Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

Vazul: An R Package for Analysis Blinding

Tam'as Nagy
'ELTE Eotvos Lorend University'

Alexandra Sarafoglou
University of Amsterdam'

Abstract

The abstract of the article.

Keywords: keywords, not capitalized, Java.

0.1. Overview of the main functions

The package provides functions for two main types of analysis blinding:

1. **Masking**: Replaces original values with anonymous labels, completely hiding the original information.
2. **Scrambling**: Randomizes the order of existing values while preserving all original data content.

Each approach is available at three levels:

- **Vector level**: `mask_labels()` and `scramble_values()` - operate on single vectors
- **Data frame level**: `mask_variables()` and `scramble_variables()` - operate on columns in a data frame
- **Row-wise level**: `mask_variables_rowwise()` and `scramble_variables_rowwise()` - operate within rows across columns

```
R> library(vazul)
R> library(dplyr)
R>
R> set.seed(123)
```

0.2. Included datasets

The `vazul` package includes two research datasets for demonstration and practice. The `marp` dataset contains cross-national survey data on religiosity, while the `williams` dataset contains experimental data from a stereotyping study.

```
R> data(marp)
R> data(williams)
R>
R> glimpse(marp)

Rows: 10,535
Columns: 46
$ subject         <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,~
$ country         <chr> "Australia", "Australia", "Australia", "Australia", "A~
$ rel_1           <dbl> 0.0000000, 0.8333333, 0.0000000, 0.0000000, 0.0000000,~
$ rel_2           <dbl> 0.0000000, 0.7142857, 0.0000000, 0.0000000, 0.0000000,~
$ rel_3           <dbl> 0.5, 1.0, 0.5, 0.0, 0.0, 0.5, 0.5, 0.0, 0.0, 0.0, 0.5,~
$ rel_4           <int> 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, ~
$ rel_5           <dbl> 0.1666667, 0.5000000, 0.1666667, 0.5000000, 0.1666667,~
$ rel_6           <dbl> 0.5000000, 0.5000000, 0.0000000, 0.6666667, 0.0000000,~
$ rel_7           <dbl> 0.5000000, 0.3333333, 0.1666667, 0.3333333, 0.3333333,~
$ rel_8           <dbl> 0.00, 0.50, 0.00, 0.25, 0.00, 0.25, 0.00, 0.00, 0.00, ~
$ rel_9           <dbl> 0.50, 0.25, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, ~
$ cnorm_1         <dbl> 0.50, 0.25, 0.25, 0.25, 0.50, 0.50, 0.25, 0.50, 0.00, ~
$ cnorm_2         <dbl> 0.50, 0.25, 0.25, 0.75, 0.50, 0.50, 0.25, 0.25, 0.00, ~
$ wb_gen_1        <int> 4, 4, 3, 4, 3, 4, 4, 3, 4, 4, 3, 5, 4, 4, 1, 4, 4, 3, ~
$ wb_gen_2        <int> 2, 3, 4, 3, 4, 4, 4, 2, 4, 4, 3, 5, 4, 4, 5, 4, 4, 2, ~
$ wb_phys_1       <int> 3, 3, 4, 4, 5, 5, 4, 3, 4, 4, 3, 4, 5, 4, 3, 4, 4, 4, ~
$ wb_phys_2       <int> 3, 3, 5, 3, 5, 5, 5, 4, 3, 4, 3, 1, 5, 5, 4, 3, 5, 4, ~
$ wb_phys_3       <int> 3, 4, 2, 2, 3, 4, 2, 1, 4, 2, 3, 5, 4, 5, 4, 4, 4, 1, ~
$ wb_phys_4       <int> 4, 5, 5, 3, 4, 4, 4, 3, 4, 3, 3, 4, 4, 5, 5, 3, 4, 4, ~
$ wb_phys_5       <int> 2, 2, 4, 2, 4, 4, 2, 2, 3, 3, 3, 4, 4, 3, 4, 4, 4, 1, ~
$ wb_phys_6       <int> 3, 4, 3, 3, 5, 4, 3, 3, 4, 2, 3, 5, 4, 4, 5, 4, 5, 3, ~
$ wb_phys_7       <int> 2, 4, 4, 3, 4, 4, 3, 2, 4, 2, 3, 5, 4, 5, 4, 3, 5, 3, ~
$ wb_psych_1      <int> 1, 2, 3, 3, 3, 3, 3, 1, 3, 3, 3, 5, 4, 4, 4, 4, 3, 3, ~
$ wb_psych_2      <int> 1, 4, 2, 2, 2, 3, 2, 1, 2, 2, 3, 4, 4, 1, 4, 4, 5, 3, ~
$ wb_psych_3      <int> 3, 3, 3, 4, 4, 4, 3, 2, 3, 3, 3, 4, 4, 3, 4, 4, 4, 2, ~
$ wb_psych_4      <int> 3, 3, 1, 3, 4, 4, 4, 1, 1, 4, 3, 5, 4, 5, 4, 3, 4, 2, ~
$ wb_psych_5      <int> 2, 3, 2, 2, 2, 4, 3, 1, 4, 3, 3, 5, 4, 5, 5, 4, 5, 2, ~
$ wb_psych_6      <int> 3, 3, 2, 4, 3, 4, 3, 1, 2, 3, 4, 4, 4, 4, 4, 4, 5, 3, ~
$ wb_soc_1        <int> 3, 3, 3, 4, 2, 4, 4, 1, 2, 4, 3, 4, 4, 4, 5, 5, 4, 4, ~
$ wb_soc_2        <int> 1, 3, 1, 3, 3, 3, 4, 2, 2, 4, 3, 3, 4, 3, 4, 4, 3, 3, ~
$ wb_soc_3        <int> 3, 4, 3, 4, 1, 4, 1, 3, 2, 1, 4, 5, 3, 2, 5, 5, NA, 4,~
$ wb_overall_mean <dbl> 2.555556, 3.333333, 3.000000, 3.111111, 3.388889, 3.94~
$ wb_phys_mean    <dbl> 2.857143, 3.571429, 3.857143, 2.857143, 4.285714, 4.28~
$ wb_psych_mean   <dbl> 2.166667, 3.000000, 2.166667, 3.000000, 3.000000, 3.66~
$ wb_soc_mean     <dbl> 2.333333, 3.333333, 2.333333, 3.666667, 2.000000, 3.66~
$ age             <int> 41, 31, 26, 51, 21, 28, 49, 23, 23, 18, 22, 20, 42, 18~
```

```
$ gender          <chr> "man", "man", "woman", "man", "man", "woman", "man", "~
$ ses             <int> 4, 6, 5, 7, 5, 4, 5, 5, 4, 7, 5, 7, 6, 7, 5, 8, 6, 5, ~
$ education       <int> 4, 5, 5, 2, 5, 5, 3, 3, 2, 4, 5, 4, 3, 5, 5, 3, 5, 4, ~
$ ethnicity       <chr> "East Asian", "East Asian", "Caucasian/European", "Cau~
$ denomination    <chr> NA, "Christian (Roman Catholic)", NA, NA, NA, "Buddhis~
$ gdp             <dbl> 57305.3, 57305.3, 57305.3, 57305.3, 57305.3, 57305.3, ~
$ gdp_scaled      <dbl> 1.001802, 1.001802, 1.001802, 1.001802, 1.001802, 1.00~
$ sample_type     <chr> "online panel", "online panel", "online panel", "onlin~
$ compensation    <chr> "monetary reward", "monetary reward", "monetary reward~
$ attention_check <int> 1, 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
```

```
R> glimpse(williams)
```

```
Rows: 112
Columns: 25
$ subject             <chr> "A30MP4LXV4MIFD", "A16X5FB3HAFCKN", "A1E9D1OT9VJYD~
$ SexUnres_1          <dbl> 5, 7, 2, 5, 5, 5, 5, 4, 6, 4, 5, 3, 5, 6, 3, 6, 2,~
$ SexUnres_2          <dbl> 3, 7, 4, 4, 5, 6, 6, 6, 5, 4, 5, 2, 5, 3, 2, 3, 1,~
$ SexUnres_3          <dbl> 2, 7, 6, 5, 6, 6, 5, 5, 6, 4, 7, 3, 5, 3, 3, 6, 3,~
$ SexUnres_4_r        <dbl> 3, 4, 3, 3, 3, 3, 3, 2, 3, 1, 3, 3, 3, 2, 3, 3, 4,~
$ SexUnres_5_r        <dbl> 2, 2, 3, 3, 3, 2, 4, 3, 2, 1, 2, 2, 4, 6, 3, 1, 3,~
$ Impuls_1            <dbl> 3, 6, 3, 4, 5, 5, 5, 6, 6, 1, 6, 2, 4, 7, 5, 4, 5,~
$ Impuls_2_r          <dbl> 3, 2, 3, 4, 2, 3, 3, 2, 3, 1, 3, 2, 3, 6, 3, 3, 7,~
$ Impul_3_r           <dbl> 2, 1, 3, 4, 4, 3, 2, 3, 2, 1, 4, 1, 4, 5, 3, 3, 5,~
$ Opport_1            <dbl> 1, 5, 3, 4, 4, 4, 5, 5, 5, 1, 7, 3, 5, 7, 5, 2, 6,~
$ Opport_2            <dbl> 2, 7, 5, 4, 6, 3, 5, 4, 4, 4, 6, 4, 6, 5, 2, 3, 4,~
$ Opport_3            <dbl> 2, 7, 5, 4, 4, 4, 6, 5, 5, 1, 7, 3, 5, 5, 5, 3, 1,~
$ Opport_4            <dbl> 3, 6, 3, 5, 4, 6, 5, 5, 4, 4, 6, 3, 6, 4, 4, 3, 3,~
$ Opport_5            <dbl> 1, 6, 3, 4, 5, 4, 6, 6, 4, 1, 6, 3, 6, 5, 5, 4, 6,~
$ Opport_6_r          <dbl> 3, 2, 3, 4, 3, 3, 2, 2, 3, 2, 1, 3, 1, 6, 4, 3, 4,~
$ InvEdu_1_r          <dbl> 2, 3, 2, 3, 4, 3, 3, 3, 3, 2, 3, 4, 4, 4, 3, 1, 6,~
$ InvEdu_2_r          <dbl> 3, 2, 3, 4, 3, 4, 2, 3, 4, 1, 2, 2, 3, 7, 4, 1, 3,~
$ InvChild_1          <dbl> 2, 5, 6, 5, 5, 5, 6, 5, 6, 1, 5, 2, 4, 4, 5, 2, 6,~
$ InvChild_2_r        <dbl> 3, 2, 3, 4, 3, 4, 2, 4, 3, 2, 4, 2, 3, 7, 3, 2, 6,~
$ age                 <dbl> 34, 30, 40, 35, 26, 33, 33, 30, 48, 33, 40, 39, 25~
$ gender              <dbl> 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 2, 1,~
$ ecology             <chr> "Hopeful", "Desperate", "Desperate", "Hopeful", "D~
$ duration_in_seconds <dbl> 164, 100, 47, 31, 40, 32, 98, 34, 32, 87, 71, 103,~
$ attention_1         <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,~
$ attention_2         <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,~
```

0.3. Masking functions

Masking functions replace categorical values with anonymous labels. This is useful when you want to hide the original information, such as treatment conditions or group assignments. Masking variables is useful when there are a limited number of unique values. The

`mask_labels()` function takes a character or factor vector and replaces each unique value with a randomly assigned masked label. The function preserves factor structure when the input is a factor. After masking, each unique value receives a unique masked label, the same original value always maps to the same masked label, and the assignment of masked labels to original values is randomized.

```
R> # Create a simple treatment vector
R> treatment <- c("control", "treatment", "control", "treatment", "control")
R>
R> # Mask the labels
R> mask_labels(treatment)

          control         treatment           control         treatment
"masked_group_01" "masked_group_02" "masked_group_01" "masked_group_02"
          control
"masked_group_01"
```

It is possible to customize the prefix used for masked labels:

```
R> mask_labels(treatment, prefix = "group_")

   control  treatment    control  treatment    control
"group_01" "group_02" "group_01" "group_02" "group_01"
```

The `mask_variables()` function extends the masking functionality to multiple columns in a data frame simultaneously. It is possible to use tidyelect helpers to select columns.

```
R> marp |>
+     select(rel_1:rel_9, country, denomination) |>
+     mask_variables(c("country", "denomination")) |>
+     head()

      rel_1     rel_2 rel_3 rel_4     rel_5     rel_6     rel_7 rel_8 rel_9
1 0.0000000 0.0000000   0.5     0 0.1666667 0.5000000 0.5000000  0.00  0.50
2 0.8333333 0.7142857   1.0     1 0.5000000 0.5000000 0.3333333  0.50  0.25
3 0.0000000 0.0000000   0.5     0 0.1666667 0.0000000 0.1666667  0.00  0.00
4 0.0000000 0.0000000   0.0     0 0.5000000 0.6666667 0.3333333  0.25  0.00
5 0.0000000 0.0000000   0.0     0 0.1666667 0.0000000 0.3333333  0.00  0.00
6 0.5000000 0.0000000   0.5     1 0.6666667 0.0000000 1.0000000  0.25  0.00
           country          denomination
1 country_group_03                  <NA>
2 country_group_03 denomination_group_13
3 country_group_03                  <NA>
4 country_group_03                  <NA>
5 country_group_03                  <NA>
6 country_group_03 denomination_group_07
```

```
R> marp |>
+     select(rel_1:rel_9, country, denomination) |>
+     mask_variables(where(is.character)) |>
+     head()

      rel_1     rel_2 rel_3 rel_4     rel_5     rel_6     rel_7 rel_8 rel_9
1 0.0000000 0.0000000   0.5     0 0.1666667 0.5000000 0.5000000  0.00  0.50
2 0.8333333 0.7142857   1.0     1 0.5000000 0.5000000 0.3333333  0.50  0.25
3 0.0000000 0.0000000   0.5     0 0.1666667 0.0000000 0.1666667  0.00  0.00
4 0.0000000 0.0000000   0.0     0 0.5000000 0.6666667 0.3333333  0.25  0.00
5 0.0000000 0.0000000   0.0     0 0.1666667 0.0000000 0.3333333  0.00  0.00
6 0.5000000 0.0000000   0.5     1 0.6666667 0.0000000 1.0000000  0.25  0.00
           country          denomination
1 country_group_01                  <NA>
2 country_group_01 denomination_group_14
3 country_group_01                  <NA>
4 country_group_01                  <NA>
5 country_group_01                  <NA>
6 country_group_01 denomination_group_03
```

By default, each column gets its own set of masked labels with the column name as prefix. When `across_variables = TRUE`, all selected columns share the same mapping. This can be useful when the same conditions appear in multiple columns.

```
R> df <- data.frame(
+   pre_condition = c("A", "B", "C", "A"),
+   post_condition = c("B", "A", "A", "C"),
+   score = c(1, 2, 3, 4)
+ )
R>
R> mask_variables(df, c("pre_condition", "post_condition"),
+                                 across_variables = TRUE)

    pre_condition  post_condition score
1 masked_group_02 masked_group_03     1
2 masked_group_03 masked_group_02     2
3 masked_group_01 masked_group_02     3
4 masked_group_02 masked_group_01     4
```

The `mask_variables_rowwise()` function applies consistent masking within each row across multiple columns. This is useful in case when of categrical data that is repeated across columns, such as treatment conditions or item responses. Each row gets its own independent mapping of original values to masked labels.

```
R> df <- data.frame(
+   treat_1 = c("control", "treatment_1", "treatment_2", "control"),
```

```
+   treat_2 = c("treatment_1", "treatment_2", "control", "control"),
+   treat_3 = c("treatment_2", "control", "treatment_1", "control"),
+   id = 1:4
+ )
R>
R> mask_variables_rowwise(df, starts_with("treat_"))

          treat_1         treat_2         treat_3 id
1 masked_group_02 masked_group_03 masked_group_01  1
2 masked_group_03 masked_group_01 masked_group_02  2
3 masked_group_01 masked_group_02 masked_group_03  3
4 masked_group_02 masked_group_02 masked_group_02  4
```

0.4. Scrambling functions

Scrambling functions randomize the order of values while preserving all original data content. This approach maintains the data distribution while breaking the connection between observations and their original values. The `scramble_values()` function randomly reorders the elements of a vector. The vector can contain numeric, character, or factor data. Scrambling is useful when you want to preserve the original values but eliminate any correspondence between observations and their values.

```
R> # Numeric data
R> numbers <- 1:10
R> scramble_values(numbers)

 [1]  4  1  6  8  7  3 10  2  9  5
```

The `scramble_variables()` function extends the scrambling functionality to data frames. It allows scrambling multiple columns simultaneously, with options for independent scrambling, joint scrambling, and within-group scrambling. Columns can be selected using tidyselect helpers.

```
R> df <-
+     williams |>
+     select(subject, age, ecology) |>
+     head()
R>
R> scramble_variables(df, c("age", "ecology"))

# A tibble: 6 x 3
  subject          age ecology
  <chr>          <dbl> <chr>
1 A3OMP4LXV4MIFD    34 Desperate
2 A16X5FB3HAFCKN    40 Desperate
3 A1E9D1OT9VJYDZ    26 Desperate
```

```
4 A16FPOYD7566WI    33 Hopeful
5 A11NOTVHWST7Y3    35 Hopeful
6 A3TDR6MXS6UO5Z    30 Desperate
```

By default, columns are scrambled independently of each other. When `together = TRUE`, the selected columns are scrambled as a unit, preserving row-level relationships:

```
R> df  <-
+     marp |>
+     select(subject, country, rel_1:rel_3)
R>
R> tail(df)

      subject country     rel_1     rel_2 rel_3
10530   10530      US 0.8333333 0.7142857   1.0
10531   10531      US 0.0000000 0.0000000   0.5
10532   10532      US 0.0000000 0.0000000   0.0
10533   10533      US 0.1666667 1.0000000   1.0
10534   10534      US 0.0000000 0.0000000   0.0
10535   10535      US 0.8333333 0.7142857   1.0

R> scramble_variables(df, starts_with("rel_"), together = TRUE) |>
+     tail()

      subject country     rel_1     rel_2 rel_3
10530   10530      US 0.0000000 0.0000000   0.5
10531   10531      US 0.3333333 0.0000000   0.5
10532   10532      US 0.0000000 0.0000000   0.0
10533   10533      US 0.0000000 0.0000000   0.5
10534   10534      US 0.0000000 0.8571429   1.0
10535   10535      US 0.0000000 0.0000000   0.5
```

Scrambling can be done in groups using the `.groups` parameter. This ensures that values are only scrambled within their original groups.

```
R> df |>
+     scramble_variables(starts_with("rel_"), .groups = "country")

# A tibble: 10,535 x 5
   subject country   rel_1 rel_2 rel_3
     <int> <chr>     <dbl> <dbl> <dbl>
 1      85 Australia 0.667 0       0
 2     300 Australia 0     1       0
 3     218 Australia 0     0       1
 4     168 Australia 0     0.571   1
 5     178 Australia 0     1       1
```

```
 6     102 Australia 0.833 1       1
 7     465 Australia 0     0.857   1
 8     240 Australia 0     0.429   0.5
 9     120 Australia 0.833 1       0.5
10     221 Australia 0.167 0.714   0.5
# i 10,525 more rows
```

The `scramble_variables_rowwise()` function scrambles values within each row across specified columns. This is useful for scrambling repeated measures or item responses.

```
R> df <- data.frame(
+   item1 = c(1, 4, 7),
+   item2 = c(2, 5, 8),
+   item3 = c(3, 6, 9),
+   id = 1:3
+ )
R>
R> scramble_variables_rowwise(df, c("item1", "item2", "item3"))

  item1 item2 item3 id
1     3     1     2  1
2     6     4     5  2
3     8     7     9  3
```

Within each row, the values are shuffled among the item columns. You can scramble multiple sets of columns independently. The function can use tidyselect helpers.

```
R> df2 <- data.frame(
+   day_1 = c(1, 4, 7),
+   day_2 = c(2, 5, 8),
+   day_3 = c(3, 6, 9),
+   score_a = c(10, 40, 70),
+   score_b = c(20, 50, 80),
+   id = 1:3
+ )
R>
R> scramble_variables_rowwise(df2, starts_with("day_"), c("score_a", "score_b"))

  day_1 day_2 day_3 score_a score_b id
1     3     1     2      20      10  1
2     5     4     6      50      40  2
3     7     8     9      70      80  3
```

0.5. Choosing between masking and scrambling

Aspect	Masking	Scrambling
Original values	Hidden (replaced)	Preserved (reordered)
Distribution	Same proportion, new labels	Unchanged
Best for	Categorical variables	Numeric or categorical
Use case	Hide treatment conditions	Break correlations

Use Masking:

- When you need to hide categorical labels (e.g., treatment conditions, group names)
- When analysts should not know the meaning of categories
- When you want different prefixes for different variables

Use Scrambling:

- When you want to preserve the original data distribution
- When you need to break the association between variables of interest (e.g., treatment and outcome)
- When working with numeric data that shouldn't be categorically relabeled

0.6. Functions

The `vazul` package provides a comprehensive toolkit for data blinding:

Function	Level	Purpose
`mask_labels()`	Vector	Replace categorical values with anonymous labels
`mask_variables()`	Data frame	Mask multiple columns
`mask_variables_rowwise()`	Row-wise	Consistent masking within rows
`scramble_values()`	Vector	Randomize value order
`scramble_variables()`	Data frame	Scramble multiple columns
`scramble_variables_rowwise()`	Row-wise	Scramble values within rows

Affiliation:

Tam'as Nagy
'ELTE Eotvos Lorend University'
'Institute of Psychology,
ELTE Eotvos Lorend University,
Budapest, Hungary'
E-mail: nagy.tamas@ppk.elte.hu

Alexandra Sarafoglou
University of Amsterdam'
Department of Psychology,
University of Amsterdam,
Amsterdam, The Netherlands'

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd