Working With Vectorized Functions: Takeaways

Syntax

WRITING VECTORIZED IF-ELSE STATEMENTS:

- Using the if_else() function to write an if-else statement:

```
if_else(vector_1 == vector_2, "condition_1", "condition_2")
```

- Nesting if_else() functions to chain if-else statements:

```
if_else(vector_1 > vector_2, "condition_1",

    if_else(vector_1 < vector_2, "condition_2",

        if_else(vector_1 == vector_2, "condition_3", "condition_4")))
```

GROUPING DATA USING DPLYR::GROUP_BY():

- Grouping by one variable:

```
data_frame %>%

     group_by(variable)
```

- Grouping by multiple variables:

```
data_frame %>%

    group_by(variable_1, variable_2) %>%

    summarize(variable_name = function(variable_1))
```

SUMMARIZING GROUPED DATA USING DPLYR::SUMMARIZE():

- Calculating one summary:

```
data_frame %>%

    group_by(variable) %>%

    summarize(variable_name = function(variable))
```

- Calculating multiple summaries:

```
data_frame %>%

    group_by(variable_1) %>%

    summarize(variable_name_1 = function_1(variable_1), variable_name_2 =

function_2(variable_1))
```

Concepts

- In R, vectorized solutions are often faster than using loops, and the code is usually easier to understand.
- Problems that involve splitting data into groups, applying a function to each group each group, and summarizing the results are known as "split_apply-combine" problems in R.
- Chaining functions using the pipe operator allows you to write more efficient code and avoid cluttering the global environment with intermediate variables.

Resources

- Blog Post on Vectorization in R
- Wickham et al. paper on split-apply-combine problems
- maggritR package documentation