4 How to rename many column names at once

> **What will this tutorial cover?**
>
> In this tutorial you will learn how to rename many columns with the `rename_with` function. With `rename_with` we can apply functions to specific column names which results in more elegant and error-free code than `rename`.

> **Who do I have to thank?**
>
> For this trick, I referred on the official documentation of dplyr.

Column names often contain spaces, special characters, or are written in a mixture of lower and upper case characters. Such poorly formatted column names can lead to numerous problems. Let's take these column names:

```
Age, Item 1, Item 2, $dollars, Where do you come from?
23 , 4     , 8     , 45      , "Germany"
```

- The column names `Item 1` and `Item 2` contain spaces. If we had to refer to these columns in R, we would have to enclose them in single quotes: `Item 1`. This is much more complicated to write than simply `item_1`. The same thing happens with the last column. Here the column name is not just one word, but a whole sentence. This is not uncommon, especially when a CSV file is generated directly from a survey program or an Excel file.
- The column `$dollars` starts with a `$` sign. Column names, however, cannot start with a special character in R. We would have to remove the dollar sign from the column name.
- The first letter of some column names are written in upper case. This mustn't be a problem, but I find that it is generally easier to write column names in lower case.
- We also see a pattern. The column names `Item 1` and `Item 2` differ only by their number. So it would be nice if we could rename these column names at once.

We could easily solve these issues with the 'rename" function but this approach does not scale:

```
the_data_frame %>%
  rename(
    age     = Age,
    item_1  = `Item 1`,
    item_2  = `Item 2`,
    dollars = `$dollars`,
    origin  = `Where do you come from?`
  )
```

Imagine how long it would take if you had 50 column names to change using the `rename` function. So obviously we need a better solution. And the solution is the `rename_with` function.

4.1 The `rename_with` function

The main difference between `remain` and `remain_with` is that `remain_with` changes the column names using a function. The three main arguments of the function are `.data`, `.fn` and `.cols`. `.data` stands for the data frame, `.fn` for the function to apply to the column names, and `.cols` for the columns to apply the function to.

```
mpg %>%
  rename_with(data  = .,
              .fn   = toupper,
              .cols = everything())
```

In this example I used the function `toupper` to convert all column names to uppercase. Here's a little trick to better understand what is happening here: Just imagine you want to replace or change strings in a vector of strings:

```
c("manufacturer", "model", "displ", "year") %>%
  toupper
```

```
[1] "MANUFACTURER" "MODEL"        "DISPL"        "YEAR"
```

Essentially, `rename_with` does nothing but use the column names as a vector of strings:

```
mpg %>%
  rename_with(.fn = toupper,
              .cols = everything()) %>%
```

```
  colnames()
```

```
 [1] "MANUFACTURER" "MODEL"        "DISPL"        "YEAR"         "CYL"
 [6] "TRANS"        "DRV"          "CTY"          "HWY"          "FL"
[11] "CLASS"
```

Let's look at another example. In our first tutorial, we learned how to use a specific naming convention for column names using the `make_clean_names` function from the janitor package. Since this is a function that works with a vector of strings, we can use this function to write all column names of the iris data frame in BigCamel notation:

```
# Make sure to install janitor first:
#   install.packages("janitor")
iris %>%
  rename_with(~ janitor::make_clean_names(., case = "big_camel")) %>%
  colnames()
```

```
[1] "SepalLength" "SepalWidth"  "PetalLength" "PetalWidth"  "Species"
```

From this example, we can see a few things:

- I did not explicitly mention the `.data` and `.cols` arguments. Since I use the pipe, I don't need to mention the `.data` argument explicitly. Also, if I don't mention the `.cols` argument explicitly, the function is applied to all column names.
- In this example, I used the tilde operator to indicate an anonymous function. This shortcut is needed whenever you need to call certain arguments of a function. In our case, this is the `case` argument of `make_clean_names`.

In summary, we can use the `toupper`, `tolower`, and `make_clean_names` functions to convert the column names to lowercase or uppercase, or to convert our column names to a particular naming convention.

4.2 How to use `rename_with` to replace characters

Another use case of `rename_with` is the replacement of characters. Suppose we want to replace all `e` characters with an underscore `_`:

```
mpg %>%
  rename_with(~ gsub("e", "_", .)) %>%
```

```
  colnames()
```

```
 [1] "manufactur_r" "mod_l"        "displ"        "y_ar"         "cyl"
 [6] "trans"        "drv"          "cty"          "hwy"          "fl"
[11] "class"
```

As you can see, I used the `gsub` function to replace a specific character. Alternatively, I could have used the str_replace function:

```
mpg %>%
  rename_with(~ str_replace(., "e", "_")) %>%
  colnames()
```

```
 [1] "manufactur_r" "mod_l"        "displ"        "y_ar"         "cyl"
 [6] "trans"        "drv"          "cty"          "hwy"          "fl"
[11] "class"
```

Let's look at a slightly more complicated example. The column names of the anscombe dataset consist of the letter x or y and the numbers 1 to 4:

```
anscombe %>%
  colnames()
```

```
[1] "x1" "x2" "x3" "x4" "y1" "y2" "y3" "y4"
```

Suppose we want to insert an underscore between the letter and the number. A trick to solve this problem is to use the grouping function in `str_replace`. A group in the argument `pattern` is everything between two brackets.

```
anscombe %>%
  rename_with(~ str_replace(., pattern = "(\\d+)",
                            replacement = "_\\1")) %>%
  colnames()
```

```
[1] "x_1" "x_2" "x_3" "x_4" "y_1" "y_2" "y_3" "y_4"
```

Let's unpack this code. With `pattern` we said that we are looking for a group of characters containing one or more digits (`\\d+`). `\\d+` is a regular expression. If you want to learn more about it, take a look at the official documentation of stringr. With `replacement` we said that we want to put an underscore in front of this group. The group itself is specified by `\\1`. If we had two groups, the second group would be specified by `\\2`.

As you can see in the output, we have added an underscore between the letter and the number. Let's create a more complicated example with two groups. Suppose we want to replace only the column names `y1` and `y2` with `ypsilon1_` and `ypsilon2_`. If we only use the techniques we have learned so far, we need to create two groups. The first group is used to replace the `y` with `ypsilon`. The second group is used to replace the number with the number and an underscore:

```
anscombe %>%
  rename_with(~ str_replace(., "(y)([1-2])",
                            "\\1psilon\\2_")) %>%
  colnames()
```

```
[1] "x1"        "x2"        "x3"        "x4"        "ypsilon1_" "ypsilon2_"
[7] "y3"        "y4"
```

As you can see, we renamed only two of the eight columns. This is because the pattern used in `str_replace` only applies to two variables: The variables that start with a `y` and are followed by the numbers 1 or 2. You can also see that we used two groups. The first group contains the letter `y`, the second group contains the number 1 or 2 (indicated by the square brackets `[1-2]`). In the replacement argument you will find these groups as `\\1` and `\\2`. If we would get rid of these groups in the replacement argument, they would be removed from the new column names (and it would throw an error because the column names would not be unique anymore).

4.3 How to rename variables for specific variables

Granted, shows the power of `rename_with`, but it is not the most elegant. So far we haven't talked about the third argument of `rename_with`: `.cols`. You can use `.cols` to specify which column names to apply the function to. And you can even use our tidyselect functions for that. So let's rewrite our previous example with the `.cols` argument:

```
anscombe %>%
  rename_with(~ str_replace(., "([:alpha:])([1-2])",
                            "\\1psilon\\2_"), c(y1, y2)) %>%
  colnames()
```

```
[1] "x1"        "x2"        "x3"        "x4"        "ypsilon1_" "ypsilon2_"
[7] "y3"        "y4"
```

Take a closer look at the end of the code. Here we have specified that the `str_replace` function should only be applied to the columns names `y1` and `y2`. Again, we have defined two groups: `([:alpha:])` and `([1-2])`. The first group searches for a single letter. The second group searches for the numbers 1 or 2. You can see that the output is the same, we just used a different method.

Similarly, you could use a tidyselect function and convert all numeric columns to uppercase:

```
mpg %>%
  rename_with(~ toupper(.), where(is.numeric)) %>%
  colnames()
```

```
 [1] "manufacturer" "model"        "DISPL"        "YEAR"         "CYL"
 [6] "trans"        "drv"          "CTY"          "HWY"          "fl"
[11] "class"
```

Or you could replace a dot with an underscore for all columns that begin with the word Sepal:

```
iris %>%
  rename_with(~ str_replace(., "\\.", "_"),
              starts_with("Sepal")) %>%
  colnames()
```

```
[1] "Sepal_Length" "Sepal_Width"  "Petal.Length" "Petal.Width"  "Species"
```

Another useful function is `matches`. With `matches`, you can search for specific patterns in your column names and apply a function to the column names that match the pattern. In the next example, we used this technique to replace the dot with an underscore in all column names that end with "Width" or "width":

```
iris %>%
  rename_with(~ str_replace(., "\\.", "_"),
              matches("[Ww]idth$")) %>%
  colnames()
```

```
[1] "Sepal.Length" "Sepal_Width"  "Petal.Length" "Petal_Width"  "Species"
```

Summary

Here is what you can take from this tutorial.

- `rename_with` allows us to change many column names using functions.
- `rename_with` has three important arguments: `.data` which stands for the data frame, `.fn` for the function to apply to the selected column names, and `.cols` which stands for the column names to apply the function to.
- The most common use cases for `rename_with` are converting column names to a specific naming convention, converting column names to lowercase or uppercase, or removing and replacing certain characters in the column names
- More complex character replacements can be achieved with the grouping function of `str_replace` in combination with the `.cols` argument of `rename_with`.