Concise indicator variable recoding with ind2cat

by Evangeline Reynolds

Abstract Indicator variables are often used in data analyses given the ease which with they are created, stored and interpreted. They concisely encode information about the presence or not of a condition for observational units. The variable name encapsulates the information about the true condition, the variable's values (TRUE and FALSE, 1 or 0, "Yes" or "No"), indicate if the condition is true for the observational unit. When using indicator variables to use in summary products, analysts often make a choice between using an indicator variable as-is or crafting categorical variables where values can be directly interpreted. Using the indicator variable as-is may be motivated by time savings. {{ind2cat}} can help analysts concisely translate indicator variables to categorical variables for reporting products, yielding more polished outputs even in the exploratory analysis stage. By default, ind2cat creates the categorical variable from the indicator variable name, resulting in a light weight syntax.

1) Is there already a solution
2) unsure if there are fundamental problems with this approach

Some things to move ind2cat out of proof of concept phase

- change to Rlang for grabbing function name (Claus Wilke)
- left join instead of ifelse to make code more performant (Emily Rederer)
- make "Y" "N" a lot stricter - right now we're assuming a ton! Danger.

1 Introduction

Recoding indicator variable values to meaningful and appropriately ordered categories often involves redundancy.

more description of example here

```
library(tidyverse)

data.frame(ind_graduated = c(T,T,F)) |>
  mutate(cat_graduated  = ifelse(ind_graduated,
                                 "graduated",
                                 "not graduated")) |>
  mutate(cat_graduated = fct_rev(cat_graduated))
```

```
      ind_graduated cat_graduated
    1          TRUE     graduated
    2          TRUE     graduated
    3         FALSE not graduated
```

ind2cat's ind_recode function avoids repetition by creating categories based on the indicator variable name. Using the the function ind_recode(), we can accomplish the same task shown above more succinctly:

```
library(ind2cat)

data.frame(ind_graduated = c(T,T,F)) |>
  mutate(cat_graduated  = ind_recode(ind_graduated))
```

```
      ind_graduated cat_graduated
    1          TRUE     graduated
    2          TRUE     graduated
    3         FALSE not graduated
```

Furthermore, ind_recode's functionality allows analysts to move from a first-cut recode that delivers meaningful categories to fully customized categories.

```
data.frame(ind_graduated = c(T,T,F)) %>%
  mutate(cat_graduated  = ind_recode(ind_graduated,
                                     cat_false = "current"))
```

```
  ind_graduated cat_graduated
1          TRUE     graduated
2          TRUE     graduated
3         FALSE       current
```

2 Status-Quo: Analysts make a choice between directly using indicators or verbose recode

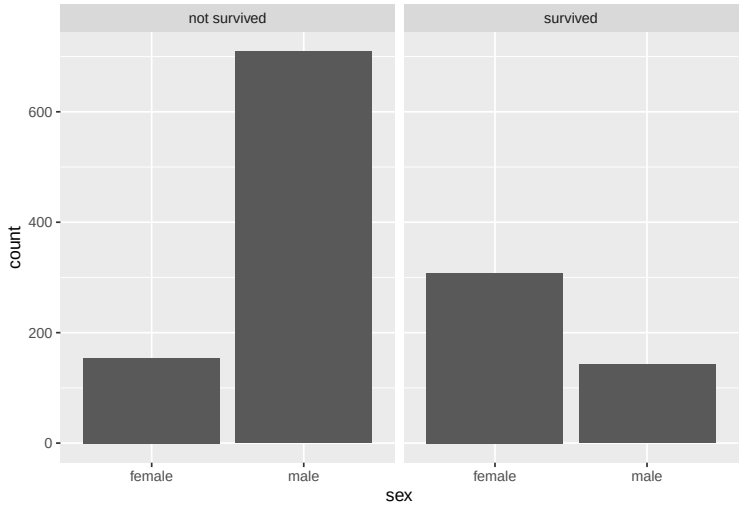
Current proceedures for recoding indicator variables to a categorial variable is inelegant.

Current methods for recoding indicator variables is repetitive and verbose, as shown in the examples that follow.

```
tidytitanic::passengers %>%
  tibble() %>%
  mutate(cat_survived = ifelse(survived, "survived", "not survived"),
         .before = 1)
```

```
   # A tibble: 1,313 x 6
      cat_survived name                                  class   age sex   survi~1
      <chr>        <chr>                                 <chr> <dbl> <chr>   <int>
    1 survived     Allen, Miss Elisabeth Walton          1st   29    fema~       1
    2 not survived Allison, Miss Helen Loraine           1st    2    fema~       0
    3 not survived Allison, Mr Hudson Joshua Creighton   1st   30    male        0
    4 not survived Allison, Mrs Hudson JC (Bessie Waldo ~ 1st   25    fema~       0
    5 survived     Allison, Master Hudson Trevor         1st    0.92 male        1
    6 survived     Anderson, Mr Harry                    1st   47    male        1
    7 survived     Andrews, Miss Kornelia Theodosia      1st   63    fema~       1
    8 not survived Andrews, Mr Thomas, jr                1st   39    male        0
    9 survived     Appleton, Mrs Edward Dale (Charlotte ~ 1st   58    fema~       1
   10 not survived Artagaveytia, Mr Ramon                1st   71    male        0
   # ... with 1,303 more rows, and abbreviated variable name 1: survived
```

```
tidytitanic::passengers %>%
ggplot() +
  aes(x = sex) +
  geom_bar() +
  facet_grid(~ ifelse(survived, "survived", "not survived"))
```



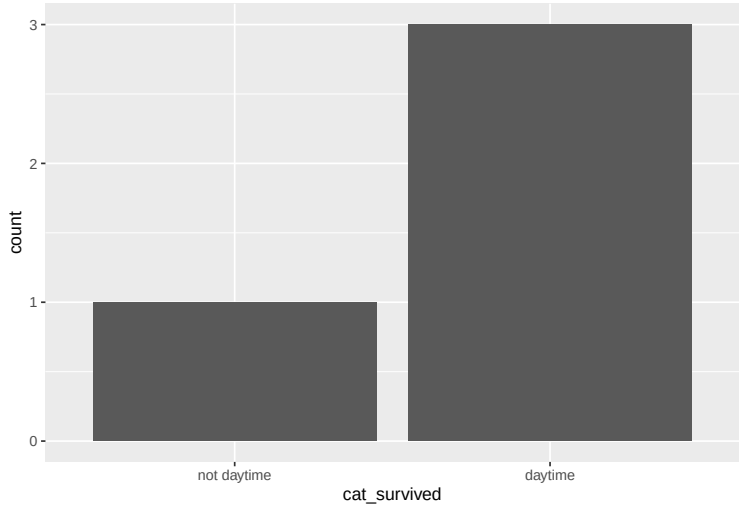
This solution above also does not address category display ordering; ordering in products will be alphabetical and not reflect the F/T order of the source variable. An additional step to reflect the source variable, using a function like forcats::fct_rev, may be required for consistency in reporting.

```
data.frame(ind_daytime = c(T, F, T, T)) %>%
    mutate(cat_survived = ifelse(ind_daytime, "daytime", "not daytime")) %>%
  mutate(cat_survived = fct_rev(cat_survived)) %>%
```

Table 1: C.

sex	0	1
female	154	308
male	709	142

```
ggplot() +
aes(x = cat_survived) +
geom_bar()
```



Direct use of indicator variables in data products makes product more difficult or impossible to interpret.

Given how verbose recoding can be, analyst may choose to forego a recoding the variable, especially in exploratory analysis.

When indicator variables are not translated to a categorical analogue in creating data products like tables and visuals, information is often awkwardly displayed and is sometimes lost. When creating tables, using an indicator variable directly can be awkward or insufficient for interpretation.

Below, the column header from variable name and 0-1 categories preserves information but is awkward:

```
tidytitanic::passengers %>%
  count(survived)
```

```
      survived   n
    1        0 863
    2        1 450
```

In the following two-way table, information is completely lost:

```
tidytitanic::passengers %>%
  janitor::tabyl(sex, survived) %>%
  knitr::kable(caption = "C. ", format = kabel_format)
```

Furthermore in creating ba

```
library(tidyverse)

tidytitanic::passengers %>%
  ggplot() +
  aes(x = survived) +
  geom_bar()


tidytitanic::passengers %>%
ggplot() +
```

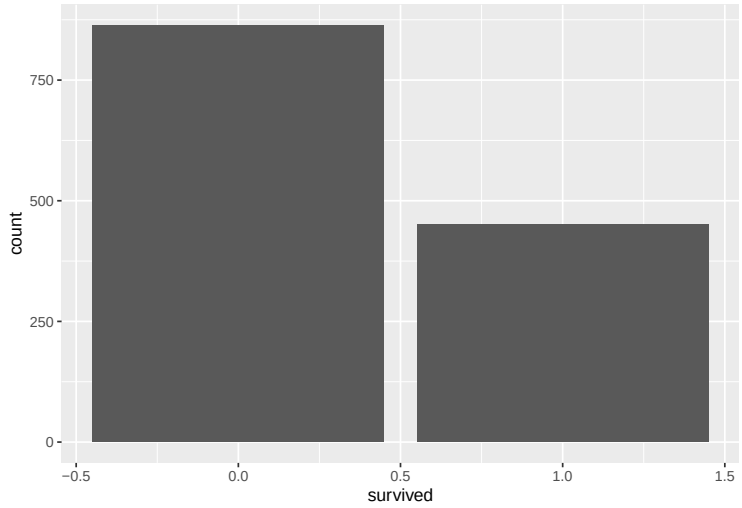


Figure 1: A. Bar labels + axis label preserves information but is awkward

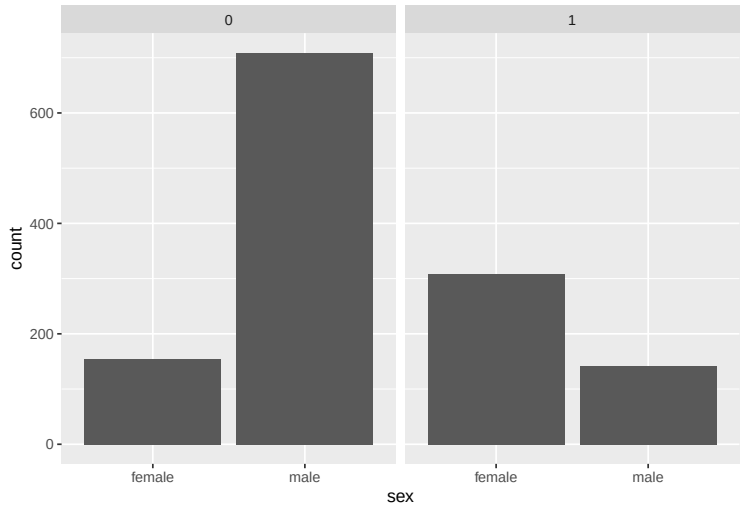


Figure 2: D. Facetting directly on an indicator variable with popular ggplot2 results in information loss

```
  aes(x = sex) +
  geom_bar() +
  facet_grid(~ survived)
```

3 Introducing ind_recode *ind_recode() function uses variable name as starting point for human-readable categories*

```
#' ind_recode
#'
#' @param var the name of an indicator variable
#' @param var_prefix a character string that will be ignored when creating the categorical variable
#' @param negator a character string used to create cat_false when cat_false is NULL, default is 'not'
#' @param cat_true a character string string to be used place of  T/1/"Yes" for the categorical variable outp
#' @param cat_false a character string string to be used place of  F/0/"No" for the categorical variable outp
#' @param rev logical indicating if the order should be reversed from the F/T ordering of the indicator sourc
#'
#' @return
#' @export
#'
#' @examples
#' library(tibble)
```

```
#' library(dplyr)
#' tibble(ind_grad = c(0,0,1,1,1 ,0 ,0)) %>%
#'   mutate(cat_grad  = ind_recode(ind_grad))
#'
#' tibble(ind_grad = c(TRUE,TRUE,FALSE)) %>%
#'   mutate(cat_grad  = ind_recode(ind_grad))
#'
#' tibble(ind_grad = c("Y", "N")) %>%
#'   mutate(cat_grad  = ind_recode(ind_grad))
#'
#' tibble(ind_grad = c("y", "n")) %>%
#'   mutate(cat_grad  = ind_recode(ind_grad))
#'
#' tibble(ind_grad = c("yes", "no")) %>%
#'   mutate(cat_grad  = ind_recode(ind_grad))
ind_recode <- function(var, var_prefix = "ind_", negator = "not",
                       cat_true = NULL, cat_false = NULL, rev = FALSE){

  if(is.null(cat_true)){
    cat_true = deparse(substitute(var)) %>%   # use r lang in rewrite
      stringr::str_remove(paste0("^", var_prefix)) %>%
      stringr::str_replace_all("_", " ")
  }

  if(is.null(cat_false)){
    cat_false = paste(negator, cat_true)
  }

  # for yes/no case
  if(is.character({{var}})){

    my_var <- {{var}} %>% as.factor() %>% as.numeric() - 1

  }else{

    my_var <- {{var}}
  }

  if(rev){
    ifelse(my_var, cat_true, cat_false) %>%
      factor(levels = c(cat_true, cat_false))
  }else{
    ifelse(my_var, cat_true, cat_false) %>%
      factor(levels = c(cat_false, cat_true))
  }


}
```

4 Basic examples: *How to use ind_recode()*

```
library(tibble)
tibble(ind_grad = c(0,0,1,1,1 ,0 ,0)) %>%
  mutate(cat_grad  = ind_recode(ind_grad))
```

```
    # A tibble: 7 x 2
      ind_grad cat_grad
         <dbl> <fct>
    1        0 not grad
    2        0 not grad
    3        1 grad
    4        1 grad
    5        1 grad
```

```
     6        0 not grad
     7        0 not grad

tibble(ind_grad = c(T,T,F)) %>%
  mutate(cat_grad  = ind_recode(ind_grad))

     # A tibble: 3 x 2
       ind_grad cat_grad
       <lgl>    <fct>
     1 TRUE     grad
     2 TRUE     grad
     3 FALSE    not grad

tibble(ind_grad = c("Y", "N")) %>%
  mutate(cat_grad  = ind_recode(ind_grad))

     # A tibble: 2 x 2
       ind_grad cat_grad
       <chr>    <fct>
     1 Y        grad
     2 N        not grad

tibble(ind_grad = c("y", "n")) %>%
  mutate(cat_grad  = ind_recode(ind_grad))

     # A tibble: 2 x 2
       ind_grad cat_grad
       <chr>    <fct>
     1 y        grad
     2 n        not grad

tibble(ind_grad = c("yes", "no")) %>%
  mutate(cat_grad  = ind_recode(ind_grad))

     # A tibble: 2 x 2
       ind_grad cat_grad
       <chr>    <fct>
     1 yes      grad
     2 no       not grad
```

5 Customizability

We believe that ind_recode is useful in quickly translating to a human understandable outcome.

However, addition functionality allows analysts to fully specify their preferences about the categories outputted.

- var_prefix a character string that will be ignored when creating the categorical variable
- negator a character string used to create cat_false when cat_false is NULL, default is 'not'
- cat_true a character string string to be used place of T/1/"Yes" for the categorical variable output, if NULL the category is automatically generated from the variable name
- cat_false a character string string to be used place of F/0/"No" for the categorical variable output, if NULL the category is automatically generated from cat_true and the negator
- rev logical indicating if the order should be reversed from the F/T ordering of the indicator source variable, default is FALSE

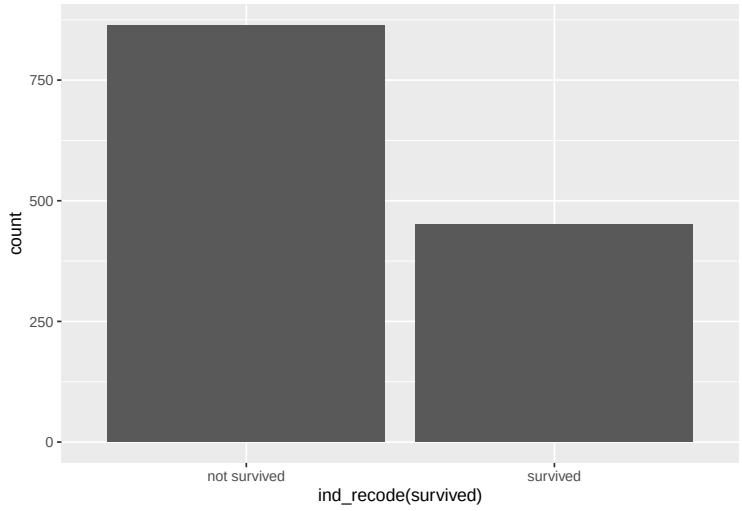
Customization examples

```
tibble(dummy_grad = c(0,0,1,1,1 ,0 ,0)) %>%
  mutate(cat_grad  = ind_recode(dummy_grad, var_prefix = "dummy_"))

     # A tibble: 7 x 2
       dummy_grad cat_grad
            <dbl> <fct>
     1          0 not grad
```

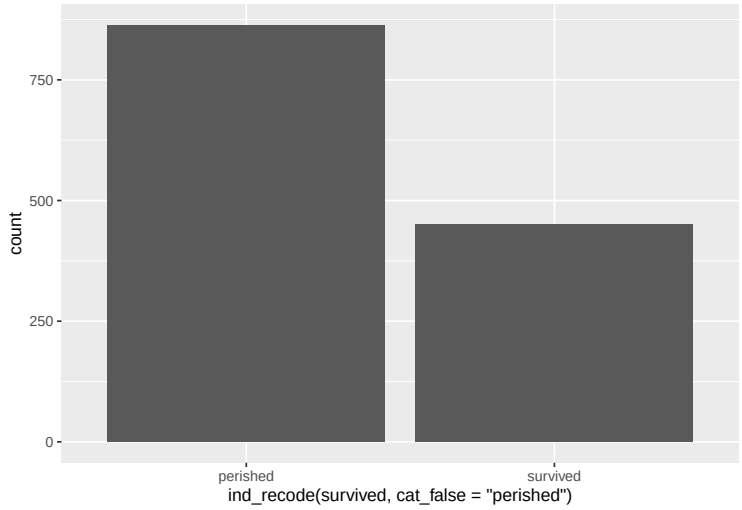
```
     2          0 not grad
     3          1 grad
     4          1 grad
     5          1 grad
     6          0 not grad
     7          0 not grad

tibble(ind_grad = c(T,T,F)) %>%
  mutate(cat_grad  = ind_recode(ind_grad, negator = "didn't"))

     # A tibble: 3 x 2
       ind_grad cat_grad
       <lgl>    <fct>
     1 TRUE     grad
     2 TRUE     grad
     3 FALSE    didn't grad

tibble(ind_grad = c("Y", "N")) %>%
  mutate(cat_grad  = ind_recode(ind_grad, cat_false = "enrolled"))

     # A tibble: 2 x 2
       ind_grad cat_grad
       <chr>    <fct>
     1 Y        grad
     2 N        enrolled

tibble(ind_grad = c("y", "n")) %>%
  mutate(cat_grad  = ind_recode(ind_grad,
                                cat_true = "graduated"))

     # A tibble: 2 x 2
       ind_grad cat_grad
       <chr>    <fct>
     1 y        graduated
     2 n        not graduated

tibble(ind_grad = c("y", "n")) %>%
  mutate(cat_grad  = ind_recode(ind_grad,
                                cat_true = "graduated",
                                cat_false = "enrolled"))

     # A tibble: 2 x 2
       ind_grad cat_grad
       <chr>    <fct>
     1 y        graduated
     2 n        enrolled

tibble(ind_grad = c("yes", "no")) %>%
  mutate(cat_grad  = ind_recode(ind_grad, rev = TRUE)) %>%
  mutate(cat_grad_num = as.numeric(cat_grad))

     # A tibble: 2 x 3
       ind_grad cat_grad cat_grad_num
       <chr>    <fct>           <dbl>
     1 yes      grad                1
     2 no       not grad            2
```

Use in data products like figures and tables

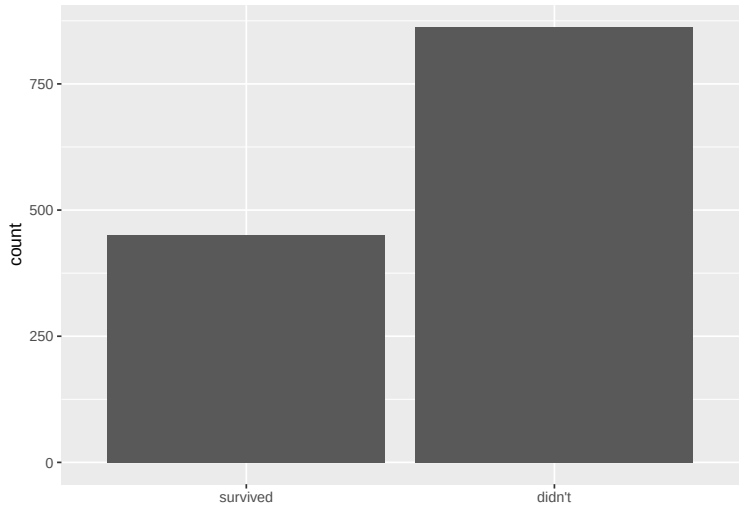
```
tidytitanic::passengers %>%
ggplot() +
  aes(x = ind_recode(survived)) +
  geom_bar()
```




```
# or
last_plot() +
  aes(x = ind_recode(survived, cat_false = "perished"))
```

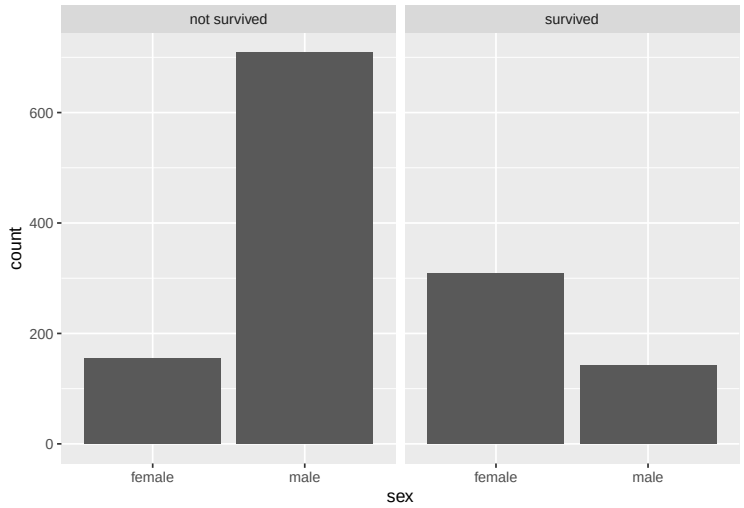



```
# or
last_plot() +
  aes(x = ind_recode(survived, cat_false = "didn't", rev = T)) +
  labs(x = NULL)
```



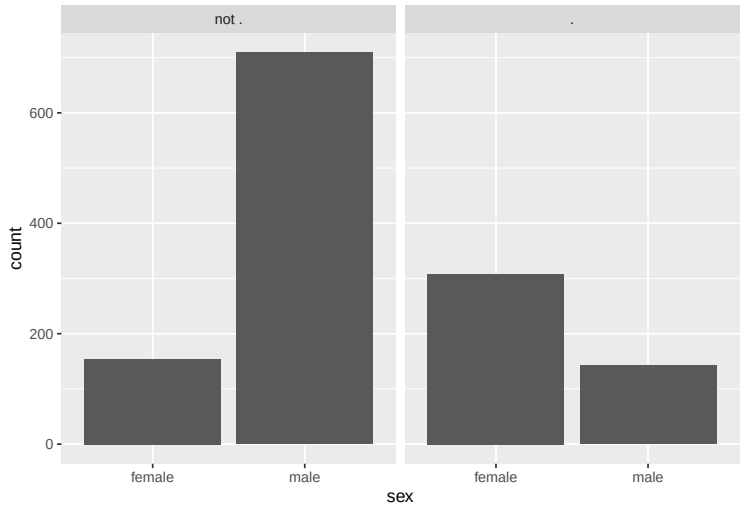

```
tidytitanic::passengers %>%
ggplot() +
```

```
  aes(x = sex) +
  geom_bar() +
  facet_grid(~ ind_recode(survived))
```

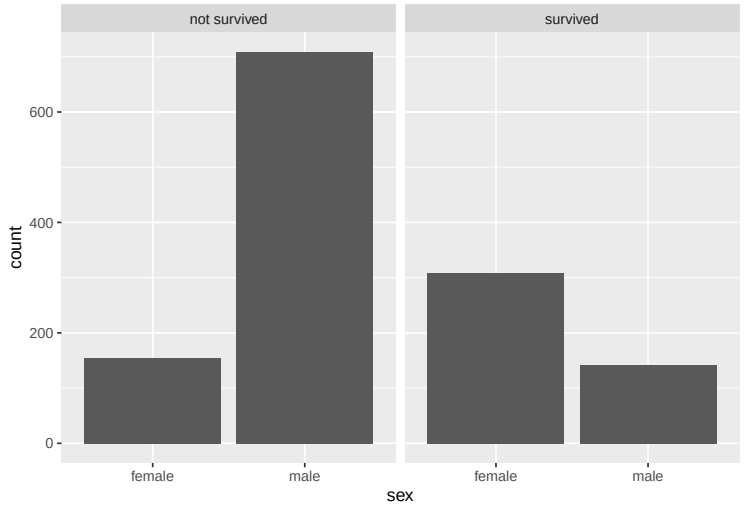


6 Known Limitations: *not for use with magrittr pipe (but base pipe works!)*

```
tidytitanic::passengers %>%
ggplot() +
  aes(x = sex) +
  geom_bar() +
  facet_grid(~ survived %>% ind_recode())
```



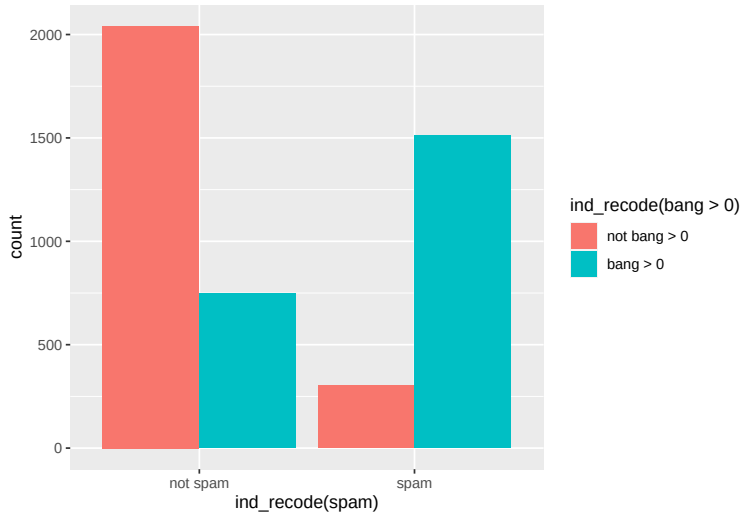

```
tidytitanic::passengers %>%
ggplot() +
  aes(x = sex) +
  geom_bar() +
  facet_grid(~ survived |> ind_recode())
```



7 worked example with tidytuesday data, Spam email

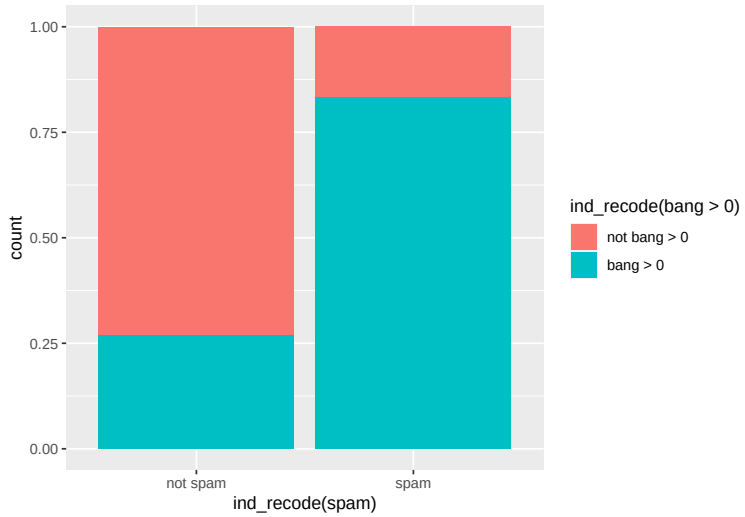
https://github.com/rfordatascience/tidytuesday/tree/master/data/2023/2023-08-15

```
read.csv("https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2023/2023-08-15/spam.csv"
  rename(spam = yesno) %>%
  ggplot() +
  aes(fill = ind_recode(bang>0), x = ind_recode(spam)) +
  geom_bar(position = "dodge")
```




```
remove_layers <- function(plot, index = NULL){

  if(is.null(index)){
  plot$layers <- NULL
  }else{
  plot$layers[[index]] <- NULL
  }

 plot

}

last_plot_wiped <- function(index = NULL){

  plot <- last_plot()
```

```
  if(is.null(index)){
  plot$layers <- NULL
  }else{
  plot$layers[[index]] <- NULL
  }

 plot

}

last_plot_wiped() +
  geom_bar(position = "fill")
```



8 learned along the way: `as_factor()` has different behavior than `as.factor()`

```
c("Y", "N") %>% as_factor()

     [1] Y N
     Levels: Y N

c("Y", "N") %>% as.factor()

     [1] Y N
     Levels: N Y

# unlink("../../temp.Rmd")

rmd_parse <- function(file = "../../README.Rmd"){

  readLines(file) %>%
    data.frame(text = .) %>%
    dplyr::mutate(ind_section_header = stringr::str_detect(text, "^#"), .before = 1) %>%
    dplyr::mutate(num_section_level = stringr::str_count(stringr::str_extract(text, "^#+"),"#") ) %>%
    dplyr::mutate(num_section = cumsum(ind_section_header)) %>%
    # dplyr::filter(ind_section_header) %>%
    dplyr::mutate(name_section = ifelse(ind_section_header, text, NA) %>% stringr::str_remove("#+ ") %>% tolo
  parsed

  parsed %>%
    dplyr::group_by(num_section) %>%
    dplyr::summarize(text %>% paste0(collapse = "\\n"))


}
```

Evangeline Reynolds
Affiliation
line 1
line 2
https://journal.r-project.org
ORCiD: 0000-0002-9079-593X
author1@work