <dbl>, wk42 <dbl>,
#   wk43 <dbl>, wk44 <dbl>, wk45 <dbl>, wk46 <dbl>, wk47 <dbl>, wk48 <dbl>, ...
```

A total of 17 songs made it to #1. If you look at the `.cols` argument again, you will notice that, similar to `across`, we can use tidyselect functions to select columns for which we want to check the condition.

Let's swap `if_any` with `if_all` and see what happens:

```
billboard %>%
  filter(
    if_all(
      .cols = contains("wk"),
      .fns  = ~ . == 1
      )
  )
```

```
# A tibble: 0 x 79
# ... with 79 variables: artist <chr>, track <chr>, date.entered <date>,
#   wk1 <dbl>, wk2 <dbl>, wk3 <dbl>, wk4 <dbl>, wk5 <dbl>, wk6 <dbl>,
#   wk7 <dbl>, wk8 <dbl>, wk9 <dbl>, wk10 <dbl>, wk11 <dbl>, wk12 <dbl>,
#   wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>, wk18 <dbl>,
#   wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>, wk24 <dbl>,
#   wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>, wk30 <dbl>,
#   wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>, wk35 <dbl>, wk36 <dbl>, ...
```

We get an empty data frame. That's because there is no song that has stayed at #1 for more than 76 weeks. In fact, most songs jumped out of the Top 100 soon after their release, leading to `NA`s in our data frame.

Here is a smarter way to use `if_all`. Let's say you want to filter those songs that stayed in the Top 50 for the first five weeks:

```
billboard %>%
  filter(
    if_all(
      .cols = matches("wk[1-5]$"),
      .fns  = ~ . <= 50
      )
    )
```

```
# A tibble: 13 x 79
   artist     track date.entered   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8
   <chr>      <chr> <date>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 "Aguilera~ I Tu~ 2000-04-15      50    39    30    28    21    19    20    17
 2 "Backstre~ Shap~ 2000-10-14      39    25    24    15    12    12    10     9
 3 "Dixie Ch~ Good~ 2000-03-18      40    29    24    24    20    20    20    19
 4 "Elliott,~ Hot ~ 1999-11-27      36    21    13     9     7     7     5     7
 5 "Guy"      Danc~ 1999-12-18      46    29    19    22    36    44    58    58
 6 "Lil Bow ~ Boun~ 2000-08-19      48    35    24    24    20    20    20    20
 7 "Madonna"  Amer~ 2000-02-19      43    35    29    29    33    32    40    58
 8 "Madonna"  Music 2000-08-12      41    23    18    14     2     1     1     1
 9 "Martin, ~ She ~ 2000-10-07      38    28    21    21    18    16    13    13
10 "N'Sync"   Bye ~ 2000-01-29      42    20    19    14    13     7     6     5
11 "No Doubt" Simp~ 2000-07-01      50    40    39    38    38    48    52    55
12 "Pink"     Ther~ 2000-03-04      25    15    12    11    11     7     7    12
13 "Santana"  Mari~ 2000-02-12      15     8     6     5     2     3     2     2
# ... with 68 more variables: wk9 <dbl>, wk10 <dbl>, wk11 <dbl>, wk12 <dbl>,
#   wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>, wk18 <dbl>,
#   wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>, wk24 <dbl>,
#   wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>, wk30 <dbl>,
#   wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>, wk35 <dbl>, wk36 <dbl>,
#   wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>, wk41 <dbl>, wk42 <dbl>,
#   wk43 <dbl>, wk44 <dbl>, wk45 <dbl>, wk46 <dbl>, wk47 <dbl>, wk48 <dbl>, ...
```

First we used the function `matches` to check the condition in the columns "wk1" to "wk5". Then we defined the condition itself, which searches for values less than or similar to 50.

11.2 How to filter rows that contain missing values

Another very useful use case is filtering rows based on missing values across multiple columns. The next data frame contains three missing values.

```
# This data frame comes from the tidyr documentation:
# https://tidyr.tidyverse.org/reference/complete.html
(df <- tibble(
  item_name = c("a", "a", "b", "b"),
  group     = c(1, NA,  1, 2),
  value1    = c(1, NA, 3, 4),
  value2    = c(4, 5,  NA, 7)
))
```

```
# A tibble: 4 x 4
  item_name group value1 value2
  <chr>     <dbl>  <dbl>  <dbl>
1 a             1      1      4
2 a            NA     NA      5
3 b             1      3     NA
4 b             2      4      7
```

Now lets keep all rows whose numeric columns do not contain missing value:

```
df %>%
  filter(
    if_all(
      .cols = where(is.numeric),
      .fns  = ~ !is.na(.)
    )
  )
```

```
# A tibble: 2 x 4
  item_name group value1 value2
  <chr>     <dbl>  <dbl>  <dbl>
1 a             1      1      4
2 b             2      4      7
```

This leaves us with two rows.

11.3 How to create new columns based conditions across multiple columns

The last example of `if_any` and `if_all` works with `mutate` instead of `filter`. It turns out that we can combine `case_when` with `if_any` / `if_all` to create a new column based on multiple column-spanning conditions. Suppose we want to create a new column that shows whether a song was #1 in the 76 weeks:

```
billboard %>%
  mutate(
    top_song = case_when(
      if_any(
        .cols = contains("wk"),
        .fns  = ~ . == 1
        ) ~ "top song",
      TRUE ~ "no top song"
    )
  ) %>%
  select(artist, track, top_song)
```

```
# A tibble: 317 x 3
   artist         track                   top_song
   <chr>          <chr>                   <chr>
 1 2 Pac          Baby Don't Cry (Keep... no top song
 2 2Ge+her        The Hardest Part Of ... no top song
 3 3 Doors Down   Kryptonite              no top song
 4 3 Doors Down   Loser                   no top song
 5 504 Boyz       Wobble Wobble           no top song
 6 98^0           Give Me Just One Nig... no top song
 7 A*Teens        Dancing Queen           no top song
 8 Aaliyah        I Don't Wanna           no top song
 9 Aaliyah        Try Again               top song
10 Adams, Yolanda Open My Heart           no top song
# ... with 307 more rows
```

This works because the left side of the two-sided `case_when` formulas expects a logical value (`<LOGICAL VALUE> == <RIGHT HAND SIDE>`).

Summary

Here's what you can take away from this tutorial.

- `if_any` and `if_all` are similar to `across`, but are usually used with `filter` and `mutate`
- Both functions have a similar structure to `across`. The only significant difference is that the `.fns` argument checks a condition rather than (re)calculating values.
- Use `if_any` if you want to check whether the condition was met by at least one of the selected columns for a given row; use `if_all` if you want to check whether the condition was met by all the selected columns for a given row.