
tidylog

downloads 2653/month

downloads 2653/month

codecov 99%

codecov 99%

The goal of tidylog is to provide feedback about dplyr and tidyr operations. It provides simple wrapper functions for almost all dplyr and tidyr functions, such as `filter`, `mutate`, `select`, `full_join`, and `group_by`.

Example

Load tidylog after dplyr and/or tidyr:

```
1 library("dplyr")
2 library("tidyr")
3 library("tidylog", warn.conflicts = FALSE)
```

Tidylog will give you feedback, for instance when filtering a data frame or adding a new variable:

```
1 filtered <- filter(mtcars, cyl == 4)
2 #> filter: removed 21 rows (66%), 11 rows remaining
3 mutated <- mutate(mtcars, new_var = wt ** 2)
4 #> mutate: new variable 'new_var' (double) with 29 unique values and 0%
  NA
```

Tidylog reports detailed information for joins:

```
1 joined <- left_join(nycflights13::flights, nycflights13::weather,
2   by = c("year", "month", "day", "origin", "hour", "time_hour"))
3 #> left_join: added 9 columns (temp, dewp, humid, wind_dir, wind_speed,
  ...)
4 #>           > rows only in nycflights13::flights      1,556
5 #>           > rows only in nycflights13::weather (    6,737)
6 #>           > matched rows                          335,220
7 #>           >                                           =====
8 #>           > rows total                             336,776
```

In this case, we see that 1,556 rows from the `flights` dataset do not have weather information.

Tidylog can be especially helpful in longer pipes:

```
1 summary <- mtcars %>%
2   select(mpg, cyl, hp, am) %>%
3   filter(mpg > 15) %>%
4   mutate(mpg_round = round(mpg)) %>%
5   group_by(cyl, mpg_round, am) %>%
6   tally() %>%
7   filter(n >= 1)
8 #> select: dropped 7 variables (disp, drat, wt, qsec, vs, ...)
```

```
9 #> filter: removed 6 rows (19%), 26 rows remaining
10 #> mutate: new variable 'mpg_round' (double) with 15 unique values and
    0% NA
11 #> group_by: 3 grouping variables (cyl, mpg_round, am)
12 #> tally: now 20 rows and 4 columns, 2 group variables remaining (cyl,
    mpg_round)
13 #> filter (grouped): no rows removed
```

Here, it might have been accidental that the last `filter` command had no effect.

Installation

Download from CRAN:

```
1 install.packages("tidylog")
```

Or install the development version:

```
1 devtools::install_github("elbersb/tidylog")
```

Benchmarks

Tidylog will add a small overhead to each function call. This can be relevant for very large datasets and especially for joins. If you want to switch off tidylog for a single long-running command, simply prefix `dplyr::` or `tidyr::`, such as in `dplyr::left_join`. See this vignette for more information.

More examples

filter, distinct, drop_na

```
1 a <- filter(mtcars, mpg > 20)
2 #> filter: removed 18 rows (56%), 14 rows remaining
3 b <- filter(mtcars, mpg > 100)
4 #> filter: removed all rows (100%)
5 c <- filter(mtcars, mpg > 0)
6 #> filter: no rows removed
7 d <- filter_at(mtcars, vars(starts_with("d")), any_vars((. %> 2) == 0))
8 #> filter_at: removed 19 rows (59%), 13 rows remaining
9 e <- distinct(mtcars)
10 #> distinct: no rows removed
11 f <- distinct_at(mtcars, vars(vs:carb))
12 #> distinct_at: removed 18 rows (56%), 14 rows remaining
13 g <- top_n(mtcars, 2, am)
14 #> top_n: removed 19 rows (59%), 13 rows remaining
```

```
15 i <- sample_frac(mtcars, 0.5)
16 #> sample_frac: removed 16 rows (50%), 16 rows remaining
17
18 j <- drop_na(airquality)
19 #> drop_na: removed 42 rows (27%), 111 rows remaining
20 k <- drop_na(airquality, Ozone)
21 #> drop_na: removed 37 rows (24%), 116 rows remaining
```

mutate, transmute, replace_na, fill

```
1 a <- mutate(mtcars, new_var = 1)
2 #> mutate: new variable 'new_var' (double) with one unique value and 0%
  NA
3 b <- mutate(mtcars, new_var = runif(n()))
4 #> mutate: new variable 'new_var' (double) with 32 unique values and 0%
  NA
5 c <- mutate(mtcars, new_var = NA)
6 #> mutate: new variable 'new_var' (logical) with one unique value and
  100% NA
7 d <- mutate_at(mtcars, vars(mpg, gear, drat), round)
8 #> mutate_at: changed 28 values (88%) of 'mpg' (0 new NAs)
9 #>
  changed 31 values (97%) of 'drat' (0 new NAs)
10 e <- mutate(mtcars, am_factor = as.factor(am))
11 #> mutate: new variable 'am_factor' (factor) with 2 unique values and
  0% NA
12 f <- mutate(mtcars, am = as.ordered(am))
13 #> mutate: converted 'am' from double to ordered factor (0 new NA)
14 g <- mutate(mtcars, am = ifelse(am == 1, NA, am))
15 #> mutate: changed 13 values (41%) of 'am' (13 new NAs)
16 h <- mutate(mtcars, am = recode(am, `0` = "zero", `1` = NA_character_))
17 #> mutate: converted 'am' from double to character (13 new NA)
18
19 i <- transmute(mtcars, mpg = mpg * 2, gear = gear + 1, new_var = vs +
  am)
20 #> transmute: dropped 9 variables (cyl, disp, hp, drat, wt, ...)
21 #>
  changed 32 values (100%) of 'mpg' (0 new NAs)
22 #>
  changed 32 values (100%) of 'gear' (0 new NAs)
23 #>
  new variable 'new_var' (double) with 3 unique values and
  0% NA
24
25 j <- replace_na(airquality, list(Solar.R = 1))
26 #> replace_na: converted 'Solar.R' from integer to double (7 fewer NA)
27 k <- fill(airquality, Ozone)
28 #> fill: changed 37 values (24%) of 'Ozone' (37 fewer NAs)
```

joins

For joins, tidylog provides more detailed information. For any join, tidylog will show the number of rows that are only present in x (the first dataframe), only present in y (the second dataframe), and rows that have been matched. Numbers in parentheses indicate that these rows are not included in the result. Tidylog will also indicate whether any rows were duplicated (which is often unintentional):

```
1 x <- tibble(a = 1:2)
2 y <- tibble(a = c(1, 1, 2), b = 1:3) # 1 is duplicated
3 j <- left_join(x, y, by = "a")
4 #> left_join: added one column (b)
5 #> > rows only in x 0
6 #> > rows only in y (0)
7 #> > matched rows 3 (includes duplicates)
8 #> > ===
9 #> > rows total 3
```

More examples:

```
1 a <- left_join(band_members, band_instruments, by = "name")
2 #> left_join: added one column (plays)
3 #> > rows only in band_members 1
4 #> > rows only in band_instruments (1)
5 #> > matched rows 2
6 #> > ===
7 #> > rows total 3
8 b <- full_join(band_members, band_instruments, by = "name")
9 #> full_join: added one column (plays)
10 #> > rows only in band_members 1
11 #> > rows only in band_instruments 1
12 #> > matched rows 2
13 #> > ===
14 #> > rows total 4
15 c <- anti_join(band_members, band_instruments, by = "name")
16 #> anti_join: added no columns
17 #> > rows only in band_members 1
18 #> > rows only in band_instruments (1)
19 #> > matched rows (2)
20 #> > ===
21 #> > rows total 1
```

Because tidylog needs to perform two additional joins behind the scenes to report this information, the overhead will be larger than for the other tidylog functions (especially with large datasets).

select, relocate, rename

```
1 a <- select(mtcars, mpg, wt)
```

```

2 #> select: dropped 9 variables (cyl, disp, hp, drat, qsec, ...)
3 b <- select_if(mtcars, is.character)
4 #> select_if: dropped all variables
5 c <- relocate(mtcars, hp)
6 #> relocate: columns reordered (hp, mpg, cyl, disp, drat, ...)
7 d <- select(mtcars, a = wt, b = mpg)
8 #> select: renamed 2 variables (a, b) and dropped 9 variables
9
10 e <- rename(mtcars, miles_per_gallon = mpg)
11 #> rename: renamed one variable (miles_per_gallon)
12 f <- rename_with(mtcars, toupper)
13 #> rename_with: renamed 11 variables (MPG, CYL, DISP, HP, DRAT, ...)

```

summarize

```

1 a <- mtcars %>%
2   group_by(cyl, carb) %>%
3   summarize(total_weight = sum(wt))
4 #> group_by: 2 grouping variables (cyl, carb)
5 #> summarize: now 9 rows and 3 columns, one group variable remaining (cyl)
6
7 b <- iris %>%
8   group_by(Species) %>%
9   summarize_all(list(min, max))
10 #> group_by: one grouping variable (Species)
11 #> summarize_all: now 3 rows and 9 columns, ungrouped

```

tally, count, add_tally, add_count

```

1 a <- mtcars %>% group_by(gear, carb) %>% tally
2 #> group_by: 2 grouping variables (gear, carb)
3 #> tally: now 11 rows and 3 columns, one group variable remaining (gear)
4 b <- mtcars %>% group_by(gear, carb) %>% add_tally()
5 #> group_by: 2 grouping variables (gear, carb)
6 #> add_tally (grouped): new variable 'n' (integer) with 5 unique values and 0% NA
7
8 c <- mtcars %>% count(gear, carb)
9 #> count: now 11 rows and 3 columns, ungrouped
10 d <- mtcars %>% add_count(gear, carb, name = "count")
11 #> add_count: new variable 'count' (integer) with 5 unique values and 0% NA

```

pivot_longer, pivot_wider

```
1 longer <- mtcars %>%
2   mutate(id = 1:n()) %>%
3   pivot_longer(-id, names_to = "var", values_to = "value")
4 #> mutate: new variable 'id' (integer) with 32 unique values and 0% NA
5 #> pivot_longer: reorganized (mpg, cyl, disp, hp, drat, ...) into (var,
6   value) [was 32x12, now 352x3]
7 wider <- longer %>%
8   pivot_wider(names_from = var, values_from = value)
9 #> pivot_wider: reorganized (var, value) into (mpg, cyl, disp, hp, drat
10  , ...) [was 352x3, now 32x12]
```

Tidylog also supports `gather` and `spread`.

Turning logging off, registering additional loggers

To turn off the output for just a particular function call, you can simply call the `dplyr` and `tidyr` functions directly, e.g. `dplyr::filter` or `tidyr::drop_na`.

To turn off the output more permanently, set the global option `tidylog.display` to an empty list:

```
1 options("tidylog.display" = list()) # turn off
2 a <- filter(mtcars, mpg > 20)
3
4 options("tidylog.display" = NULL)    # turn on
5 a <- filter(mtcars, mpg > 20)
6 #> filter: removed 18 rows (56%), 14 rows remaining
```

This option can also be used to register additional loggers. The option `tidylog.display` expects a list of functions. By default (when `tidylog.display` is set to `NULL`), tidylog will use the `message` function to display the output, but if you prefer a more colorful output, simply overwrite the option:

```
1 library("crayon") # for terminal colors
2 crayon <- function(x) cat(red$bold(x), sep = "\n")
3 options("tidylog.display" = list(crayon))
4 a <- filter(mtcars, mpg > 20)
5 #> filter: removed 18 rows (56%), 14 rows remaining
```

To print the output both to the screen and to a file, you could use:

```
1 log_to_file <- function(text) cat(text, file = "log.txt", sep = "\n",
2   append = TRUE)
3 options("tidylog.display" = list(message, log_to_file))
4 a <- filter(mtcars, mpg > 20)
5 #> filter: removed 18 rows (56%), 14 rows remaining
```

Namespace conflicts

Tidylog redefines several of the functions exported by dplyr and tidyr, so it should be loaded last, otherwise there will be no output. A more explicit way to resolve namespace conflicts is to use the conflicted package:

```
1 library("dplyr")
2 library("tidyr")
3 library("tidylog")
4 library("conflicted")
5 for (f in getNamespaceExports("tidylog")) {
6   conflicted::conflict_prefer(f, "tidylog", quiet = TRUE)
7 }
```