Data Transformation with dplyr 1.0 (part 2)

A guide to using (c_)across() to apply the same functions repeatedly

 @rdataberlin https://github.com/courtiol/Rguides

Use across() to apply the same function(s) on multiple columns

```
tbl %>%
    [ group_by(name_grouping_col_X, name_grouping_col_Y...) %>% ] ## grouping is optional
    verb♣(across(selected_columns, fn_with_args, [.names = prototype_for_new_col_names♠])) ## prototype for naming is optional
```

verb♣: mutate ☺ | transmute ♪ | group_by 🍎 | arrange | filter ☼ | summarize ♫ | count ♨

selected_columns:

- n+p: `name_col_X:name_col_Y` ♨
- names: `c(name_col_X, name_col_Y)` ☺ | `all_of/any_of(c("name_col_X", "name_col_Y"))` | `starts_with/ends_with/contains("partial_name_col")` ♪ | `matches("regular_expression")` 🍎 | `num_range("name_col_without_number", vector_numbers)`
- values: `where(fn_returning_TRUE/FALSE_for_each_col)` ☼
- positions: `c(position_col_X, position_col_Y)` | `everything()` ♫ | `last_col()`

♦ when `fn` runs, `.x` will be internally replaced by the content of the selected columns

fn_with_args:

- 1 function: `fn [, args*]` ☺ | `~ fn(.x♦ [, args*])` ☼
- ≥1 functions: `list(suffix1_new_col_names♠ = fn1, suffix2_new_col_names♠ = fn2), [, args*]` 🍎 | `list(suffix1_new_col_names♠ = ~ fn1(.x♦ [, args*]), suffix2_new_col_names♠ = ~ fn2(.x♦ [, args*]))` 🎼

*args = possible arguments for `fn`

A quoted expression with any text and the placeholders **{.fn}** referring to function position and/or **{.col}** ☺ referring to former column names

Only useful when no suffix defined in `fn_with_args`

♠ if no name is defined, some will automatically be generated by **dplyr**

♣ across() and c_across() should not to be used within **dplyr** verbs shown in part 1 and not here, as it would not make sense. Note however that there is a special function to rename multiple columns: `rename_with(fn_with_args, selected_columns)`

EXAMPLES for across()

```
☺ ## turning specific columns into z-scores:
  iris %>%
    mutate(across(c("Sepal.Length", "Sepal.Width"), scale, .names = "{.col}_z"))

☼ ## keeping only rows where numeric values are > 2 in all numeric columns:
  iris %>% filter(across(where(is.numeric), ~ .x > 2))

🍎 ## defining 3 groups for all columns with name made of 2 words around a point:
  iris %>% group_by(across(matches("\\w\\.\\w"), list(discrete = cut), breaks = 3))

  ## counting NA in each column:
♫ iris %>% summarise(across(everything(), list("NA" = ~ sum(is.na(.x)))))
```

(suffix for new col names; fn; args for fn)

Use c_across() to apply the same function across multiple columns within each row

```
tbl %>%
    rowwise() %>%
    verb♣(fn♥(c_across(selected_columns), [args*]))
```

Never forget! If you do, c_across() will concatenate values across rows, which is wrong

as for across(), see above

verb♣: mutate ☺ | transmute ♪ | filter ☼ | count ♨

♥ unlike `fn_with_args` in across() calls, the function `fn` is not here provided as a definition but used directly, so you can only use one function that has already been defined (you cannot define it on the fly using ~ & .x) ♪♨

EXAMPLES for c_across()

```
  ## computing the area of petal (approximated as rectangles) and only keep that:
  iris %>%
♪   rowwise() %>%
    transmute(Petal_Area = prod(c_across(contains("Petal"))))

  ## counting rows where at least one numeric values is > 6 for a range of columns:
  iris %>%
♨   rowwise() %>%
    count(any(c_across(Sepal.Length:Petal.Width) > 6))
```