Useful things

true

2022-10-09

Table of contents

This chapter is a collection of things I wish I had known earlier in my years using R and that I hope can be of use to you. Sections are named after R packages or whatever applies and sorted alphabetically.

1 {dlookr} - descriptive statistics

When providing descriptive statistics tables, one may find the number of relevant measures become annoyingly large so that even with the {tidyverse}, several lines of code are necessary. Here are just five measures, and they are not even including the `na.rm = TRUE` argument, which is necessary for data with missing values.

```
library(tidyverse)

PlantGrowth %>%
  group_by(group) %>%
  summarise(
    mean = mean(weight),
    stddev = sd(weight),
    median = median(weight),
```

```
    min = min(weight),
    max = max(weight)
  )
```

```
# A tibble: 3 x 6
  group  mean stddev median   min   max
  <fct> <dbl>  <dbl>  <dbl> <dbl> <dbl>
1 ctrl   5.03  0.583   5.15  4.17  6.11
2 trt1   4.66  0.794   4.55  3.59  6.03
3 trt2   5.53  0.443   5.44  4.92  6.31
```

Obviously, there are multiple packages who try to address just that. The one I've been using for some time now is {dlookr} with its `describe()` function. It actually provides more measures than I usually need[1], but it has everything I want and I disregard the rest (via `select()`).

```
library(dlookr)

PlantGrowth %>%
  group_by(group) %>%
  describe(weight)
```

```
# A tibble: 3 x 27
  described_~1 group     n    na  mean    sd se_mean   IQR skewn~2 kurto~3   p00
  <chr>        <fct> <int> <int> <dbl> <dbl>   <dbl> <dbl>   <dbl>   <dbl> <dbl>
1 weight       ctrl     10     0  5.03 0.583   0.184 0.743   0.321  -0.229  4.17
2 weight       trt1     10     0  4.66 0.794   0.251 0.662   0.659  -0.203  3.59
3 weight       trt2     10     0  5.53 0.443   0.140 0.467   0.673  -0.324  4.92
# ... with 16 more variables: p01 <dbl>, p05 <dbl>, p10 <dbl>, p20 <dbl>,
#   p25 <dbl>, p30 <dbl>, p40 <dbl>, p50 <dbl>, p60 <dbl>, p70 <dbl>,
#   p75 <dbl>, p80 <dbl>, p90 <dbl>, p95 <dbl>, p99 <dbl>, p100 <dbl>, and
#   abbreviated variable names 1: described_variables, 2: skewness, 3: kurtosis
```

2 {forcats} - factor level handling

In my experience, R beginners really only care about the difference between `factor` and `character` variables once the factor level order is not as they want it to be - typically on the x-axis of a plot. Luckily, {forcats} can deal with this.

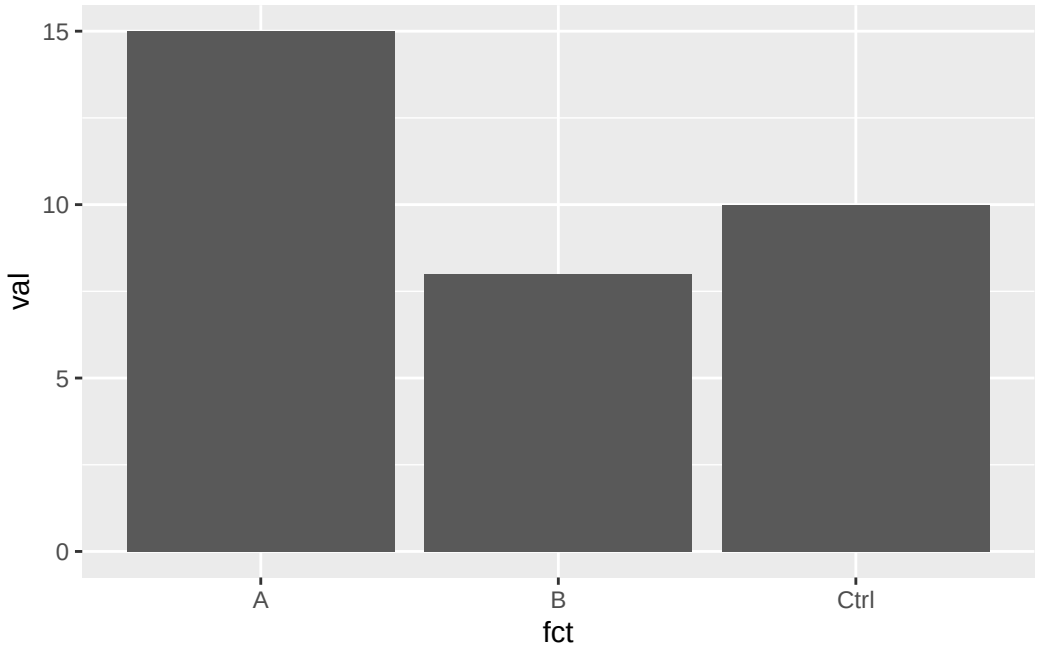
[1] Keep in mind that `p00` is the 0th percentile and thus the minimum. Analogously, `p50` is the median and `p100` the maximum.

In the following example, we create a column `fct` that is a copy of the column `chr`, except that they are formatted as `factor` and `character`, respectively.

```
library(tidyverse)

dat <- tribble(
     ~val,   ~chr,
       10, "Ctrl",
       15,    "A",
        8,    "B"
) %>%
  mutate(fct = as.factor(chr))

ggplot(dat) +
  aes(y = val, x = fct) +
  geom_col()
```



Even though the data is sorted so that Ctrl is first, then A, then B, the x-Axis is sorted differently[2]. This is because factor levels are always sorted alphabetically by default. We can reorder them via different functions in {forcats}:

[2]It does not make a difference here, whether we put `x = chr` or `x = fct` in the ggplot statement.

```r
dat <- dat %>%
  mutate(
    fct2 = fct_relevel(fct, c("Ctrl", "A", "B")),
    fct3 = fct_reorder(fct, val, mean)
    )

str(dat)
```

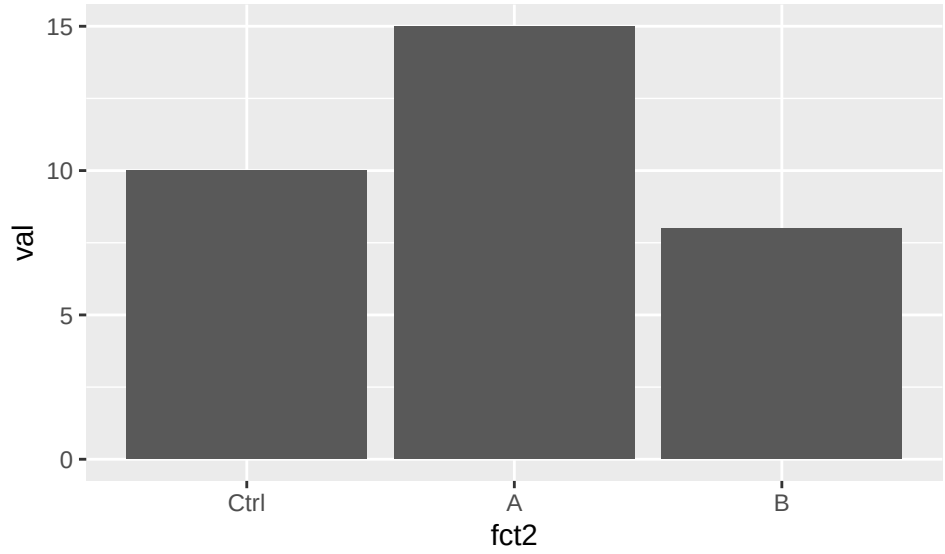
```
tibble [3 x 5] (S3: tbl_df/tbl/data.frame)
 $ val : num [1:3] 10 15 8
 $ chr : chr [1:3] "Ctrl" "A" "B"
 $ fct : Factor w/ 3 levels "A","B","Ctrl": 3 1 2
 $ fct2: Factor w/ 3 levels "Ctrl","A","B": 1 2 3
 $ fct3: Factor w/ 3 levels "B","Ctrl","A": 2 3 1
```

Above are just two popular examples for sorting factor levels: `fct_relevel` sorts the levels manually by providing a vector with the level names in the order they should appear, while `fct_reorder` here sorts the levels according to their respective group mean[3] of the values in the `val` column.

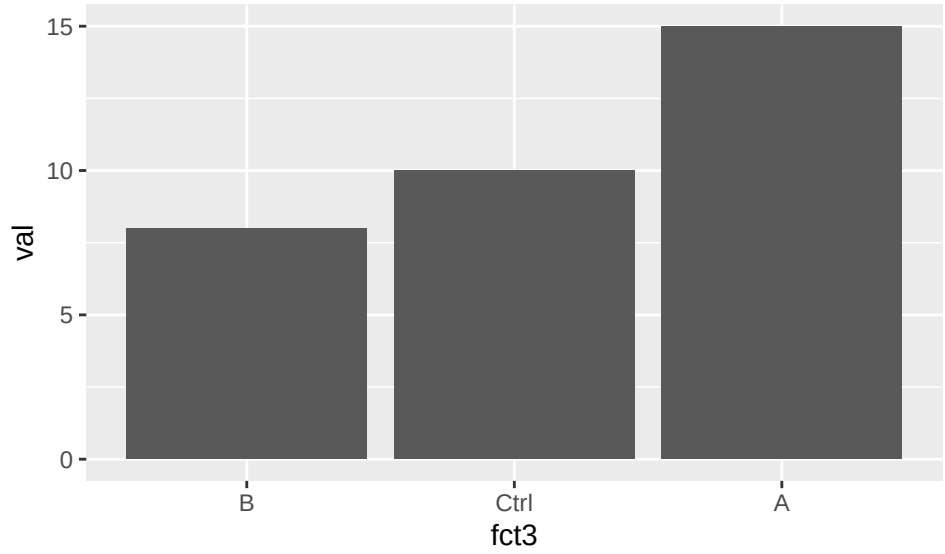
You can see that it worked in the output of `str(dat)` above and in the plots below.

```r
ggplot(dat) +
  aes(y = val, x = fct2) +
  geom_col()
```

[3] Yes, the mean in this example is not really a mean, since there is only one number per group.




```
ggplot(dat) +
  aes(y = val, x = fct3) +
  geom_col()
```



3 {ggtext} - automatic word wrapping

Adding long text to plots created via {ggplot2} is problematic, since you have to insert line breaks yourself. However, {ggext}'s `geom_textbox()` for data labels and `element_textbox_simple()` for title, caption etc. will automatically add line breaks:

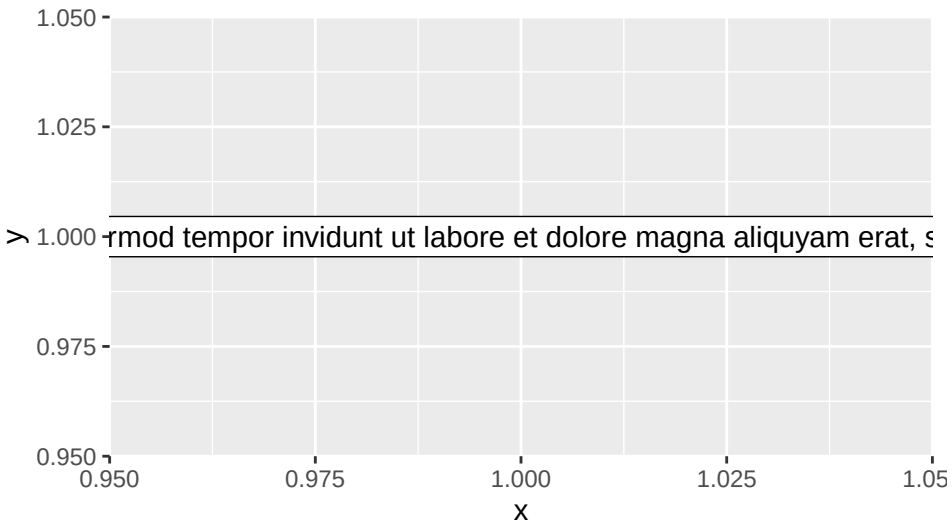
```
longtext <- "Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirm
```

```
library(ggplot2)

ggplot() +



  aes(y = 1, x = 1, label = longtext) +
  geom_label() +
  labs(caption = longtext)
```

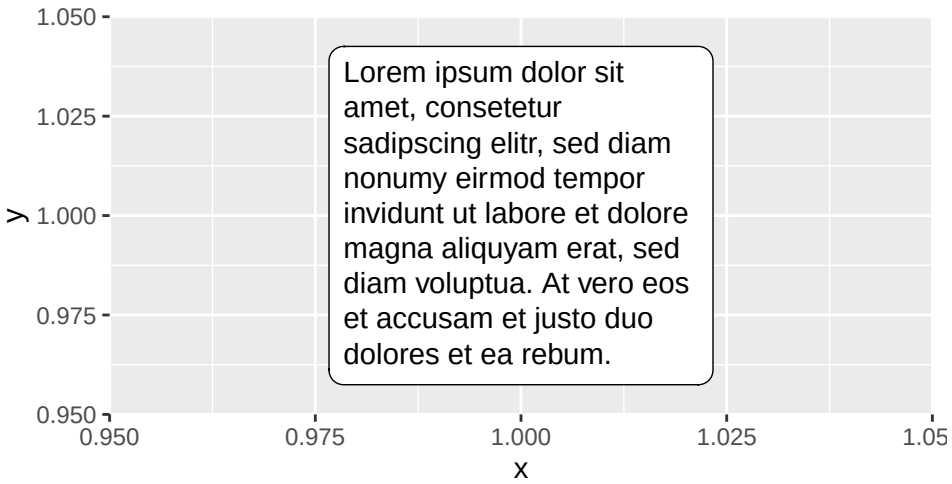


quyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum.

```
library(ggtext)

ggplot() +
  theme(plot.caption =
          element_textbox_simple()
```

```
      ) +
aes(y = 1, x = 1, label = longtext) +
geom_textbox() +
labs(caption = longtext)
```



Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum.

4 %in% and %not_in%

R has the built-in function `%in%` which checks whether something is present in a vector.

```
treatments <- c("Ctrl", "A", "B")
```

Not only can we checke which treatments are present in our `treatment` vector (left), but we can also easily keep only those that are (right).

```
c("A", "D") %in% treatments
```

```
[1]  TRUE FALSE
```

```
c("A", "D") %>% .[. %in% treatments]
```

```
[1] "A"
```

Not built-in, for some reason, is the opposite of that function - checking whether something is **not** present. Yet, we can quickly built our own function that does exactly that:

```
`%not_in%` <- Negate(`%in%`)
```

```
c("A", "D") %not_in% treatments
```

```
[1] FALSE  TRUE
```

```
c("A", "D") %>% .[. %not_in% treatments]
```

```
[1] "D"
```