NeateR

Tynan

2023-10-01

Overview

A collection of helpful R functions. Many of these function wrap code created by other people. The original sources are linked. Converting them into functions enables **easy reuse**.

The name NeateR is meant as a pun in that some of these examples both make the code cleaner while also interesting, at least to me.

Conditional Mean Difference

The idea is to bracket index a variable `variable[condition]` to get a conditional mean difference within dplyr::summarise(). This allow filtering on specific variables.

https://community.rstudio.com/t/dplyr-summarise-with-condition/100885/4.

```
data <- tribble(~Date, ~A1, ~A2,~B1,~B2,
                as.Date("2019-01-01"), 20, 10,20, 10,
                as.Date("2019-01-01"), 20 ,5,20,5,
                as.Date("2019-01-01"), 10, 2,10,20,
                as.Date("2019-01-01"), 20, 60,0,0,
                as.Date("2019-01-01"), 30, 4,20,5,
                as.Date("2019-02-01"), 0, 0,16,8,
                as.Date("2019-02-01"), 0, 0,0,40,
                as.Date("2019-02-01"), 0, 0,4,2,
                as.Date("2019-02-01"), 4, 8,10,6,
                as.Date("2019-02-01"), 6, 3,0,0,
                as.Date("2019-03-01"), 20, 8,23,9,
                as.Date("2019-03-01"), 60, 4,0,0,
                as.Date("2019-03-01"), 4, 2,8,3,
                as.Date("2019-03-01"), 0, 6,10,0)
```

The question asker had written the condition out twice for each metric being calculated.

```
data %>%
    group_by(Date) %>%
    dplyr::summarise(mean_1 = mean(A1[A1>=B1 & A1>0 & B1>0] - B1[A1>=B1 & A1>0 & B1>0]),
                     mean_2 = mean(A2[A2>=B2 & A2>0 & B2>0] - B2[A2>=B2 & A2>0 & B2>0]))
```

```
## # A tibble: 3 x 3
##   Date       mean_1 mean_2
##   <date>      <dbl>  <dbl>
## 1 2019-01-01    2.5      0
## 2 2019-02-01  NaN        2
## 3 2019-03-01  NaN      NaN
```

A more elegant solution is provided in that page below by creating two new columns based on the two conditions being used to index. Then the variables are subset using them. This code is cleaner because it writes out each condition only once, the previous code wrote each condition twice. Which increases the chance of errors.

```
data %>%
  dplyr::mutate(condition_1 = A1>=B1 & A1>0 & B1>0,
                condition_2 = A2>=B2 & A2>0 & B2>0) %>%
  group_by(Date) %>%
  dplyr::summarise(mean_1 = mean(A1[condition_1] - B1[condition_1]),
                   mean_2 = mean(A2[condition_2] - B2[condition_2]))
```

```
## # A tibble: 3 x 3
##   Date       mean_1 mean_2
##   <date>      <dbl>  <dbl>
## 1 2019-01-01    2.5      0
## 2 2019-02-01  NaN        2
## 3 2019-03-01  NaN      NaN
```

Curly curly {{}} can be used to provide both the variable and the condition. The conditional_mean_diff() function provides variables for the metrics of interest as well as the condition.

- A possible next step is to add a second condition and use an if condition to determine whether the one condition or two condition summarise() version is called.
- Or perhaps to loop over a combination of variables and conditions and then use bind_cols() to merge into one data.frame().

Note: the function is split over two lines to improve readability in Rmd output. First line makes the name for the mean diff and the second line inserts the conditions and gets the mean diff.

```
conditional_mean_diff  <- function(data, var_1, var_2, cond) {

  data %>%
    group_by(Date) %>%
    dplyr::summarise("{{var_1}}_{{cond}}_{{var_2}}_{{cond}}_mean_diff" :=

                       mean( {{var_1}} [ {{cond}} ] - {{var_2}} [ {{cond}} ] ))

}

data_w_condition <- data %>%
                    dplyr::mutate(condition_1 = A1>=B1 & A1>0 & B1>0,
                                  condition_2 = A2>=B2 & A2>0 & B2>0)

data_w_condition %>% conditional_mean_diff(., A1, B1, condition_1)
```

```
## # A tibble: 3 x 2
##   Date       A1_condition_1_B1_condition_1_mean_diff
##   <date>                                      <dbl>
## 1 2019-01-01                                    2.5
## 2 2019-02-01                                  NaN
## 3 2019-03-01                                  NaN
```

```
data_w_condition %>% conditional_mean_diff(., A2, B2, condition_2)
```

```
## # A tibble: 3 x 2
##   Date       A2_condition_2_B2_condition_2_mean_diff
```

```
##   <date>                                          <dbl>
## 1 2019-01-01                                        0
## 2 2019-02-01                                        2
## 3 2019-03-01                                      NaN
```

Next

The idea is to bracket index a variable `variable[condition]` to get a conditional mean difference within dplyr::summarise(). This allow filtering on specific variables.

https://gist.github.com/andrewheiss/0724b29a8c8b0c3bd0a49b896454be87

- Note: a next step could make this n_tile. With the n specified as a variable.

```
# Manual way that gives you the most control over the areas
variable <- rnorm(1000)
variable_density <- density(variable)

density_df <- tibble(x = variable_density$x,
                     y = variable_density$y) %>%
              mutate(fill_stuff = case_when(x < -1 ~ "Lower end",
                                            x > 1 ~ "Upper end",
                                            TRUE ~ "Middle area"))
```

```
density_df %>%
  mutate(ntile = ntile(x, 3),
         name = ordinal(ntile)) %>%
  distinct(ntile, name)
```

```
## # A tibble: 3 x 2
##   ntile name
##   <int> <ordinal>
## 1     1 first
## 2     2 second
## 3     3 third
```

```
  # pull(x_ntile) %>% unique(.) %>% sort(.)

sym("geom_area(data = filter(data, fill_stuff == 'Lower end'),   aes(fill = 'Lower end'))")
```

```
## `geom_area(data = filter(data, fill_stuff == 'Lower end'),   aes(fill = 'Lower end'))`
```

```
# for (i in 1:3) {
#
#   geom_area(data = filter(data, fill_stuff == ordinal(i)),   aes(fill = ordinal(i)))
#
# }
```

Curly Curly {{}} is used to pull the column names for adding labels to the chart with glue().

`plyr::select({{x_var}}) %>% colnames(.)`

https://www.datanovia.com/en/blog/top-r-color-palettes-to-know-for-great-data-visualization/
http://www.sthda.com/english/wiki/ggplot2-colors-how-to-change-colors-automatically-and-manually

```
shaded_density_tertiles <- function(data, x_var, y_var, ntiles) {

  x_name <- data %>% dplyr::select({{x_var}}) %>% colnames(.)
  y_name <- data %>% dplyr::select({{y_var}}) %>% colnames(.)
```

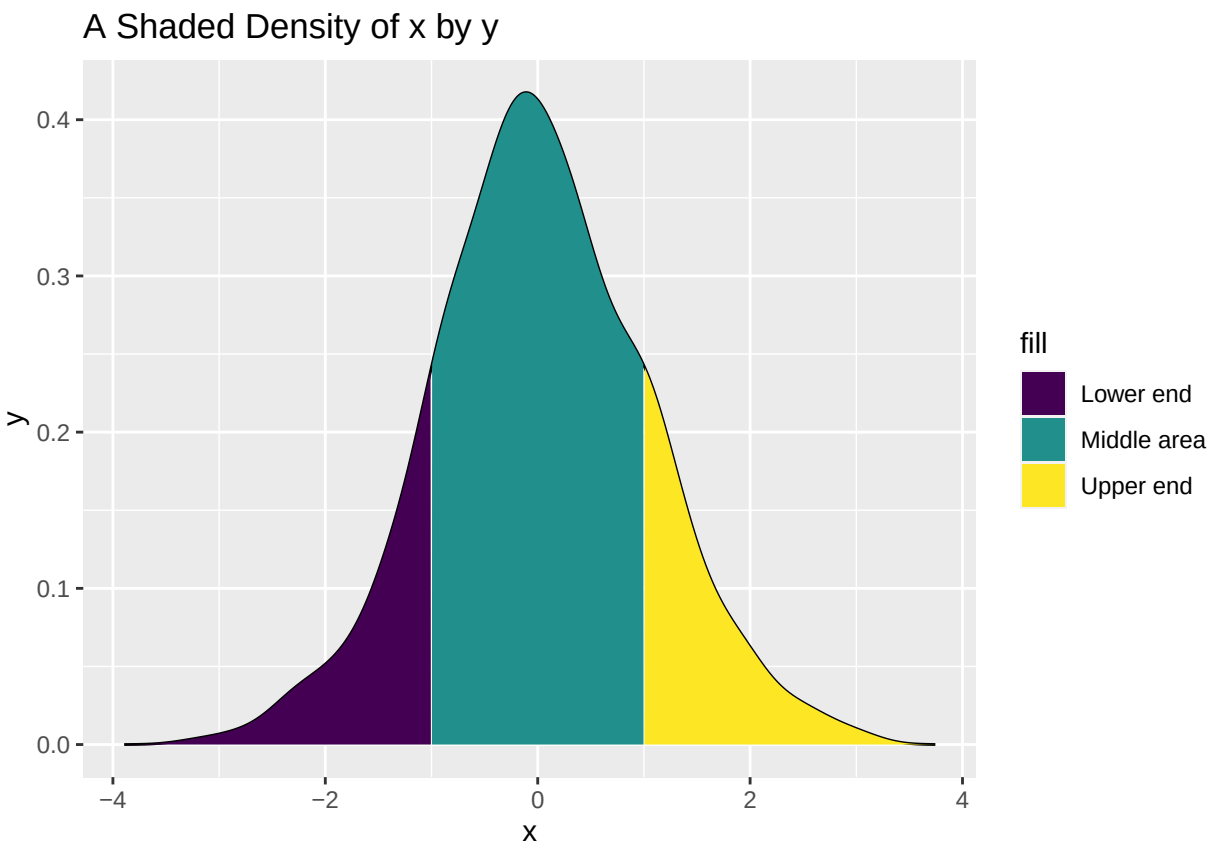
```
  lower <- sym(c("geom_area(data = filter(data, fill_stuff == 'Lower end'),   aes(fill = 'Lower end'))"))

  lower <- geom_area(data = filter(data, fill_stuff == 'Lower end'),   aes(fill = 'Lower end'))

  data %>%
    ggplot(., aes(x = {{x_var}}, y = {{y_var}})) +
    geom_line() +
    lower +
    # geom_area(data = filter(data, fill_stuff == "Lower end"),   aes(fill = "Lower end")) +
    geom_area(data = filter(data, fill_stuff == "Upper end"),   aes(fill = "Upper end")) +
    geom_area(data = filter(data, fill_stuff == "Middle area"), aes(fill = "Middle area")) +
    # scale_fill_manual(values = c("red", "lightblue", "orange")) +
    # scale_fill_brewer(palette="Spectral") +
    # scale_color_viridis(option = "D") +
    scale_fill_viridis(discrete = TRUE) +
    labs(title = glue::glue("A Shaded Density of {x_name} by {y_name}"),
         color = 'huh')

}

shaded_density_tertiles(density_df, x, y)
```




```
shaded_density_tertiles <- function(data, x_var, y_var, ntiles) {

  x_name <- data %>% dplyr::select({{x_var}}) %>% colnames(.)
  y_name <- data %>% dplyr::select({{y_var}}) %>% colnames(.)
```

```
  data <- data %>%
            mutate(x_ntile = ntile(x, ntiles))

  data %>%
    ggplot(., aes(x = {{x_var}}, y = {{y_var}})) +
    geom_line() +
    geom_area(data = filter(data, fill_stuff == "Lower end"),   aes(fill = "Lower end")) +
    geom_area(data = filter(data, fill_stuff == "Upper end"),   aes(fill = "Upper end")) +
    geom_area(data = filter(data, fill_stuff == "Middle area"), aes(fill = "Middle area")) +
    # scale_fill_manual(values = c("red", "lightblue", "orange")) +
    # scale_fill_brewer(palette="Spectral") +
    # scale_color_viridis(option = "D") +
    scale_fill_viridis(discrete = TRUE) +
    labs(title = glue::glue("A Shaded Density of {x_name} by {y_name}"),
         color = 'huh')

}

shaded_density_tertiles(density_df, x, y)
```