Tips and Tricks for Table 1's

Kat Hoffman

WCM Biostatistics Computing Club

June 23, 2022

Three packages/functions to speed up your table-making

1. `library(tidylog)`
2. `labelled::set_variable_names()`
3. `snakecase::to_title_case()`

Data set-up

- First we'll simulate a `patients` data set (wide-format demographic characteristics)

```
# load tidyverse for data creation and set seed for reproducible data
library(tidyverse)
set.seed(7)

# data set of basic patient demographics
patients <-
  tribble(
    ~id, ~admit_dt, ~death_or_discharge_dt,
    ~age, ~sex, ~height, ~weight, ~current_smoker, ~immunosuppressed,
    100, "2020-03-21 00:10", "2020-05-13 12:10",
64, "Male", 68, 199, "Yes", "No",
    104, "2020-04-03 12:15", "2020-04-29 18:34",
25, "Male", 72, NA, "Yes", "No",
    106, "2020-03-28 12:22", "2020-04-05 19:18",
49, "Female", 64, 189, "No", "Yes",
     107, "2020-04-10 18:15","2020-04-14 19:12",
 88, "Male", 62, 111, "No", "Yes",
    111, "2020-04-18 00:49", "2020-04-25 19:18",
61, "Female", 67, 156, "No", "Yes"
  ) |>
  # set time zone for date time variables
  mutate_at(vars(ends_with("_dt")), ~as.POSIXct(., tz="America/New_York"))
```

patients

- Wide data set - one row per patient

```
patients
```

```
## # A tibble: 5 × 9
##      id admit_dt           death_or_discharge_dt   age sex    height weight
##   <dbl> <dttm>             <dttm>                <dbl> <chr>   <dbl>  <dbl>
## 1   100 2020-03-21 00:10:00 2020-05-13 12:10:00      64 Male       68    199
## 2   104 2020-04-03 12:15:00 2020-04-29 18:34:00      25 Male       72     NA
## 3   106 2020-03-28 12:22:00 2020-04-05 19:18:00      49 Female     64    189
## 4   107 2020-04-10 18:15:00 2020-04-14 19:12:00      88 Male       62    111
## 5   111 2020-04-18 00:49:00 2020-04-25 19:18:00      61 Female     67    156
## # … with 2 more variables: current_smoker <chr>, immunosuppressed <chr>
```

Tip #1: Use `library(tidylog)`!

- `tidylog` is a package that gives additional feedback when you use `dplyr` functions. Simply load it at the top of your `R` script.

```
library(tidylog)
```

- Then, for example, when you mutate a column, it will tell you how many new and `NA` values you created:

```
patients <-
  patients |>
  # compute BMI
  mutate(bmi = weight / height^2 * 703) |>
  # remove the patients height and weight from the data frame
  select(-height, -weight)
```

```
## mutate: new variable 'bmi' (double) with 5 unique values and 20% NA

## select: dropped 2 variables (height, weight)
```

Using `tidylog` for joins

- I've found it most useful for the feedback when you join two data sets:

```
patient_labs <-
  patients |>
  left_join(labs)
```

```
## Joining, by = "id"
## left_join: added 3 columns (lab_time, lab_name, lab_value)
## > rows only in x 1
## > rows only in y (994)
## > matched rows 629 (includes duplicates)
## > =====
## > rows total 630
```

- Also provides feedback for `summarize`-related and `pivot_*` functions

"`tidylog` is not a package...it's a lifestyle."

> *-Imaani Easthausen, former WCM biostatistician*

Lesson: load `tidylog` at the top of all your scripts for more efficient and accurate data manipulation. Save time the next time you experience this:

Tip #2: Use

`labelled::set_variable_labels()`

`labelled` package

- `labelled` is a package to quickly easily relabel variables and values
- `set_variable_labels()` allows you to input a named list of variable names and labels within `dplyr` syntax, EX:

```
library(labelled)
df <- tibble(s1 = c("M", "M", "F"), s2 = c(1, 1, 2)) %>%
    set_variable_labels(s1 = "Sex", s2 = "Yes or No?")
```

Use `labelled` to improve your tables

```
library(gtsummary)
library(gt)
patients |>
  # select vars of interest for tables
  select(age, sex, bmi, current_smoker, immunosuppressed) |>
  tbl_summary(
     # don't show missing (unknown) values
    missing = "no",
    # make sure all numeric variables are reported as continuous
    type = list(where(is.numeric) ~ "continuous")
  ) |>
  # bold the labels
  bold_labels()
```

Characteristic	N = 5[1]
age	61 (49, 64)
sex	
Female	2 (40%)
Male	3 (60%)
bmi	27.3 (23.4, 30.8)
current_smoker	2 (40%)
immunosuppressed	3 (60%)

[1] Median (IQR); n (%)

Option 1: use `labelled` to rename variables manually

```
tbl1_vars <-
  patients |>
  # select vars of interest for tables
  select(age, sex, bmi, current_smoker, immunosuppressed)
```

```
tbl1_vars |>
  # edit variable names using labelled package
  labelled::set_variable_labels(
    # change all variable labels to "Title Case"
    age = "Age",
    sex = "Sex",
    current_smoker = "Current Smoker",
    immunosuppressed = "Immunosuppressed",
    bmi = "BMI"
    ) |>
  tbl_summary(
    # make sure all numeric variables are reported as continuous
    type = list(where(is.numeric) ~ "continuous")
  )
```

Output from Option 1

Characteristic	N = 5[1]
Age	61 (49, 64)
Sex	
Female	2 (40%)
Male	3 (60%)
BMI	27.3 (23.4, 30.8)
Unknown	1
Current Smoker	2 (40%)
Immunosuppressed	3 (60%)

[1] Median (IQR); n (%)

But can we make it easier??

Introducing the `snakecase` package (Tip #3)

- `snakecase` parses string to a specified case, e.g. snake_case, lowerCamel, UpperCamel, ALL_CAPS, lowerUPPER, UPPERlower, Sentence case, Title Case
- use it to clean up your variable names

```
snakecase::to_upper_lower_case(names(mtcars))
```

```
##  [1] "MPG"  "CYL"  "DISP" "HP"   "DRAT" "WT"   "QSEC" "VS"   "AM"   "GEAR"
## [11] "CARB"
```

Option 2: add labelling schema from the `snakecase` package

```
tbl1_vars |>
  # edit variable names using labelled package
  labelled::set_variable_labels(
    # change all variable labels to "Title Case"
    .labels = snakecase::to_title_case(names(tbl1_vars)),
    # change any extra variables that are not title case, like BMI
    bmi = "BMI"
    ) |>
  tbl_summary(
    # make sure all numeric variables are reported as continuous
    type = list(where(is.numeric) ~ "continuous")
  )
```

Output from Option 2

Characteristic	N = 5[1]
Age	61 (49, 64)
Sex	
Female	2 (40%)
Male	3 (60%)
BMI	27.3 (23.4, 30.8)
Current Smoker	2 (40%)
Immunosuppressed	3 (60%)

[1] Median (IQR); n (%)

the end!

> These tips and a few more in the blog post *Lessons learned: my top five coding 'tricks' during the NYC COVID-19 outbreak* (www.khstats.com)