The Journal of Open Source Software

dropout: an R Package for Addressing Dropouts, Missing Values, and Sectional Challenges in Survey Data Analysis

Hendrik Mann [1]

1 University of Wuppertal, Germany

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Samuel Forbes

Reviewers:

- @medewitt
- @max-alletsee
- @Darthpathos

Submitted: 20 November 2023
Published: unpublished

Summary

Missing data can reduce the statistical power of a study and can produce biased estimates, leading to invalid conclusions (Kang, 2013). When working with survey data, capturing and categorizing missing values is crucial for maintaining data quality. The dropout package assists in distinguishing whether missing values are isolated instances, sectional missing values, or complete dropouts. This distinction enables effective data cleaning and provides insights into the study design and the response behaviors of participants.

Statement of need

`dropout` is an R package (R Core Team, 2022) available on CRAN, designed to differentiate various types of missing values in survey data. It enables users to identify whether missing values are due to complete dropouts—participants ceasing to answer the questionnaire entirely, participants skipping entire sections, or isolated instances of 'NA' values. This distinction is crucial for accurate data analysis and understanding participant engagement in surveys. Beyond creating summary statistics, the package adeptly identifies participants who have either completely dropped out, skipped sections, or left specific questions unanswered. This capability not only facilitates thorough data cleaning but also provides insights into the various factors influencing dropout behavior. The `dropout` package's scalability is enhanced by the integration of C++ code through the Rcpp package, developed by Eddelbuettel et al. (2023). This incorporation makes `dropout` particularly effective for analyzing large survey datasets.

Usage

`dropout` includes two essential functions, beginning with `drop_summary`. This function is designed to generate summary statistics for missing values in your dataset. It requires two primary inputs: the dataframe to analyze and the identifier of the last survey item. The latter is particularly important in cases where your data includes additional columns generated by the survey platform, such as participation time. These extra items can lead to biased results in the function, as no dropouts may be detected if the last column has no missing values. To counter this, if the last_col argument is left unspecified, `drop_summary` will automatically address the issue by setting `last_col` to the last column in the dataframe that contains missing values, and it will issue a warning to inform the user. For optimal usage of the `dropout` package, it's generally good practice to ensure your dataframe either exclusively contains the survey items, or that the `last_col` argument is manually set to the last survey item in your dataset. Generally, it is essential to index all survey items in the dataframe in their correct order. An optional argument of the `drop_summary` function is `section_min`. This parameter plays a

The Journal of Open Source Software

crucial role when `drop_summary` is used to detect sections that participants may have skipped. By default, it looks for at least three consecutive missing items to identify a skipped section. This threshold seems to be sensitive enough to differentiate between single missing values and section omissions. However, it's advisable to experiment with different `section_min` settings to find the optimal threshold that aligns with your specific study design.

`drop_detect` works similarly to `drop_summary` and requires the same input arguments. However, instead of generating summary statistics for each question in your dataframe, drop_detect focuses on identifying dropouts for each participant. The output includes a logical column indicating whether a participant dropped out, and if so, it specifies the question and the index in the dataframe where this occurred. This functionality allows for the filtering of dropouts based on specific columns or indexes. Furthermore, the `last_col` argument in `drop_detect` is particularly useful for identifying participants who skipped specific sections. This can be achieved either by creating a subset of the dataframe containing only the items from that section or by setting the last_col argument to the last item of the section. These and other applications can be easily integrated with verbs from the tidyverse (Wickham et al., 2019).

Examples

For the following examples we will use an adapted version of the Flying Etiquette by five-thirty-eight dataset that is included in the `dropout` package. In these workflow examples, I will be using dplyr verbs (Wickham et al., 2023) - although this is not necessary.

As illustrated with the `flying` dataset example, even though all columns are arranged in the correct order of survey items, the last column survey_type does not correspond to a survey item. In such scenarios, the dropout package intuitively addresses this issue by disregarding the non-survey column and automatically setting the last_col argument to `location_census_region`. This adjustment is accompanied by a warning to inform the user. However, in more complex situations, it's advisable to either create a subset of your data or manually set the last_col argument to the actual last survey item. We will demonstrate this approach in the following examples.

```r
# install.packages(c("dropout", "tidyverse"))

library(dropout)
library(tidyverse)
data(flying)
```

Initially, we use the `drop_summary` function to generate an overview of the different types of missing values in our dataframe. From this analysis, it becomes evident that certain parts of the survey experience higher dropout rates. Notably, 18 participants dropped out early, at the third survey item, culminating in a total of 42 dropouts by the end of the survey. The ´section_na´ column reveals that 164 participants skipped an entire section of the questionnaire, or at least a consecutive portion identifiable as a section in this context. Furthermore, single missing values are particularly prevalent in responses to the household income question.

```r
flying %>%
drop_summary(last_col = "location_census_region") %>%
print(n = Inf)
```

In the subsequent step, we aim to refine our dataset to include only those survey participants who did not experience an early dropout at the third question and who completed the survey without any dropouts. To achieve this, we utilize the drop_detect function, which identifies participants according to dropout status at specified points within the survey. By merging the output with our original data, we can then apply a filter to retain only the desired respondents. Once filtered, we remove the additional columns introduced by `drop_detect` as they are no longer necessary for further analysis. While not demonstrated in this example, this method of indexing is particularly advantageous when we need to perform complex manipulations, such as excluding all participants who dropped out before reaching the tenth item in the survey.

```
flying_dropouts <- flying %>%
drop_detect(last_col = "location_census_region")

head(flying_dropouts)

flying_cleaned <- flying_dropouts %>%
bind_cols(flying) %>%
filter(dropout_col != "seat_recline" | dropout == FALSE) %>%
select(-starts_with("dropout"))
```

Next, we aim to exclude the 164 participants who skipped an entire section of the survey without fully dropping out. This can be accomplished through two approaches. The first method involves setting the `last_col` argument of the `drop_detect` function to the last column of the omitted section. By doing so, all participants who skipped the entire section will be flagged as dropouts, making it straightforward to exclude them. The second method requires creating a subset of the dataset that includes only the columns of the concerned section. This subset can then be used to specifically filter for section-based dropouts. It is important to note that when working with such a subset, the indices provided by `dropout_index` might correspond to the subset's dataframe and not the original one. This distinction is crucial for accurately mapping the dropout information back to the complete dataset, when using a subset starting not from column one of the origional dataset. In the method 2 this is not an issue.

```
# method 1 (recommended as indexes of dropout_index will still match the data)
flying_cleaned %>%
drop_detect(last_col = "smoking_violation") %>%
bind_cols(flying_cleaned) %>%
filter(dropout_col != "seat_recline" | dropout == FALSE) %>%
select(-starts_with("dropout"))

# method 2 (if the dropout_index will still match the data depends on the subset)
flying_cleaned %>%
select(1:22) %>%
drop_detect() %>%
bind_cols(flying_cleaned) %>%
filter(dropout_col != "seat_recline" | dropout == FALSE) %>%
select(-starts_with("dropout"))
```


In this article's concluding section, we explore the practical applications of these techniques in analyzing distinct dropout behaviors. Through the visualisation in Figure 1, we will contrast participants who omitted a particular survey section with those who partially completed it or had missing values that did not start at the beginning of the section. We will segment this comparative analysis by gender to illustrate how one might investigate varying dropout behaviors across different demographic groups. This approach exemplifies how data on dropout patterns can be dissected to yield insights into the participant experience and inform improvements in survey design.

```
# library(ggplot)

flying_section <- flying_cleaned %>%
select(3:22) %>%
drop_detect(last_col = "smoking_violation") %>%
bind_cols(flying_cleaned) %>%
filter(dropout_col == "seat_recline") %>%
count(gender) %>%
```

```r
rename(omitted = n)

figure1 <- flying_cleaned %>%
count(gender) %>%
rename(N = n) %>%
drop_na() %>%
left_join(flying_section) %>%
mutate(completed = N - omitted) %>%
pivot_longer(3:4, values_to = "n", names_to = "condition") %>%
  ggplot(aes(x = gender, y = n, fill = condition)) +
  geom_col(position = "dodge")
```

Both the drop_summary and drop_detect functions are designed for seamless integration into data analysis workflows and pipelines. These functions facilitate an easy visualization of their output. In the follwing example of Figure 2, we utilize the drop_summary function to visually represent missing values in the dataframe. The visualization distinctly categorizes the missing values into three types: dropouts, section_na (entire sections left out), and single_na (individual missing values).

```r
fig2 <- flying %>%
  drop_summary(last_col = "location_census_region") %>%
  select(column_name, dropout, section_na, single_na) %>%
  pivot_longer(-column_name, names_to = "missing", values_to = "values") %>%
  mutate(column_name = fct_inorder(column_name)) %>%
    ggplot(aes(x = column_name, y = values, group = missing, col = missing))+
    geom_line()+
    geom_point()+
    scale_x_discrete(guide = guide_axis(angle = 45))+
    xlab("items")+
    ylab("missing values")
```

References

Figures

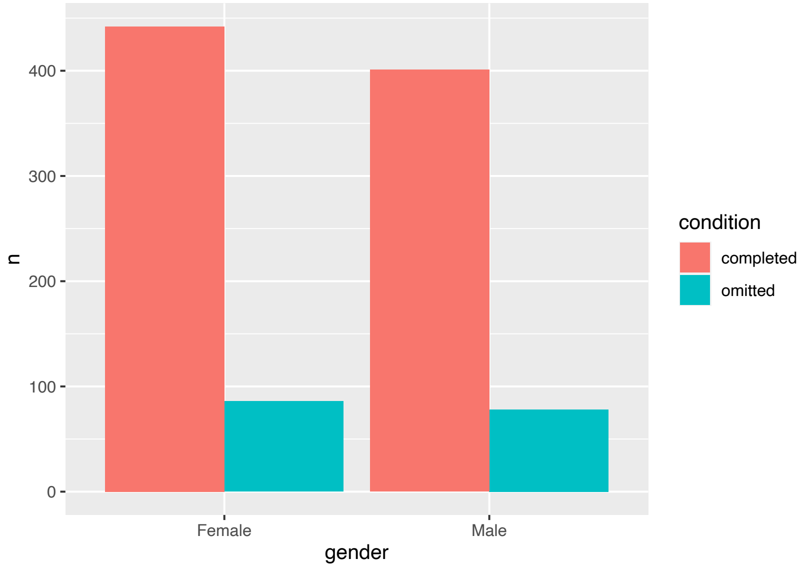

Figure 1: Analyzing Dropout Patterns and Missing Values with the 'dropout' Package. This graph illustrates the proportions of male and female participants who either omitted or completed the section of the survey from 'seat_recline' to 'smoking_validation'. It compares those who skipped this entire section (labeled as 'omitted') with those who either fully completed it or ceased responding past this section (labeled as 'completed').

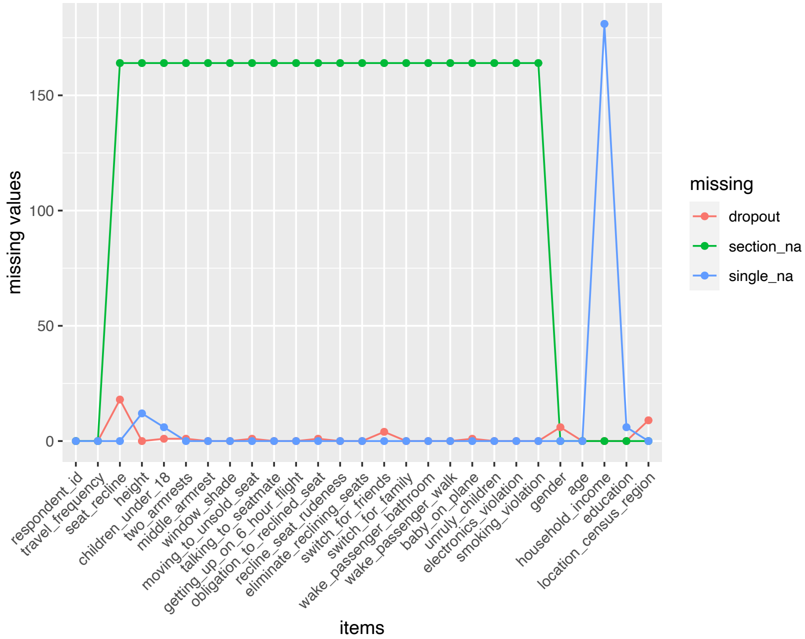

Figure 2: Visualization of Missing Values in the Flying Dataset. Note: A concentration of missing values is observed in the section ranging from 'seat_recline' to 'smoking_violation', as indicated by the 'section_na' category. Additionally, the 'household income' variable is notably omitted by a portion of the participants, categorized as 'single_na' values. This pattern highlights specific areas in the survey where participant engagement varies.

Eddelbuettel, D., Francois, R., Allaire, J., Ushey, K., Kou, Q., Russell, N., Ucar, I., Bates, D., & Chambers, J. (2023). *Rcpp: Seamless r and c++ integration*. https://CRAN.R-project.org/package=Rcpp

Kang, H. (2013). The prevention and handling of the missing data. *Korean Journal of Anesthesiology*, *64*(5), 402. https://doi.org/10.4097/kjae.2013.64.5.402

R Core Team. (2022). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., … Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, *4*(43), 1686. https://doi.org/10.21105/joss.01686

Wickham, H., François, R., Henry, L., Müller, K., & Vaughan, D. (2023). *Dplyr: A grammar of data manipulation*. https://dplyr.tidyverse.org