The tilde-shorthand

2020-11-17

During our second `tidyverse` workshop a question was raised regarding the `~` (*tilde*) operator in the context of `dplyr` and `purrr` - when does it occurs and when does not not occur? A straight forward answer to this question would be:

> Within the `tidyverse`, the `~` -shorthand likely occurs whenever an external function is required as an argument to one of the `tidyverse` functions.

Since the use of the `~` -shorthand can appear quite bizarre at first, I will try to address its use in more detail.

Altogether, we have seen several use cases of the `~` -shorthand during the `dplyr` part of the workshop:

```
penguins %>%
  mutate(across(contains("mm"), ~ . / 1000), .keep = "all")
```

```
penguins %>%
  select(where(~ is.numeric(.))) %>%
  select(where(~ mean(., na.rm = T) > 1000))
```

```
penguins %>%
  group_by(species) %>%
  summarise(
    across(contains("mm"), ~ mean(.x, na.rm = T), .names = "{.col}_avg"),
    .groups = "drop"
  )
```

These examples have in common, that one or multiple external functions are required as function arguments to `across()` respectively `where()`:

```
across(.cols, .fns, ...)
```

```
where(fn)
```

Whenever a function requires another external function as an argument to the function call, the `tidyverse` offers different ways for specifying this external function:

1. by using the name of the external function, e.g., `mean`,
2. by defining an anonymous function inline, e.g., `function(x) { mean(x) }` (note that here you could also omit `{ }`, since there is only expression that constitutes the function),
3. by defining an anonymous function inline using the `~` -shorthand (`purrr`-style), e.g., `~ { mean(.x) }` (again we could omit `{ }` as the anonymous function is a one-liner).
   Note that whenever we use the `~` -shorthand, we refer to the argument of the anonymous function by `.x` or simply `.` (if it only requires one input).

If a `tidyverse` function requires multiple external functions as an argument to the function call, the `tidyverse` demands the following approach:

4. by passing a list of named and/or anonymous functions, e.g., `list(mean = mean, mean2 = ~ mean(.x))`

Now, option 1. only ever works in cases where the external function takes only one argument, e.g., `mean(x)`:

```
penguins %>%
  mutate(across(contains("mm"), mean))  # equivalent to across(contains("mm"), ~ mean(.))
```

As soon as you specify a second argument, e.g., `mean(x, na.rm = T)` you need to rely on option 2. or 3., otherwise `R` is not recognizing it as an external function any longer:

```
penguins %>%
  mutate(across(contains("mm"), function(x) mean(x, na.rm = T)))
```

Which is in turn equivalent to:

```
penguins %>%
  mutate(across(contains("mm"), ~ mean(., na.rm = T)))
```

The same holds for the `map_*()` functions from the `purrr` package which require an external function as the second function argument (`.f`):

```
map(.x, .f, ...)
```

You may either use option 1., 2. or 3. if the external function requires only a single argument respectively option 2. or 3. if it requires multiple arguments - depending on which style your prefer. Most of the time, explicitly defining named functions and then choosing option 1. only makes sense if you require them at least more than once. Otherwise, I would strongly recommend using anonymous function, i.e. option 2. or 3.

Note: If you find any inconsistencies in the slide deck relating to these explanations or if you are still puzzled with regards to the ~ -shorthand, please reach out, either via Mail or Learnweb :)