Package 'misty'

Takuya Yanagida

February 9, 2020

Contents

1 misty: Miscellaneous Functions T. Yanagida

The `misty` package provides miscellaneous functions for descriptive statistics, missing data, data management, and statistical analysis, e.g., reading and writing a SPSS file, frequency table, cross tabulation, multilevel and missing data descriptive statistics, various effect size measures, scale and group scores, centering at the grand mean or within cluster, intraclass correlation coefficient, or coefficient alpha and item statistics.

1.1 Installation

The `misty` package is published on the Comprehensive R Archive Network (CRAN) and can be installed by using the `install.packages()` function:

```
> install.packages("misty", repos = "https://cloud.r-project.org")
 Installing package into 'C:/.../R/win-library/3.6' (as 'lib' is unspecified)
 package 'misty' successfully unpacked and MD5 sums checked
```

After installation, the `misty` package can be loaded by using the `library()` function:

```
> library(misty)
 |-------------------------------------|
 | misty 0.2.1 (2020-02-02)            |
 | Miscellaneous Functions T. Yanagida |
 |-------------------------------------|
```

1.2 Introduction

R is a powerful software environment and programming language designed for data manipulation, statistical computing, and graphics and is based on a package system which allows users to contribute functions, documentations and data sets to extend R. The R base system comprises seven pre-installed packages which are automatically loaded each R session and provides a variety of standard statistical methods. There are over 15,000 additional packages on CRAN offering a broad range of statistical methods like latent variable modeling (e.g., R package `lavaan`), missing data imputation (e.g., R package `mice`), or item response modeling (e.g., R package `TAM`). In order to use an package not included in the R base system, the package needs to be installed once, but loaded each time R is started by using the `library()` function. For example, in data management and descriptive statistics, following functions from various R packages might be needed:

- The `read.spss()` function from the `foreign` for reading a SPSS file.
- The `recode()` function from the `car` package for recoding a variable.
- The `gmc()` function from the `rockchalk` package for centering a predictor within cluster.
- The `skewness()` function from the `moments` package for computing skewness of a variable.
- The `alpha()` function from the `psych` package for computing coefficient alpha.
- The `cohen.d()` function from the `effsize` package for computing Cohen's *d*.

The R package system is the main advantage of R resulting in a widespread availability of statistical methods from various fields of research (see the CRAN Task Views). One disadvantage of R frequently mentioned is the steep learning curve in particular for people who are used to a point-and-click software environment (e.g., SPSS). One of the main challenges in learning R stem from the fact that the R base system does not fully cover all functions commonly needed for descriptive statistics and data management. Thus, additional functions spread across different packages need to be found to install these packages which are loaded every R session. Depending on the author(s), functions in an R package can be more or less user-friendly in terms of the required input for the function and the output provided by the function.

The **main goal** for programming the `misty` package was to provide user-friendly functions for descriptive statistics, data management, missing data, and statistical analysis. More specifically, the `misty` package provides functions which (1) simplify descriptive statistics, (2) have sensible default options for arguments, (3) results in clearly arranged outputs, and (4) allow to analyze more than one dependent variable by using a function call. The long-term goal of the `misty` package is to offer a set of functions which covers the process of data management and descriptive statistics in most of the applications in the social sciences.

1.2.1 Descriptive statistics in the R base system

The R base system provides numerous functions for descriptive statistics. Some of these functions, however, only provide limited information so that additional programming is required to obtain all information needed. For example, following syntax is required to obtain a table with absolute frequencies and percentage frequencies with two digits:

```
> # Table with absolute and percentage frequencies
> cbind(Freq = table(mtcars$gear),
+       Perc = round(prop.table(table(mtcars$gear)) * 100 , digits = 2))
  Freq  Perc
3   15 46.88
4   12 37.50
5    5 15.62
```

In the `misty` package, a table with absolute frequencies and percentage frequencies with two digits can be obtained by using the `freq()` function:

```
> # Table with absolute and percentage frequencies
> freq(mtcars$gear)
                Freq    Perc
 Value   3        15  46.88%
         4        12  37.50%
         5         5  15.62%
```

```
          Total   32 100.00%
  Missing NA       0   0.00%
```

1.2.2 Default setting of function arguments

There are additional packages with useful functions for data management and descriptive statistics. Some of these functions, however, have an odd default setting for argument so that these arguments need to be specified whenever the function is used. For example, the `read.sav()` function in the `foreign` package can be used to read a SPSS file. This function has the default setting `to.data.frame = FALSE` which needs to be specified as `to.data.frame = TRUE` to obtain a data frame:

```
> # Location and name of the SPSS data set
> sav <- system.file("files", "electric.sav", package = "foreign")

> # Read SPSS data and print first six cases
> head(foreign::read.spss(sav, to.data.frame = TRUE))
  CASEID      FIRSTCHD AGE DBP58 EDUYR CHOL58 CGT58 HT58 WT58  DAYOFWK VITAL10
1     13    NONFATALMI  40    70    16    321     0 68.8  190     <NA>   ALIVE
2     30    NONFATALMI  49    87    11    246    60 72.2  204 THURSDAY   ALIVE
3     53 SUDDEN  DEATH  43    89    12    262     0 69.0  162 SATURDAY    DEAD
4     84    NONFATALMI  50   105     8    275    15 62.5  152 WEDNSDAY   ALIVE
5     89 SUDDEN  DEATH  43   110    NA    301    25 68.0  148   MONDAY    DEAD
6    102    NONFATALMI  50    88     8    261    30 68.0  142   SUNDAY    DEAD
```

In the `misty` package, a SPSS file can be read by using the `read.sav()` function. By default, this function returns a data frame without using value labels:

```
> # Read SPSS data and print first six cases
> head(read.sav(sav))
  CASEID FIRSTCHD AGE DBP58 EDUYR CHOL58 CGT58 HT58 WT58 DAYOFWK VITAL10
1     13        3  40    70    16    321     0 68.8  190      NA       0
2     30        3  49    87    11    246    60 72.2  204       5       0
3     53        2  43    89    12    262     0 69.0  162       7       1
4     84        3  50   105     8    275    15 62.5  152       4       0
5     89        2  43   110    NA    301    25 68.0  148       2       1
6    102        3  50    88     8    261    30 68.0  142       1       1
```

1.2.3 Output provided by functions

Some outputs provided by functions in additional packages are not very user-friendly, i.e., they are not clearly arranged and/or provide additional nonessential results. For example, the `alpha()` function in the `psych` package can be used to compute coefficient alpha and item-total correlations, but provides a lot of nonessential results:

```
> dat <- data.frame(item1 = c(5, 2, 3, 4, 1, 2, 4, 2),
+                   item2 = c(5, 1, 3, 5, 2, 2, 5, 1),
+                   item3 = c(4, 2, 4, 5, 1, 3, 5, 1),
+                   item4 = c(5, 1, 2, 5, 2, 3, 4, 2))

> # Compute coefficient alpha and item-total correlations
> psych::alpha(dat)

 Reliability analysis
 Call: psych::alpha(x = dat)
```

```
   raw_alpha std.alpha G6(smc) average_r S/N   ase mean  sd median_r
       0.96      0.96    0.96      0.85  23 0.025    3 1.5     0.86

  lower alpha upper     95% confidence boundaries
 0.91 0.96 1

  Reliability if an item is dropped:
       raw_alpha std.alpha G6(smc) average_r S/N alpha se  var.r med.r
 item1      0.94      0.94    0.94      0.85  16    0.035 0.0073  0.88
 item2      0.92      0.93    0.90      0.81  13    0.048 0.0028  0.84
 item3      0.95      0.96    0.94      0.88  22    0.027 0.0014  0.89
 item4      0.95      0.95    0.93      0.87  20    0.029 0.0007  0.88

  Item statistics
       n raw.r std.r r.cor r.drop mean  sd
 item1 8  0.94  0.95  0.92   0.90  2.9 1.4
 item2 8  0.98  0.98  0.98   0.96  3.0 1.8
 item3 8  0.92  0.92  0.89   0.86  3.1 1.6
 item4 8  0.92  0.93  0.90   0.87  3.0 1.5

 Non missing response frequency for each item
         1    2    3    4    5 miss
 item1 0.12 0.38 0.12 0.25 0.12    0
 item2 0.25 0.25 0.12 0.00 0.38    0
 item3 0.25 0.12 0.12 0.25 0.25    0
 item4 0.12 0.38 0.12 0.12 0.25    0
```

In the `misty` package, coefficient alpha and item-total correlations can be computed by using the `alpha.coef()` function which provides a concise output:

```
> # Compute coefficient alpha and item-total correlations
> alpha.coef(dat)
 Unstandardized Coefficient Alpha with 95% Confidence Interval

  Items Alpha  Low  Upp
      4  0.96 0.87 0.99

 Item-Total Correlation and Coefficient Alpha if Item Deleted

  Variable n nNA   pNA    M   SD  Min  Max It.Cor Alpha
  item1    8   0 0.00% 2.88 1.36 1.00 5.00   0.90  0.94
  item2    8   0 0.00% 3.00 1.77 1.00 5.00   0.96  0.92
  item3    8   0 0.00% 3.12 1.64 1.00 5.00   0.86  0.95
  item4    8   0 0.00% 3.00 1.51 1.00 5.00   0.87  0.95
```

1.2.4 Number of dependent variables

Functions in additional packages are sometimes limited to one dependent variable, so that multiple function calls are needed to analyze all dependent variables. For example, the `cohen.d()` function in the `effsize` package for computing Cohen's *d* is limited to one dependent variable. Note that a warning message is printed every function call because the function requires a factor as grouping variable:

```
> # Compute Cohen's d
> effsize::cohen.d(disp ~ vs, data = mtcars)
```

```
 Warning in cohen.d.formula(disp ~ vs, data = mtcars): Cohercing rhs of formula to factor

 Cohen's d

 d estimate: 1.970198 (large)
 95 percent confidence interval:
    lower    upper
 1.085549 2.854847
> effsize::cohen.d(hp ~ vs, data = mtcars)
 Warning in cohen.d.formula(hp ~ vs, data = mtcars): Cohercing rhs of formula to factor

 Cohen's d

 d estimate: 2.043209 (large)
 95 percent confidence interval:
    lower    upper
 1.147832 2.938587
```

In the `misty` package, Cohen's *d* can be computed by using the `cohens.d()` function which is not limited to one dependent variable:

```
> # Compute Cohen's d
> cohens.d(cbind(disp, hp) ~ vs, data = mtcars, digits = 1)
  Cohen's d for bewteen-subject design with 95% confidence interval

   Variable n1 nNA1    M1   SD1 n2 nNA2    M2  SD2 M.Diff   SD Estimate  SE  Low  Upp
   disp     18    0 307.1 106.8 14    0 132.5 56.9 -174.7 88.7     -2.0 0.5 -3.0 -1.2
   hp       18    0 189.7  60.3 14    0  91.4 24.4  -98.4 48.1     -2.0 0.5 -3.1 -1.3

  Note. SD = weighted pooled standard deviation
```

1.3 Bug reports, feedback, and feature requests

If you find bugs or any problems specific to the `misty` package, please send me a report including reproducible examples. Of course, feedback about how to improve the package and feature requests are also very welcome. You can contact me at: <takuya.yanagida@univie.ac.at>

1.4 Acknowledgement

Special thanks to Martin Müller and Žiga Puklavec for designing the hexagon sticker for the `misty` package. I would also like to thank Jerome Olsen for providing valuable ideas regarding the `cohens.d()` function.

2 Functions in misty

Functions provided in the `misty` package can be grouped in (1) functions for descriptive statistics, (2) functions for missing data, (3) functions for data management, and (4) functions for statistical analysis.

2.1 Functions for descriptive statistics

2.1.1 Frequency Tables

The `freq()` function computes frequency tables with absolute and percentage frequencies for one or more than one variable.

```
> # Data frame
> dat <- data.frame(x1 = c(3, 3, 2, 3, 2, 3, 3, 2, 1, -99),
+                   x2 = c(2, 2, 1, 3, 1, 1, 3, 3, 2, 2),
+                   y1 = c(1, 4, NA, 5, 2, 4, 3, 5, NA, 1),
+                   y2 = c(2, 3, 4, 3, NA, 4, 2, 3, 4, 5),
+                   z = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))

> # Frequency table for one variable
> freq(dat$x1)
              Freq    Perc
  Value   -99    1  10.00%
          1      1  10.00%
          2      3  30.00%
          3      5  50.00%
          Total 10 100.00%
  Missing NA     0   0.00%

> # Frequency table for one variable, convert value -99 into NA
> freq(dat$x1, as.na = -99)
              Freq    Perc  V.Perc
  Value   1      1  10.00%  11.11%
          2      3  30.00%  33.33%
          3      5  50.00%  55.56%
          Total  9  90.00% 100.00%
  Missing NA     1  10.00%
  Total         10 100.00%

> # Frequency table for one variable, values shown in columns
> freq(dat$x1, val.col = TRUE, as.na = -99)
  Value       1      2      3   Total Missing   Total
  Freq        1      3      5       9       1      10
  Perc   10.00% 30.00% 50.00%  90.00%  10.00% 100.00%
  V.Perc 11.11% 33.33% 55.56% 100.00%

> # Frequency table for more than one variable
> freq(dat[, c("x1", "x2", "y1", "y2")], as.na = -99)
 Frequencies
              x1 x2 y1 y2
  Value   1    1  3  2  0
          2    3  4  1  2
          3    5  3  1  3
```

```
         4      0  0  2  3
         5      0  0  2  1
         Total  9 10  8  9
 Missing NA     1  0  2  1
 Total         10 10 10 10

> # Frequency table for more than one variable, values shown in columns
> freq(dat[, c("x1", "x2", "y1", "y2")], val.col = TRUE, as.na = -99)
Frequencies
    1 2 3 4 5 Total Missing Total
 x1 1 3 5 0 0     9       1    10
 x2 3 4 3 0 0    10       0    10
 y1 2 1 1 2 2     8       2    10
 y2 0 2 3 3 1     9       1    10

> # Frequency table for more than one variable, with percentage frequencies
> freq(dat[, c("x1", "x2", "y1", "y2")], print = "all", as.na = -99)
Frequencies
               x1 x2 y1 y2
 Value   1      1  3  2  0
         2      3  4  1  2
         3      5  3  1  3
         4      0  0  2  3
         5      0  0  2  1
         Total  9 10  8  9
 Missing NA     1  0  2  1
 Total         10 10 10 10

Percentages
                    x1      x2      y1      y2
 Value   1      10.00%  30.00%  20.00%   0.00%
         2      30.00%  40.00%  10.00%  20.00%
         3      50.00%  30.00%  10.00%  30.00%
         4       0.00%   0.00%  20.00%  30.00%
         5       0.00%   0.00%  20.00%  10.00%
         Total  90.00% 100.00%  80.00%  90.00%
 Missing NA     10.00%   0.00%  20.00%  10.00%
 Total         100.00% 100.00% 100.00% 100.00%

Valid Percentages
              x1      x2      y1      y2
 Value 1  11.11%  30.00%  25.00%   0.00%
       2  33.33%  40.00%  12.50%  22.22%
       3  55.56%  30.00%  12.50%  33.33%
       4   0.00%   0.00%  25.00%  33.33%
       5   0.00%   0.00%  25.00%  11.11%
 Total   100.00% 100.00% 100.00% 100.00%

> # Frequency table for more than one variable, split output table
> freq(dat[, c("x1", "x2")], split = TRUE, as.na = -99)

$x1
               Freq    Perc  V.Perc
```

```
  Value   1        1  10.00%  11.11%
          2        3  30.00%  33.33%
          3        5  50.00%  55.56%
          Total    9  90.00% 100.00%
  Missing NA       1  10.00%
  Total           10 100.00%

 $x2
                Freq    Perc
  Value   1        3  30.00%
          2        4  40.00%
          3        3  30.00%
          Total   10 100.00%
  Missing NA       0   0.00%
```

2.1.2 Cross Tabulation

The `crosstab()` function creates a two-way and three-way cross tabulation with absolute frequencies and row-wise, column-wise and total percentages.

```
> dat <- data.frame(x1 = c(1, 2, 2, 1, 1, 2, 2, 1, 1, 2),
+                   x2 = c(1, 2, 2, 1, 2, 1, 1, 1, 2, 1),
+                   x3 = c(-99, 2, 1, 1, 1, 2, 2, 2, 2, 1))

> # Cross Tabulation for x1 and x2
> crosstab(dat[, c("x1", "x2")])
       x2
  x1      1 2 Total
       1  3 2     5
       2  3 2     5
   Total  6 4    10

> # Cross Tabulation for x1 and x2, print all percentages
> crosstab(dat[, c("x1", "x2")], print = "all")
             x2
  x1               1      2 Total
       1 Freq        3      2     5
         Row %  60.00% 40.00%
         Col %  50.00% 50.00%
         Tot %  30.00% 20.00%
       2 Freq        3      2     5
         Row %  60.00% 40.00%
         Col %  50.00% 50.00%
         Tot %  30.00% 20.00%
   Total             6      4    10

> # Cross Tabulation for x1, x2, and x3
> crosstab(dat[, c("x1", "x2", "x3")])
            x3
  x1      x2    -99 1 2 Total
       1   1      1 1 1     3
           2      0 1 1     2
       2   1      0 1 2     3
```

```
          2     0 1 1     2
   Total        1 4 5    10

> # Cross Tabulation for x1, x2, and x3, print all percentages
> crosstab(dat[, c("x1", "x2", "x3")], print = "all")
                  x3
  x1     x2            -99      1      2 Total
       1   1 Freq        1      1      1     3
             Row %   33.33% 33.33% 33.33%
             Col %  100.00% 50.00% 50.00%
             Tot %   10.00% 10.00% 10.00%
           2 Freq        0      1      1     2
             Row %    0.00% 50.00% 50.00%
             Col %    0.00% 50.00% 50.00%
             Tot %    0.00% 10.00% 10.00%
       2   1 Freq        0      1      2     3
             Row %    0.00% 33.33% 66.67%
             Col %      NA% 50.00% 66.67%
             Tot %    0.00% 10.00% 20.00%
           2 Freq        0      1      1     2
             Row %    0.00% 50.00% 50.00%
             Col %      NA% 50.00% 33.33%
             Tot %    0.00% 10.00% 10.00%
   Total                 1      4      5    10

> # Cross Tabulation for x1, x2, and x3, print all percentages, split output table
> crosstab(dat[, c("x1", "x2", "x3")], print = "all", split = TRUE)
  Frequencies
             x3
   x1     x2   -99 1 2 Total
        1   1    1 1 1     3
        1   2    0 1 1     2
        2   1    0 1 2     3
        2   2    0 1 1     2
    Total        1 4 5    10

  Row-Wise Percentages
             x3
   x1     x2      -99      1      2   Total
        1   1  33.33% 33.33% 33.33% 100.00%
        1   2   0.00% 50.00% 50.00% 100.00%
        2   1   0.00% 33.33% 66.67% 100.00%
        2   2   0.00% 50.00% 50.00% 100.00%

  Column-Wise Percentages
             x3
   x1     x2       -99       1       2
        1   1  100.00%  50.00%  50.00%
        1   2    0.00%  50.00%  50.00%
    Total      100.00% 100.00% 100.00%
        2   1      NA%  50.00%  66.67%
        2   2      NA%  50.00%  33.33%
    Total          NA% 100.00% 100.00%
  Total Percentages
              x3
   x1     x2      -99      1      2   Total
        1   1  10.00% 10.00% 10.00%
        1   2   0.00% 10.00% 10.00%
        2   1   0.00% 10.00% 20.00%
        2   2   0.00% 10.00% 10.00%
                                    100.00%
```

2.1.3 Descriptive Statistics

The `descript()` function computes summary statistics for one or more variables optionally by a grouping variable.

```
> dat <- data.frame(group1 = c(1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2),
+                   group2 = c(1, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2),
+                   x1 = c(3, 1, 4, 2, 5, 3, 2, 4, NA, 4, 5, 3),
+                   x2 = c(4, NA, 3, 6, 3, 7, 2, 7, 5, 1, 3, 6),
+                   x3 = c(7, 8, 5, 6, 4, NA, 8, NA, 6, 5, 8, 6))
>
> # Descriptive statistics for x1
> descript(dat$x1)
  n nNA   pNA    M   SD  Min  Max  Skew  Kurt
 11   1 8.33% 3.27 1.27 1.00 5.00 -0.26 -0.62
>
> # Descriptive statistics for x1, print all available statistical measures
> descript(dat$x1, print = "all")
  n nNA   pNA    M  Var   SD  Min  p25  Med  p75  Max Range  IQR  Skew  Kurt
 11   1 8.33% 3.27 1.62 1.27 1.00 2.50 3.00 4.00 5.00  4.00 1.50 -0.26 -0.62
>
> # Descriptive statistics for x1, x2, and x3, analysis by group1 separately
> descript(dat[, c("x1", "x2", "x3")], group = dat$group1)
 Group  Variable n nNA    pNA    M   SD  Min  Max  Skew  Kurt
  1     x1       6   0  0.00% 3.00 1.41 1.00 5.00  0.00 -0.30
  1     x2       5   1 16.67% 4.60 1.82 3.00 7.00  0.57 -2.23
  1     x3       5   1 16.67% 6.00 1.58 4.00 8.00  0.00 -1.20
  2     x1       5   1 16.67% 3.60 1.14 2.00 5.00 -0.40 -0.18
  2     x2       6   0  0.00% 4.00 2.37 1.00 7.00  0.00 -1.88
  2     x3       5   1 16.67% 6.60 1.34 5.00 8.00  0.17 -2.41
>
> # Descriptive statistics for x1, x2, and x3, split analysis by group1
> descript(dat[, c("x1", "x2", "x3")], split = dat$group2)
 Split Group: 1
   Variable  n nNA    pNA    M   SD  Min  Max  Skew  Kurt
    x1       5   1 16.67% 2.80 1.30 1.00 4.00 -0.54 -1.49
    x2       5   1 16.67% 4.20 1.92 2.00 7.00  0.59 -0.02
    x3       5   1 16.67% 6.80 1.30 5.00 8.00 -0.54 -1.49

 Split Group: 2
   Variable  n nNA    pNA    M   SD  Min  Max  Skew  Kurt
    x1       6   0  0.00% 3.67 1.21 2.00 5.00 -0.08 -1.55
    x2       6   0  0.00% 4.33 2.34 1.00 7.00 -0.32 -1.66
    x3       5   1 16.67% 5.80 1.48 4.00 8.00  0.55  0.87
```

```
>
> # Descriptive statistics for x1, x2, and x3, analysis by group1 separately,
> # split analysis by group2
> descript(dat[, c("x1", "x2", "x3")], group = dat$group1, split = dat$group2)
  Split Group: 1
    Group  Variable n nNA    pNA    M   SD  Min  Max  Skew Kurt
     1     x1       3   0  0.00% 2.67 1.53 1.00 4.00 -0.94   NA
     1     x2       2   1 33.33% 3.50 0.71 3.00 4.00    NA   NA
     1     x3       3   0  0.00% 6.67 1.53 5.00 8.00 -0.94   NA
     2     x1       2   1 33.33% 3.00 1.41 2.00 4.00    NA   NA
     2     x2       3   0  0.00% 4.67 2.52 2.00 7.00 -0.59   NA
     2     x3       2   1 33.33% 7.00 1.41 6.00 8.00    NA   NA

  Split Group: 2
    Group  Variable n nNA    pNA    M   SD  Min  Max  Skew Kurt
     1     x1       3   0  0.00% 3.33 1.53 2.00 5.00  0.94   NA
     1     x2       3   0  0.00% 5.33 2.08 3.00 7.00 -1.29   NA
     1     x3       2   1 33.33% 5.00 1.41 4.00 6.00    NA   NA
     2     x1       3   0  0.00% 4.00 1.00 3.00 5.00  0.00   NA
     2     x2       3   0  0.00% 3.33 2.52 1.00 6.00  0.59   NA
     2     x3       3   0  0.00% 6.33 1.53 5.00 8.00  0.94   NA
```

2.1.4 Multilevel Descriptve Statistics

The `multilevel.descript()` function computes descriptive statistics for multilevel data, e.g. average group size, intraclass correlation coefficient, design effect and effectice sample size.

```
> dat <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
+                   group = c(1, 1, 1, 1, 2, 2, 3, 3, 3),
+                   x1 = c(2, 3, 2, 2, 1, 2, 3, 4, 2),
+                   x2 = c(3, 2, 2, 1, 2, 1, 3, 2, 5),
+                   x3 = c(2, 1, 2, 2, 3, 3, 5, 2, 4))

> # Multilevel descriptive statistics for x1
> multilevel.descript(dat$x1, group = dat$group)
 Multilevel Descriptive Statistics

  No. of cases             9
  No. of missing values    0

  No. of groups            3
  Average group size    3.00
  SD group size         1.00
  Min group size           2
  Max group size           4

  ICC(1)               0.339
  ICC(2)               0.606

  Design effect         1.68
  Design effect sqrt    1.30
  Effective sample size 5.36
```

2.1.5 Intraclass Correlation Coefficient, ICC(1) and ICC(2)

The `multileve.icc()` function computes the intraclass correlation coefficient ICC(1), i.e., proportion of the total variance explained by the grouping structure, and ICC(2), i.e., reliability of aggregated variables.

```
> dat <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
+                   group = c(1, 1, 1, 1, 2, 2, 3, 3, 3),
+                   x1 = c(2, 3, 2, 2, 1, 2, 3, 4, 2),
+                   x2 = c(3, 2, 2, 1, 2, 1, 3, 2, 5),
+                   x3 = c(2, 1, 2, 2, 3, 3, 5, 2, 4))

> # ICC(1) for x1
> multilevel.icc(dat$x1, group = dat$group)
 [1] 0.3389831
```

2.1.6 Correlation Matrix with Statistical Significance Testing

The `cor.matrix()` function computes a correlation matrix and computes significance values (p-values) for testing the hypothesis H0: $\rho = 0$ for all possible pairs of variables.

```
> dat <- data.frame(group = c("a", "a", "a", "a", "a", "b", "b", "b", "b", "b"),
+                    x = c(5, NA, 6, 4, 6, 7, 9, 5, 8, 7),
+                    y = c(3, 3, 5, 6, 7, 4, 7, NA, NA, 8),
+                    z = c(1, 3, 1, NA, 2, 4, 6, 5, 9, 6))

> # Pearson product-moment correlation coefficient matrix using pairwise deletion
> cor.matrix(dat[, c("x", "y", "z")])
 Pearson Product-Moment Correlation Coefficient

       x    y z
  x
  y 0.38
  z 0.68 0.58

> # Pearson product-moment correlation coefficient matrix using pairwise deletion,
> # print sample size and significance values
> cor.matrix(dat[, c("x", "y", "z")], print = "all")
 Pearson Product-Moment Correlation Coefficient

       x    y z
  x
  y 0.38
  z 0.68 0.58

 Sample Size Using Pairwise Deletion

    x y z
  x
  y 7
  z 8 7

 Significance Value (p-value)

        x     y z
```

```
   x
   y 0.401
   z 0.066 0.168

   Adjustment for multiple testing: none
```

2.1.7 Polychoric Correlation Matrix

The `poly.cor()` function computes a polychoric correlation matrix, which is the estimated Pearson product-moment correlation matrix between underlying normally distributed latent variables which generate the ordinal scores.

```
> dat <- data.frame(x1 = c(1, 1, 3, 2, 1, 2, 3, 2, 3, 1),
+                   x2 = c(1, 2, 1, 1, 2, 2, 2, 1, 3, 1),
+                   x3 = c(1, 3, 2, 3, 3, 1, 3, 2, 1, 2))
>
> # Polychoric correlation matrix
> poly.cor(dat)
 Polychoric Correlation Matrix

       x1    x2   x3
  x1  1.00
  x2  0.36  1.00
  x3 -0.17 -0.10 1.00
```

2.1.8 Coefficient Alpha and Item Statistics

The `alpha.coef()` function computes point estimate and confidence interval for the coefficient alpha (aka Cronbach's alpha) along with the corrected item-total correlation and coefficient alpha if item deleted.

```
> dat <- data.frame(item1 = c(5, 2, 3, 4, 1, 2, 4, 2),
+                   item2 = c(5, 1, 3, 5, 2, 2, 5, 1),
+                   item3 = c(4, 2, 4, 5, 1, 3, 5, 1),
+                   item4 = c(5, 1, 2, 5, 2, 3, 4, 2))

> # Compute unstandardized coefficient alpha and item statistics
> alpha.coef(dat)
 Unstandardized Coefficient Alpha with 95% Confidence Interval

  Items Alpha  Low  Upp
      4  0.96 0.87 0.99

 Item-Total Correlation and Coefficient Alpha if Item Deleted

  Variable n nNA   pNA    M   SD  Min  Max It.Cor Alpha
  item1    8   0 0.00% 2.88 1.36 1.00 5.00   0.90  0.94
  item2    8   0 0.00% 3.00 1.77 1.00 5.00   0.96  0.92
  item3    8   0 0.00% 3.12 1.64 1.00 5.00   0.86  0.95
  item4    8   0 0.00% 3.00 1.51 1.00 5.00   0.87  0.95
```

2.1.9 Cohen's d for Between- and Within-Subject Design

The `cohens.d()` function computes Cohen's d for between- and within-subject designs with confidence intervals. By default, the function computes the standardized mean difference divided by the weighted pooled standard deviation without applying the correction factor for removing the small sample bias.

```
> #----------------------------------------
> # Between-subject design
> dat.bs <- data.frame(group = c("cont", "cont", "cont", "treat",  "treat"),
+                      y1 = c(1, 3, 2, 5, 7),
+                      y2 = c(4, 3, 3, 6, 4),
+                      y3 = c(7, 5, 7, 3, 2))

> # Standardized mean difference divided by the weighted pooled standard deviation
> # without small sample correction factor
> cohens.d(y1 ~ group, data = dat.bs)
  Cohen's d for bewteen-subject design with 95% confidence interval

   Variable n1 nNA1   M1   SD1 n2 nNA2   M2   SD2 M.Diff   SD Estimate   SE  Low   Upp
   y1        3    0 2.00 1.00  2    0 6.00 1.41   4.00 1.15     3.46 3.95 1.44 13.67

  Note. SD = weighted pooled standard deviation

> # Cohens's d for for more than one outcome variable
> cohens.d(cbind(y1, y2, y3) ~ group, data = dat.bs)
  Cohen's d for bewteen-subject design with 95% confidence interval

   Variable n1 nNA1   M1   SD1 n2 nNA2   M2   SD2 M.Diff   SD Estimate   SE    Low   Upp
   y1        3    0 2.00 1.00  2    0 6.00 1.41   4.00 1.15     3.46 3.95   1.44 13.67
   y2        3    0 3.33 0.58  2    0 5.00 1.41   1.67 0.94     1.77 2.43  -0.02  7.88
   y3        3    0 6.33 1.15  2    0 2.50 0.71  -3.83 1.03    -3.73 4.20 -14.62 -1.63

  Note. SD = weighted pooled standard deviation

> #----------------------------------------
> # Within-subject design
> dat.ws <- data.frame(pre = c(1, 3, 2, 5, 7),
+                      post = c(2, 2, 1, 6, 8))

> # Standardized mean difference divided by the pooled standard deviation
> # while controlling for the correlation, without small sample correction factor
> cohens.d(post ~ pre, data = dat.ws, paired = TRUE)
  Cohen's d for within-subject design with 95% confidence interval

   n nNA Variable1   M1   SD1 Variable2   M2   SD2 M.Diff   SD Estimate   SE   Low  Upp
   5   0      post 3.80 3.03       pre 3.60 2.41  -0.20 1.10    -0.06 0.17 -0.43 0.26

  Note. SD = controlling for the correlation between measures
```

2.1.10 Phi Coefficient

The `phi.coef()` function computes the (adjusted) Phi coefficient between two or more than two dichotomous variables.

```
> dat <- data.frame(x1 = c(0, 1, 0, 1, 0, 1, 0, 1, 1, 0),
+                   x2 = c(0, 1, 0, 0, 1, 1, 1, 1, 1, 1),
+                   x3 = c(0, 1, 0, 1, 1, 1, 1, 1, 0, 0))

> # Phi coefficient matrix between x1, x2, and x3
> phi.coef(dat)
 Phi Coefficient Matrix

       x1    x2 x3
  x1
  x2 0.218
  x3 0.408 0.356
```

2.1.11 Pearson's Contingency Coefficient

The `cont.coef()` function computes the (adjusted) Pearson's contingency coefficient between two or more than two variables.

```
> dat <- data.frame(x = c(1, 1, 2, 1, 3, 3, 2, 2, 1, 2),
+                   y = c(3, 2, 3, 1, 2, 4, 1, 2, 3, 4),
+                   z = c(2, 2, 2, 1, 2, 2, 1, 2, 1, 2))

> # Contingency coefficient matrix between x, y, and z
> cont.coef(dat[, c("x", "y", "z")])
 Contingency Coefficient Matrix

       x     y z
  x
  y 0.522
  z 0.378 0.637
```

2.1.12 Cramer's V

The `cramers.v()` function computes the (bias-corrected) Cramer's V between two or more than two variables.

```
> dat <- data.frame(x = c(1, 1, 2, 1, 3, 3, 2, 2, 1, 2),
+                   y = c(1, 2, 2, 1, 3, 4, 1, 2, 3, 1),
+                   z = c(1, 1, 2, 1, 2, 3, 1, 2, 3, 2))

> # Bias-corrected Cramer's V matrix between x, y, and z
> cramers.v(dat[, c("x", "y", "z")])
 Bias-Corrected Cramer's V Matrix

       x     y z
  x
  y 0.283
  z 0.395 0.401
```

2.1.13 Eta Squared

The `eta.sq()` function computes eta squared for one or more outcome variables in combination with one or more grouping variables.

```
> dat <- data.frame(x1 = c(1, 1, 1, 1, 2, 2, 2, 2, 2),
+                   x2 = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
+                   y1 = c(3, 2, 4, 5, 6, 4, 7, 5, 7),
+                   y2 = c(2, 4, 1, 5, 3, 3, 4, 6, 7))

> # Eta squared for y1 explained by x1
> eta.sq(dat$y1, group = dat$x1)
 Eta Squared Matrix

   Estimate   0.499
```

2.1.14 Skewness

The `skewness()` function computes the skewness.

```
> # Compute skewness
> skewness(rnorm(100))
 [1] 0.3869627
```

2.1.15 Excess Kurtosis

The `kurtosis()` function computes the excess kurtosis.

```
> # Compute excess kurtosis
> kurtosis(rnorm(100))
 [1] 0.1674761
```

2.2 Functions for missing data

2.2.1 Descriptive Statistics for Missing Data

The `na.descript()` function computes descriptive statistics for missing data, e.g. number (%) of incomplete cases, number (%) of missing values, and summary statistics for the number (%) of missing values across all variables.

```
> dat <- data.frame(x1 = c(1, NA, 2, 5, 3, NA, 5, 2),
+                   x2 = c(4, 2, 5, 1, 5, 3, 4, 5),
+                   x3 = c(NA, 3, 2, 4, 5, 6, NA, 2),
+                   x4 = c(5, 6, 3, NA, NA, 4, 6, NA))

> # Descriptive statistics for missing data
> na.descript(dat)
  Descriptive Statistics for Missing Data

   No. of cases                 8
   No. of complete cases        1 (12.50%)
   No. of incomplete cases      7 (87.50%)

   No. of values               32
   No. of observed values      25 (78.12%)
   No. of missing values        7 (21.88%)
   No. of variables            4
   No. of missing values across all variables
     Mean                    1.75 (21.88%)
     SD                      1.26 (15.73%)
     Minimum                 0.00  (0.00%)
     P25                     1.50 (18.75%)
     P75                     2.25 (28.12%)
     Maximum                 3.00 (37.50%)
```

2.2.2 Missing Data Pattern

The `na.pattern()` function computes a summary of missing data patterns, i.e., number (%) of cases with a specific missing data pattern.

```
> dat <- data.frame(x = c(1, NA, NA, 6, 3),
+                   y = c(7, NA, 8, 9, NA),
+                   z = c(2, NA, 3, NA, 5))

> # Compute a summary of missing data patterns
> dat.pattern <- na.pattern(dat)
 Missing Data Pattern

  Pattern n    Perc x y z nNA     pNA
        1 1  20.00% 1 1 1   0   0.00%
        2 1  20.00% 1 1 0   1  33.33%
        3 1  20.00% 1 0 1   1  33.33%
        4 1  20.00% 0 1 1   1  33.33%
        5 1  20.00% 0 0 0   3 100.00%
          5 100.00% 2 2 2
```

2.2.3 Variance-Covariance Coverage

The `na.coverage()` function computes the proportion of cases that contributes for the calculation of each variance and covariance.

```
> dat <- data.frame(x = c(1, NA, NA, 6, 3),
+                   y = c(7, NA, 8, 9, NA),
+                   z = c(2, NA, 3, NA, 5))

> # Create missing data indicator matrix R
> na.coverage(dat)
 Variance-Covariance Coverage

      x    y    z
  x 0.60
  y 0.40 0.60
  z 0.40 0.40 0.60
```

2.2.4 Missing Data Indicator Matrix

The `na.indicator()` function creates a missing data indicator matrix R that denotes whether values are observed or missing, i.e., $r = 1$ if a value is observed, and $r = 0$ if a value is missing.

```
> dat <- data.frame(x = c(1, NA, NA, 6, 3),
+                   y = c(7, NA, 8, 9, NA),
+                   z = c(2, NA, 3, NA, 5))

> # Create missing data indicator matrix R
> na.indicator(dat)
  x y z
1 1 1 1
2 0 0 0
3 0 1 1
4 1 1 0
5 1 0 1
```

2.2.5 Auxiliary Variables

The `na.auxiliary()` function computes (1) Pearson product-moment correlation matrix to identify variables related to the incomplete variable and (2) Cohen's d comparing cases with and without missing values to identify variables related to the probability of missigness.

```
> dat <- data.frame(x1 = c(1, NA, 2, 5, 3, NA, 5, 2),
+                   x2 = c(4, 2, 5, 1, 5, 3, 4, 5),
+                   x3 = c(NA, 3, 2, 4, 5, 6, NA, 2),
+                   x4 = c(5, 6, 3, NA, NA, 4, 6, NA))

> # Auxiliary variables
> na.auxiliary(dat)
 Auxiliary Variables

  Variables related to the incomplete variable

   Pearson product-moment correlation matrix
         x1    x2    x3    x4
   x1
   x2 -0.62
   x3  0.63 -0.28
   x4  0.58 -0.57  0.05

  Variables related to the probability of missigness

   Cohen's d
         x1    x2    x3    x4
   x1        1.04 -0.75 -0.22
   x2    NA          NA    NA
   x3  0.00 -0.31       -0.89
   x4 -0.37 -0.04  0.00

  Note. Indicator variables are in the rows (0 = miss, 1 = obs)
```

2.2.6 Proportion of Missing Data for Each Case

The `na.prop()` function computes the proportion of missing data for each case in a matrix or data frame.

```
> dat <- data.frame(x = c(1, NA, NA, 6, 3),
+                   y = c(7, NA, 8, 9, NA),
+                   z = c(2, NA, 3, NA, 5))

> # Compute proportion of missing data (NA) for each case in the data frame
> na.prop(dat)
 [1] 0.00 1.00 0.33 0.33 0.33
```

2.2.7 Replace User-Specified Values with Missing Values

The `as.na()` function replaces user-spefied values in the argument `na` in a vector, factor, matrix or data frame with `NA`.

```
> x.num <- c(1, 3, 2, 4, 5)

> # Replace 2 with NA
> as.na(x.num, as.na = 2)
 [1]  1  3 NA  4  5

> # Replace 2, 3, and 4 with NA
> as.na(x.num, as.na = c(2, 3, 4))
 [1]  1 NA NA NA  5
```

2.2.8 Replace Missing Values with User-Specified Values

The `na.as()` function replaces `NA` in a vector, factor, matrix or data frame with user-spefied values in the argument `value`.

```
> x.num <- c(1, 3, NA, 4, 5)

> # Replace NA with 2
> na.as(x.num, value = 2)
 [1] 1 3 2 4 5
```

2.3 Functions for data mangement

2.3.1 Merge Multiple Data Frames

The `df.merge()` function merges data frames by a common column (i.e., matching variable).

```
> adat <- data.frame(id = c(1, 2, 3),
+                    x1 = c(7, 3, 8))

> bdat <- data.frame(id = c(1, 2),
+                    x2 = c(5, 1))

> cdat <- data.frame(id = c(2, 3),
+                    y3 = c(7, 9))

> ddat <- data.frame(id = 4,
+                    y4 = 6)
```

```
> # Merge adat, bdat, cdat, and data by the variable id
> df.merge(adat, bdat, cdat, ddat, by = "id", output = FALSE)
   id x1 x2 y3 y4
 1  1  7  5 NA NA
 2  2  3  1  7 NA
 3  3  8 NA  9 NA
 4  4 NA NA NA  6
```

2.3.2 Combine Data Frames by Rows, Filling in Missing Columns

The `df.rbind()` function takes a sequence of data frames and combines them by rows, while filling in missing columns with `NA`s.

```
> adat <- data.frame(id = c(1, 2, 3),
+                    a = c(7, 3, 8),
+                    b = c(4, 2, 7))

> bdat <- data.frame(id = c(4, 5, 6),
+                    a = c(2, 4, 6),
+                    c = c(4, 2, 7))

> cdat <- data.frame(id = c(7, 8, 9),
+                    a = c(1, 4, 6),
+                    d = c(9, 5, 4))

> df.rbind(adat, bdat, cdat)
   id a  b  c  d
 1  1 7  4 NA NA
 2  2 3  2 NA NA
 3  3 8  7 NA NA
 4  4 2 NA  4 NA
 5  5 4 NA  2 NA
 6  6 6 NA  7 NA
 7  7 1 NA NA  9
 8  8 4 NA NA  5
 9  9 6 NA NA  4
```

2.3.3 Rename Columns in a Matrix or Variables in a Data Frame

The `df.rename()` function renames columns in a matrix or variables in a data frame by specifying a character string or character vector indicating the columns or variables to be renamed and a character string or character vector indicating the corresponding replacement values.

```
> dat <- data.frame(a = c(3, 1, 6),
+                   b = c(4, 2, 5),
+                   c = c(7, 3, 1))

> # Rename variable b in the data frame 'dat' to y
> df.rename(dat, from = "b", to = "y")
   a y c
 1 3 4 7
 2 1 2 3
 3 6 5 1
```

2.3.4 Data Frame Sorting

The `df.sort()` function arranges a data frame in increasing or decreasing order according to one or more variables.

```
> dat <- data.frame(x = c(5, 2, 5, 5, 7, 2),
+                   y = c(1, 6, 2, 3, 2, 3),
+                   z = c(2, 1, 6, 3, 7, 4))

> # Sort data frame 'dat' by "x" in increasing order
> df.sort(dat, x)
   x y z
 1 2 6 1
 2 2 3 4
 3 5 1 2
 4 5 2 6
 5 5 3 3
 6 7 2 7
```

2.3.5 Recode Variable

The `rec()` function recodes a numeric vector, character vector, or factor according to recode specifications.

```
> x.num <- c(1, 2, 4, 5, 6, 8, 12, 15, 19, 20)

> # Recode 5 = 50 and 19 = 190
> rec(x.num, "5 = 50; 19 = 190")
  [1]   1   2   4  50   6   8  12  15 190  20

> # Recode 1, 2, and 5 = 100 and 4, 6, and 7 = 200 and else = 300
> rec(x.num, "c(1, 2, 5) = 100; c(4, 6, 7) = 200; else = 300")
  [1] 100 100 200 100 200 300 300 300 300 300
```

2.3.6 Reverse Code Scale Item

The `reverse.item()` function reverse codes an inverted item, i.e., item that is negatively worded.

```
> dat <- data.frame(item1 = c(5, 2, 3, 4, 1, 2, 4, 2),
+                   item2 = c(1, 5, 3, 1, 4, 4, 1, 5),
+                   item3 = c(4, 2, 4, 5, 1, 3, 5, -99))

> # Reverse code item2
> reverse.item(dat$item1, min = 1, max = 5)
 [1] 1 4 3 2 5 4 2 4
```

2.3.7 Compute Scale Scores

The `scores()` function computes (prorated) scale scores by averaging the (available) items that measure a single construct by default.

```
> dat <- data.frame(item1 = c(3,  2,  4, 1,  5, 1,  3, NA),
+                   item2 = c(2,  2, NA, 2,  4, 2, NA,  1),
+                   item3 = c(1,  1,  2, 2,  4, 3, NA, NA),
+                    item4 = c(4,  2,  4, 4, NA, 2, NA, NA),
+                    item5 = c(3, NA, NA, 2,  4, 3, NA,  3))

> # Prorated mean scale scores
> scores(dat)
 [1] 2.600000 1.750000 3.333333 2.200000 4.250000 2.200000 3.000000 2.000000

> # Prorated standard deviation scale scores
> scores(dat, fun = "sd")
 [1] 1.140175 0.500000 1.154701 1.095445 0.500000 0.836660       NA 1.414214

> # Prorated mean scale scores, minimum proportion of available item responses = 0.8
> scores(dat, p.avail = 0.8)
 [1] 2.60 1.75   NA 2.20 4.25 2.20   NA   NA
```

2.3.8 Group Scores

The `group.scores()` function computes group means by default.

```
> dat.ml <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
+                       group = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
+                       x = c(4, 2, 5, 6, 3, 4, 1, 3, 4))

> # Compute group means and expand to match the input x
> group.scores(dat.ml$x, group = dat.ml$group)
 [1] 3.666667 3.666667 3.666667 4.333333 4.333333 4.333333 2.666667 2.666667
 [9] 2.666667

> # Compute standard deviation for each group and expand to match the input x
> group.scores(dat.ml$x, group = dat.ml$group, fun = "sd")
 [1] 1.527525 1.527525 1.527525 1.527525 1.527525 1.527525 1.527525 1.527525
 [9] 1.527525
```

2.3.9 $r*_{wg(j)}$ Within-Group Agreement Index for Multi-Item Scales

The `rwg.lindell()` function computes r*wg(j) within-group agreement index for multi-item scales as described in Lindell, Brandt and Whitney (1999).

```
> dat <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
+                    group = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
+                    x1 = c(2, 3, 2, 1, 1, 2, 4, 3, 5),
+                    x2 = c(3, 2, 2, 1, 2, 1, 3, 2, 5),
+                    x3 = c(3, 1, 1, 2, 3, 3, 5, 5, 4))

> # Compute Fisher z-transformed r*wg(j) for a multi-item scale with A = 5 response options
> rwg.lindell(dat[, c("x1", "x2", "x3")], group = dat$group, A = 5)
        1         1         1         2         2         2         3         3 
0.8047190 0.8047190 0.8047190 1.1989476 1.1989476 1.1989476 0.4104903 0.4104903 
        3 
0.4104903
```

2.3.10 Centering at the Grand Mean or Centering within Cluster

The `center()` function is used to center predictors at the grand mean (CGM, i.e., grand mean centering) or within cluster (CWC, i.e., group-mean centering).

```
> #---------------------------------------
> # Predictors in a single-level regression
> dat.sl <- data.frame(x = c(4, 2, 5, 6, 3, 4, 1, 3, 4),
+                      y = c(5, 3, 6, 3, 4, 5, 2, 6, 5))

> # Center predictor at the sample mean
> center(dat.sl$x)
 [1]  0.4444444 -1.5555556  1.4444444  2.4444444 -0.5555556  0.4444444 -2.5555556
 [8] -0.5555556  0.4444444

> #---------------------------------------
> # Predictors in a multilevel regression
> dat.ml <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
+                      group = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
+                      x.l1 = c(4, 2, 5, 6, 3, 4, 1, 3, 4),
+                      x.l2 = c(4, 4, 4, 1, 1, 1, 3, 3, 3),
+                      y = c(5, 3, 6, 3, 4, 5, 2, 6, 5))

> # Center level-1 predictor at the grand mean (CGM)
> center(dat.ml$x.l1)
 [1]  0.4444444 -1.5555556  1.4444444  2.4444444 -0.5555556  0.4444444 -2.5555556
 [8] -0.5555556  0.4444444

> # Center level-1 predictor within cluster (CWC)
> center(dat.ml$x.l1, type = "CWC", group = dat.ml$group)
 [1]  0.3333333 -1.6666667  1.3333333  1.6666667 -1.3333333 -0.3333333 -1.6666667
 [8]  0.3333333  1.3333333

> # Center level-2 predictor at the grand mean (CGM)
> center(dat.ml$x.l2, type = "CGM", group = dat.ml$group)
 [1]  1.3333333  1.3333333  1.3333333 -1.6666667 -1.6666667 -1.6666667  0.3333333
 [8]  0.3333333  0.3333333
```

2.3.11 Dummy Coding

The `dummy.c()` function creates $k - 1$ dummy coded 0/1 variables for a vector with k distinct values.

```
> dat <- data.frame(x = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
+                   y = c("a", "a", "a", "b", "b", "b", "c", "c", "c"),
+                   z = factor(c("B", "B", "B", "A", "A", "A", "C", "C", "C")),
+                   stringsAsFactors = FALSE)

> # Dummy coding of a numeric variable, reference = 3
> dummy.c(dat$x)
      d1 d2
 [1,]  1  0
 [2,]  1  0
 [3,]  1  0
 [4,]  0  1
 [5,]  0  1
 [6,]  0  1
 [7,]  0  0
 [8,]  0  0
 [9,]  0  0

> # Dummy coding of a numeric variable, reference = 1
> dummy.c(dat$x, ref = 1)
      d2 d3
 [1,]  0  0
 [2,]  0  0
 [3,]  0  0
 [4,]  1  0
 [5,]  1  0
 [6,]  1  0
 [7,]  0  1
 [8,]  0  1
 [9,]  0  1
```

2.3.12 Omit Strings

The `stromit()` function omits user-specified values or strings from a numeric vector, character vector or factor.

```
> x.chr <- c("a", "", "c", NA, "", "d", "e", NA)

> # Omit character string ""
> stromit(x.chr)
[1] "a" "c" NA  "d" "e" NA

> # Omit character string "" and missing values (NA)
> stromit(x.chr, na.omit = TRUE)
[1] "a" "c" "d" "e"

> # Omit character string "c" and "e"
> stromit(x.chr, omit = c("c", "e"))
[1] "a" ""  NA  ""  "d" NA
```

2.3.13 Read SPSS File

The `read.sav()` function calls the `read_sav()` function in the *haven* package by Hadley Wickham and Evan Miller (2019) to read an SPSS file.

```
> # Read SPSS data
> # read.sav("SPSS_Data.sav")
```

2.3.14 Write SPSS File

The `write.sav()` function writes a data frame or matrix into a SPSS file by either by using the `write_sav()` function in the *haven* package by Hadley Wickham and Evan Miller (2019) or the free software `PSPP` (see: https://www.gnu.org/software/pspp/pspp.html).

```
> # dat <- data.frame(id = 1:5,
> #                   gender = c(NA, 0, 1, 1, 0),
> #                   age = c(16, 19, 17, NA, 16),
> #                   status = c(1, 2, 3, 1, 4),
> #                   score = c(511, 506, 497, 502, 491))
> #
> # Write SPSS file using the haven package
> # write.sav(dat, file = "Dataframe_haven.sav")
> #
> # Write SPSS file using PSPP,
> # write CSV file and SPSS syntax along with the SPSS file
> # write.sav(dat, file = "Dataframe_PSPP.sav", pspp.path = "C:/Program Files/PSPP",
> #           write.csv = TRUE, write.sps = TRUE)
> #
> # Specify variable attributes
> # Note that it is recommended to manually specify the variables attritbues in a CSV or
> # Excel file which is subsequently read into R
> # attr <- data.frame(# Variable names
> #                    var = c("id", "gender", "age", "status", "score"),
> #                    # Variable labels
> #                    label = c("Identification number", "Gender", "Age in years",
> #                              "Migration background", "Achievement test score"),
> #                    # Value labels
> #                    values = c("", "0 = female; 1 = male", "",
> #                               "1 = Austria; 2 = former Yugoslavia; 3 = Turkey; 4 = other", ""),
> #                    # User-missing values
> #                    missing = c("", "-99", "-99", "-99", "-99"))
> #
> # Write SPSS file with variable attributes using the haven package
> # write.sav(dat, file = "Dataframe_haven_Attr.sav", var.attr = attr)
> #
> # Write SPSS with variable attributes using PSPP
> # write.sav(dat, file = "Dataframe_PSPP_Attr.sav", var.attr = attr,
> #           pspp.path = "C:/Program Files/PSPP")
```

2.3.15 Read Mplus Data File and Variable Names

The `read.mplus()` function reads a Mplus data file and/or Mplus input/output file to return a data frame with variable names extracted from the Mplus input/output file.

```
> # Read Mplus data file and variable names extracted from the Mplus input file
> # dat <- read.mplus("Mplus_Data.dat", input = "Mplus_Input.inp")
```

2.3.16 Write Mplus Data File

The `write.mplus()` function writes a matrix or data frame to a tab-delimited file without variable names and a text file with variable names. Only numeric values are allowed, missing data will be coded as a single numeric value.

```
> # dat <- data.frame(id = 1:5,
> #                   x = c(NA, 2, 1, 5, 6),
> #                   y = c(5, 3, 6, 8, 2),
> #                   z = c(2, 1, 1, NA, 4))
> #
> # Write Mplus Data File and a text file with variable names
> # write.mplus(dat)
```

2.4 Functions for statistical analysis

2.4.1 Run Mplus Models

The `run.mplus()` function runs a group of Mplus models (`.inp` files) located within a single directory or nested within subdirectories.

```
> # Run Mplus models located within a single directory
> # run.mplus(Mplus = "C:/Program Files/Mplus/Mplus.exe")
```

2.4.2 Sample Size Determination for Testing Arithmetic Means

The `size.mean()` function performs sample size computation for the one-sample and two-sample t-test based on precision requirements (i.e., type-I-risk, type-II-risk and an effect size).

```
> # Two-sided one-sample test
> size.mean(theta = 0.5, sample = "one.sample",
+           alternative = "two.sided", alpha = 0.05, beta = 0.2)

 Sample size determination for the one-sample t-test

   H0: mu = mu.0  versus  H1: mu != mu.0
   alpha: 0.05  beta: 0.2  theta: 0.5

  optimal sample size: n = 34

> # One-sided two-sample test
> size.mean(theta = 1, sample = "two.sample",
+           alternative = "greater", alpha = 0.01, beta = 0.1)

 Sample size determination for the two-sample t-test

   H0: mu.1 <= mu.2  versus  H1: mu.1 > mu.2
   alpha: 0.01  beta: 0.1  theta: 1

  optimal sample size: n = 28 (in each group)
```

2.4.3 Sample Size Determination for Testing Proportions

The `size.prop()` function performs sample size computation for the one-sample and two-sample test for proportion based on precision requirements (i.e., type-I-risk, type-II-risk and an effect size).

```
> # Two-sided one-sample test
> size.prop(pi = 0.5, delta = 0.2, sample = "one.sample",
+           alternative = "two.sided", alpha = 0.05, beta = 0.2)

 Sample size determination for the one-sample proportion test without continuity correction
```

```
  H0: pi = 0.5  versus  H1: pi != 0.5
  alpha: 0.05  beta: 0.2  delta: 0.2

   optimal sample size: n = 47
> # One-sided two-sample test
> size.prop(pi = 0.5, delta = 0.2, sample = "two.sample",
+           alternative = "greater", alpha = 0.01, beta = 0.1)

 Sample size determination for the two-sample proportion test without continuity correction

  H0: pi.1 <= pi.2  versus  H1: pi.1 > pi.2
  alpha: 0.01  beta: 0.1  delta: 0.2

   optimal sample size: n = 154 (in each group)
```

2.4.4 Sample Size Determination for Testing Pearson's Correlation Coefficient

The `size.cor()` function performs sample size computation for testing Pearson's product-moment correlation coefficient based on precision requirements (i.e., type-I-risk, type-II-risk and an effect size).

```
> # Two-sided test
> size.cor(rho = 0.3, delta = 0.2, alpha = 0.05, beta = 0.2)

 Sample size determination for Pearson's product-moment correlation coefficient

  H0: rho = 0.3  versus  H1: rho != 0.3
  alpha: 0.05  beta: 0.2  delta: 0.2

    optimal sample size: n = 140
```