Filter rows

Prof. Dr. Nicolas Meseth

Besides selecting the columns we need, we also have to learn tools to restrict the rows in data frame. The means to do that with the `{dplyr}` package is the `filter` command.

The `filter` command

The `filter` command takes one or more expressions, which must evaluate to `TRUE` or `FALSE`. These types of expressions are called *boolean expressions*, named after George Boole, who invented the Boolean algebra. Every expression passed to the `filter` command is evaluated for every row in the data frame. Only if the expression returns `TRUE` for a row, this row is included in the resulting data frame.

To form expressions, we can use a number of operators and functions. This chapter introduces the basic ways to express filter conditions on our data.

Equals operator

The simplest way to filter data is to compare the value of a column to a given value. This way, we can get all orders from female customers:

```
orders %>%
  filter(customer_gender == "f")

#> filter: removed 1,613 rows (56%), 1,261 rows remaining
```

As you can see, the equals operator consists of two equal signs in a row (`==`). This is important, as using only one equals sign results in an error. A single equals sign is reserved for assignments, such as when we create a new column with `mutate`.

In the example above, the `customer_gender` column is of the data type `chr`, which means it contains alphanumeric symbols. For such columns, when comparing values, we must enclose the literal values with quotations marks. This is because the data type `chr` can contain spaces.

If we didn't use quotation marks, R wouldn't know where the string of alphanumeric character starts and ends.

The equals comparison `==` is useful mostly for discrete data types. Un R, these include strings (or `chr`), whole numbers (`integer`), dates, and factors. Data types such as decimal numbers (`double`) or datetime can in principle compared to a specific value using the comparison operator `==`, but given their continuous nature, it usually doesn't make too much sense. Artihmetic operators, such as less or greater than, are much more useful in these cases.

Arithmetic operators

```
orders %>%
  filter(total_price < 50)

#> filter: removed 633 rows (22%), 2,241 rows remaining
```

Logical combinations of filter expressions

When we list two filter expressions separated by comma, they are connected with the logical operator *and*:

```
# Customer who are female and university staff at the same time
orders %>%
  filter(customer_gender == "f", customer_is_hsos == TRUE)

#> filter: removed 2,651 rows (92%), 223 rows remaining
```

We can do that explicitly by using the official *and* operator, which is denoted by the symbol `&`.

```
# Same as above, with explicit AND symbol
orders %>%
  filter(customer_gender == "f" & customer_is_hsos == TRUE)
```

Another option is the logical *or*, which is symbolized by the `|` character:

```
# Customers who are either female or university staff (or both)
orders %>%
  filter(customer_gender == "f" | customer_is_hsos == TRUE)
```

```
#> filter: removed 1,352 rows (47%), 1,522 rows remaining
```