3 How to select variables with the tidyselect functions

What will this tutorial cover?

In this tutorial you will get an introduction to the tidyselect functions. These functions give you many options to select columns in R.

Who do I have to thank?

For this tutorial I referred to the official dplyr documentation of tidyselect. A big thanks to Lionel Henry, who was instrumental in implementing these functions.

Selecting columns sounds fair and easy. But could you solve the following problems spontaneously in R?

- Select all columns of a data frame that are numeric
- Select all numeric columns that do not contain the numbers 1 or 2
- Select all columns that start with the string "wk"
- Select all columns that contain a number

If not, this tutorial might be of interest to you. We will cover a range of functions from the tidyselect package. The tidyselect package was designed specifically for these use cases and makes solving these problems much easier.

The following tricks are actually not too complicated. But we need them for the upcoming tutorials. So in this tutorial, we will go through these functions in detail:

- everything()
- last_col()
- starts_with()
- ends_with()
- contains()
- matches()
- num_range()
- where()

A common feature of all functions is that they allow you to select columns. Later in this course you will use them not only inside `select`, but also inside `summarise`, or `mutate.` Remember that any column selection can be negated with an exclamation mark `!`. I'll sprinkle in a few examples of negated column selections, just remember that negation can be applied to any of these functions. The most complicated function will be `matches` because it works with regular expressions. If you are familiar with regular expressions, you should have no problems with the examples. If not, you can read more about regular expressions on the official stringr website. I'll try to explain the examples as good as I can.

3.1 Selecting every column

The first function is `everything`. As the name suggests, it lets you select all columns of a data frame:

```
mpg %>%
  select(everything()) %>%
  glimpse()
```

```
Rows: 234
Columns: 11
$ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "audi", "audi", "~
$ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4", "a4 quattro", "~
$ displ        <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1.8, 2.0, 2.0, 2.~
$ year         <int> 1999, 1999, 2008, 2008, 1999, 1999, 2008, 1999, 1999, 200~
$ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 8, 8, ~
$ trans        <chr> "auto(l5)", "manual(m5)", "manual(m6)", "auto(av)", "auto~
$ drv          <chr> "f", "f", "f", "f", "f", "f", "f", "4", "4", "4", "4", "4~
$ cty          <int> 18, 21, 20, 21, 16, 18, 18, 18, 16, 20, 19, 15, 17, 17, 1~
$ hwy          <int> 29, 29, 31, 30, 26, 26, 27, 26, 25, 28, 27, 25, 25, 25, 2~
$ fl           <chr> "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p~
$ class        <chr> "compact", "compact", "compact", "compact", "compact", "c~
```

You will rightly ask yourself why you need such a function. One use case is relocating columns with `everything`. For example, we could move the column `cyl` and manufacturer to the beginning of the data frame:

```
mpg %>%
  select(manufacturer, cyl, everything()) %>%
  glimpse()
```

```
Rows: 234
Columns: 11
$ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "audi", "audi", "~
$ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 8, 8, ~
$ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4", "a4 quattro", "~
$ displ        <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1.8, 2.0, 2.0, 2.~
$ year         <int> 1999, 1999, 2008, 2008, 1999, 1999, 2008, 1999, 1999, 200~
$ trans        <chr> "auto(l5)", "manual(m5)", "manual(m6)", "auto(av)", "auto~
$ drv          <chr> "f", "f", "f", "f", "f", "f", "f", "4", "4", "4", "4", "4~
$ cty          <int> 18, 21, 20, 21, 16, 18, 18, 18, 16, 20, 19, 15, 17, 17, 1~
$ hwy          <int> 29, 29, 31, 30, 26, 26, 27, 26, 25, 28, 27, 25, 25, 25, 2~
$ fl           <chr> "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p~
$ class        <chr> "compact", "compact", "compact", "compact", "compact", "c~
```

Other than that, I haven't seen any other use cases for `everything`.

3.2 Selecting the last column

Then we have `last_col`. With this function you can select the last column in a data frame:

```
mpg %>%
  select(last_col()) %>%
  glimpse()
```

```
Rows: 234
Columns: 1
$ class <chr> "compact", "compact", "compact", "compact", "compact", "compact"~
```

I have already told you that you can also negate a selection of columns. So let's then try to select all columns except the last one:

```
mpg %>%
  select(!last_col()) %>%
  glimpse()
```

```
Rows: 234
Columns: 10
$ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "audi", "audi", "~
$ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4", "a4 quattro", "~
$ displ        <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1.8, 2.0, 2.0, 2.~
$ year         <int> 1999, 1999, 2008, 2008, 1999, 1999, 2008, 1999, 1999, 200~
$ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 8, 8, ~
$ trans        <chr> "auto(l5)", "manual(m5)", "manual(m6)", "auto(av)", "auto~
$ drv          <chr> "f", "f", "f", "f", "f", "f", "f", "4", "4", "4", "4", "4~
$ cty          <int> 18, 21, 20, 21, 16, 18, 18, 18, 16, 20, 19, 15, 17, 17, 1~
$ hwy          <int> 29, 29, 31, 30, 26, 26, 27, 26, 25, 28, 27, 25, 25, 25, 2~
$ fl           <chr> "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p~
```

Also, you can use `last_col` to select the n-to-last column:

```
mpg %>%
  select(last_col(1)) %>%
  glimpse
```

```
Rows: 234
Columns: 1
$ fl <chr> "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p~
```

Not that the index starts with 0, so the number 1 indicates the second to last column.

3.3 Selecting columns that start and end with a specific string

Next we have the two functions `starts_with` and `ends_with`. You use these functions when you want to select columns that start or end with exactly a certain string. We could use `starts_with` to select all columns that start with the letter "m":

```
mpg %>%
  select(starts_with("m")) %>%
  glimpse()
```

```
Rows: 234
Columns: 2
$ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "audi", "audi", "~
$ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4", "a4 quattro", "~
```

`starts_with` and `ends_with` works with any character, but also with a vector of characters. Suppose we want to select columns ending with the letter l or r:

```r
mpg %>%
  select(ends_with(c("l", "r"))) %>%
  glimpse()
```

```
Rows: 234
Columns: 6
$ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4", "a4 quattro", "~
$ displ        <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1.8, 2.0, 2.0, 2.~
$ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 8, 8, ~
$ fl           <chr> "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p~
$ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "audi", "audi", "~
$ year         <int> 1999, 1999, 2008, 2008, 1999, 1999, 2008, 1999, 1999, 200~
```

Of course, the character string can be longer than just one character:

```r
billboard %>%
  select(starts_with("wk")) %>%
  colnames()
```

```
 [1] "wk1"  "wk2"  "wk3"  "wk4"  "wk5"  "wk6"  "wk7"  "wk8"  "wk9"  "wk10"
[11] "wk11" "wk12" "wk13" "wk14" "wk15" "wk16" "wk17" "wk18" "wk19" "wk20"
[21] "wk21" "wk22" "wk23" "wk24" "wk25" "wk26" "wk27" "wk28" "wk29" "wk30"
[31] "wk31" "wk32" "wk33" "wk34" "wk35" "wk36" "wk37" "wk38" "wk39" "wk40"
[41] "wk41" "wk42" "wk43" "wk44" "wk45" "wk46" "wk47" "wk48" "wk49" "wk50"
[51] "wk51" "wk52" "wk53" "wk54" "wk55" "wk56" "wk57" "wk58" "wk59" "wk60"
[61] "wk61" "wk62" "wk63" "wk64" "wk65" "wk66" "wk67" "wk68" "wk69" "wk70"
[71] "wk71" "wk72" "wk73" "wk74" "wk75" "wk76"
```

3.4 Selecting columns that contain certain strings

Next we have the `contains` function. `contains` searches for columns that contain a specific string. Note that it does not work with regular expressions, but searches for exactly the string you specify. By default, however, the function is not case-sensitive. It doesn't matter if your columns are in uppercase or lowercase. Let's select all columns that contain the letter "m":

```r
mpg %>%
  select(contains("m")) %>%
  glimpse()
```

```
Rows: 234
Columns: 2
$ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "audi", "audi", "~
$ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4", "a4 quattro", "~
```

I told you that `contains` is not case sensitive. If you are concerned about case sensitivity, set the `ignore.case` argument to `FALSE` (this also works with `starts_with`, `ends_with`, and `matches`):

```
mpg %>%
  rename(Manufacturer = manufacturer) %>%
  select(contains("m", ignore.case = FALSE)) %>%
  glimpse()
```

```
Rows: 234
Columns: 1
$ model <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4", "a4 quattro", "a4 quat~
```

3.5 Selecting columns with regular expressions

Unlike contains, `matches` works with regular expressions. As mentioned at the beginning, if you don't know too much about regular expressions, don't worry. I'll do my best to explain the code as good as I can.

Suppose you want to select all columns that contain a number. The billboard dataset is a good use case for this. The dataset contains the rankings of songs over a period of 76 weeks.

```
billboard %>%
  select(matches("\\d")) %>%
  colnames()
```

```
 [1] "wk1"  "wk2"  "wk3"  "wk4"  "wk5"  "wk6"  "wk7"  "wk8"  "wk9"  "wk10"
[11] "wk11" "wk12" "wk13" "wk14" "wk15" "wk16" "wk17" "wk18" "wk19" "wk20"
[21] "wk21" "wk22" "wk23" "wk24" "wk25" "wk26" "wk27" "wk28" "wk29" "wk30"
[31] "wk31" "wk32" "wk33" "wk34" "wk35" "wk36" "wk37" "wk38" "wk39" "wk40"
[41] "wk41" "wk42" "wk43" "wk44" "wk45" "wk46" "wk47" "wk48" "wk49" "wk50"
[51] "wk51" "wk52" "wk53" "wk54" "wk55" "wk56" "wk57" "wk58" "wk59" "wk60"
[61] "wk61" "wk62" "wk63" "wk64" "wk65" "wk66" "wk67" "wk68" "wk69" "wk70"
[71] "wk71" "wk72" "wk73" "wk74" "wk75" "wk76"
```

The regular expression `\\d` in the function stands for any digit. Similarly, one could search only for columns that begin with the string "wk" and are followed by only one digit:

```
billboard %>%
  select(matches("wk\\d{1}$")) %>%
  colnames()
```

```
[1] "wk1" "wk2" "wk3" "wk4" "wk5" "wk6" "wk7" "wk8" "wk9"
```

Here, the curly braces indicate that we are looking for only one digit, and the dollar sign `$` indicates that the column should end with that one digit. From this example it should be obvious that matches provides the most versatile way to select columns among the tidyselect functions.

Let's try another example. Imagine you want to select columns starting with the letter `x` or `y` and then followed by the digits `1` to `2`:

```
anscombe %>%
  select(matches("[xy][1-2]")) %>%
  glimpse()
```

```
Rows: 11
Columns: 4
$ x1 <dbl> 10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5
$ x2 <dbl> 10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5
$ y1 <dbl> 8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.68
$ y2 <dbl> 9.14, 8.14, 8.74, 8.77, 9.26, 8.10, 6.13, 3.10, 9.13, 7.26, 4.74
```

The square brackets are used to search for sets of characters. The first bracket indicates that we are looking for the letter `x` or `y`. If we put a hyphen `-` between the characters, we are looking for a range of values. Here we are looking for numbers between `1` and `2`.

3.6 Select columns with number ranges

Next, we have `num_range.` The function is useful if your column names follow a certain pattern. In the anscombe data frame the column names start with a letter and are then followed by a number. Let's try use `num_range` with this data frame. And let's try to find all columns that start with the letter `x` and are followed by the numbers 1 or 2:

```r
anscombe %>%
  select(num_range("x", 1:2)) %>%
  glimpse()
```

```
Rows: 11
Columns: 2
$ x1 <dbl> 10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5
$ x2 <dbl> 10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5
```

The same idea can be applied to the billboard dataset. In this dataset we have a list of columns starting with the letters “wk” (for week) and the numbers 1 to 76:

```r
billboard %>%
  select(num_range("wk", 1:15)) %>%
  glimpse()
```

```
Rows: 317
Columns: 15
$ wk1  <dbl> 87, 91, 81, 76, 57, 51, 97, 84, 59, 76, 84, 57, 50, 71, 79, 80, 9~
$ wk2  <dbl> 82, 87, 70, 76, 34, 39, 97, 62, 53, 76, 84, 47, 39, 51, 65, 78, 9~
$ wk3  <dbl> 72, 92, 68, 72, 25, 34, 96, 51, 38, 74, 75, 45, 30, 28, 53, 76, 9~
$ wk4  <dbl> 77, NA, 67, 69, 17, 26, 95, 41, 28, 69, 73, 29, 28, 18, 48, 77, 9~
$ wk5  <dbl> 87, NA, 66, 67, 17, 26, 100, 38, 21, 68, 73, 23, 21, 13, 45, 92, ~
$ wk6  <dbl> 94, NA, 57, 65, 31, 19, NA, 35, 18, 67, 69, 18, 19, 13, 36, NA, 9~
$ wk7  <dbl> 99, NA, 54, 55, 36, 2, NA, 35, 16, 61, 68, 11, 20, 11, 34, NA, 93~
$ wk8  <dbl> NA, NA, 53, 59, 49, 2, NA, 38, 14, 58, 65, 9, 17, 1, 29, NA, 96, ~
$ wk9  <dbl> NA, NA, 51, 62, 53, 3, NA, 38, 12, 57, 73, 9, 17, 1, 27, NA, NA, ~
$ wk10 <dbl> NA, NA, 51, 61, 57, 6, NA, 36, 10, 59, 83, 11, 17, 2, 30, NA, NA,~
$ wk11 <dbl> NA, NA, 51, 61, 64, 7, NA, 37, 9, 66, 92, 1, 17, 2, 36, NA, 99, N~
$ wk12 <dbl> NA, NA, 51, 59, 70, 22, NA, 37, 8, 68, NA, 1, 3, 3, 37, NA, NA, 9~
$ wk13 <dbl> NA, NA, 47, 61, 75, 29, NA, 38, 6, 61, NA, 1, 3, 3, 39, NA, 96, N~
$ wk14 <dbl> NA, NA, 44, 66, 76, 36, NA, 49, 1, 67, NA, 1, 7, 4, 49, NA, 96, N~
$ wk15 <dbl> NA, NA, 38, 72, 78, 47, NA, 61, 2, 59, NA, 4, 10, 12, 57, NA, 99,~
```

3.7 Selecting columns of a specific type

Finally, there is the `where` function. `where` is used when you want to select variables of a certain data type. For example, we could select character variables:

```r
billboard %>%
  select(where(is.character)) %>%
  glimpse()
```

```
Rows: 317
Columns: 2
$ artist <chr> "2 Pac", "2Ge+her", "3 Doors Down", "3 Doors Down", "504 Boyz",~
$ track  <chr> "Baby Don't Cry (Keep...", "The Hardest Part Of ...", "Kryptoni~
```

`where` was introduced in tidyselect in 2020 to avoid confusing error messages. So make sure that if you use a predicate function (e.g. is.character), you include it in `where`. Other predicate functions are:

- is.double
- is.logical
- is.factor
- is.integer

3.8 Combining selections

Before we conclude this tutorial, you should know that you can combine the different selection functions with the `&` and `|` operators. Suppose we want to select all columns that are of type character and that contain the letter `l`:

```r
mpg %>%
  select(where(is.character) & contains("l")) %>%
  glimpse()
```

```
Rows: 234
Columns: 3
$ model <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4", "a4 quattro", "a4 quat~
$ fl    <chr> "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p",~
$ class <chr> "compact", "compact", "compact", "compact", "compact", "compact"~
```

Similarly, we can use the or operator `|` to select columns that satisfy one of several conditions:

```r
mpg %>%
  select(where(is.character) | contains("l")) %>%
  glimpse()
```

```
Rows: 234
Columns: 8
$ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "audi", "audi", "~
$ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4", "a4 quattro", "~
$ trans        <chr> "auto(l5)", "manual(m5)", "manual(m6)", "auto(av)", "auto~
$ drv          <chr> "f", "f", "f", "f", "f", "f", "f", "4", "4", "4", "4", "4~
$ fl           <chr> "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p", "p~
$ class        <chr> "compact", "compact", "compact", "compact", "compact", "c~
$ displ        <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1.8, 2.0, 2.0, 2.~
$ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 8, 8, ~
```

Compared to our previous example, the selection results in 8 columns instead of 3 because we used the or operator.

This concludes our brief tour of the tidyselect functions. These functions are not the most exciting in this course, but they are fundamental to many things that we will cover later in this course.

Summary

Here is what you can take from this tutorial.

- The tidyselect functions provide you with versatile possibilities to select variables
- Tidyselect provides the following functions: `everything`, `last_col`, `starts_with`, `ends_with`, `contains`, `matches`, `num_range`, `where`
- All functions can be used with other functions as `select`. For example, `summarise` or `mutate`
- Any selection can be negated within select
- `everything` can be used to relocate variables
- `matches` is the most complicated function because it works with regular expressions