

bwrappers:
**A Package of Wrappers and Other R Functions for
Efficiently Cleaning, Analyzing, and Visualizing
Data in the Behavioral Sciences**

January 2020
github.com/michaelkardas/bwrappers
Contact: Michael Kardas (mkardas@chicagobooth.edu)

PREFACE

Many behavioral researchers perform several common analyses—such as ANOVA, regression, and chi-square analyses—repeatedly across multiple data sets. But rewriting code from scratch and transcribing outputs by hand takes time and leaves these analyses potentially susceptible to error. With this in mind, I developed *bwrappers*, a package containing wrappers and other R functions for more efficiently cleaning, analyzing, and visualizing data in the behavioral sciences. By "efficiently" I mean that these functions are designed to enhance both the speed and accuracy of your analyses. They do so by:

1. **Tailoring syntax** to the structure of a typical data sheet in the behavioral sciences.
2. **Automating intermediate steps** such as splitting and reshaping data frames.
3. **Returning statistics in APA format** and copying these outputs to the clipboard.

In this overview, I first provide instructions for installing the package and outline the sample data; then I describe the purpose and statistical assumptions of each function, provide syntax, and display reproducible examples. Many of the functions correspond to analyses that can also be performed within SPSS, and for each of these functions, I have cross-checked the outputs of the *bwrappers* functions against the outputs of SPSS. You can download the sample data sheet, sample R script, and SPSS syntax (for error checks) through GitHub [here](#).

Importantly, the functions in this package are not comprehensive and do *not* substitute for learning basic data cleaning, analysis, and visualization skills using other R packages as well. Some analyses will require departing from the statistical assumptions that are built into these functions, and many data sets will require advanced analytic techniques that fall outside the scope of these functions altogether. This package is instead intended to streamline several common operations that you may find yourself performing repeatedly as a behavioral scientist.

CONTENTS

PREFACE	2
INSTALLATION & TROUBLESHOOTING	4
Installation	4
Troubleshooting.....	4
SAMPLE DATA.....	5
FUNCTIONS FOR DATA CLEANING.....	6
Splitting Data Frames	6
Merging Columns.....	6
Generating Columns	7
FUNCTIONS FOR DATA ANALYSIS.....	8
Descriptive Statistics	8
T Tests (One Sample)	8
T Tests (Independent Samples).....	8
T Tests (Paired Samples)	9
ANOVA (Main Effects & Interaction Effects).....	9
ANOVA (Planned Contrasts).....	10
ANOVA (Simple Main Effects)	11
Linear Regression	11
Correlation Tests.....	12
Chi-Square Tests	12
FUNCTIONS FOR DATA VISUALIZATION.....	13
Bar Plots	13
Line Plots.....	14
Histogram Plots.....	15

INSTALLATION & TROUBLESHOOTING

Installation

The *bwrappers* package is hosted on GitHub. To install the package through the R console, you will first need to load or install the *devtools* package and then run the `install_github()` command as follows:

```
if(!require(devtools)){install.packages("devtools"); library(devtools)}  
install_github("michaelkardas/bwrappers")
```

Troubleshooting

Here are some possible error messages and troubleshooting tips:

- Error message: *package 'bwrappers' is not available (for R version X.X.X)*
 - The *bwrappers* package requires R version 3.2.0 or greater. If you use an older version of R and prefer not to update, you can alternatively download the function code directly through GitHub [here](#).
- Error message: *ERROR: compilation failed for package 'package_name'*
 - Try uninstalling and then reinstalling the package listed in the error message:

```
remove.packages("package_name")  
install.packages("package_name")
```
- Error message: *Error: object 'x' is not exported by 'namespace:package_name'*
 - Try updating the package listed in the error message:

```
update.packages("package_name")
```
 - Otherwise, try uninstalling and then reinstalling the package:

```
remove.packages("package_name")  
install.packages("package_name")
```

If you encounter a different error message—and better still, if you encounter a different error message and then discover a solution—please let me know at mkardas@chicagobooth.edu.

SAMPLE DATA

To allow you to quickly test out the functions, I created sample data using a random number generator. The first 10 rows of the sample data are displayed below. Each row represents data from one participant, similar to data that you would obtain through Qualtrics or similar survey software. To enable a wide range of analyses, the sample data includes three independent variables plus several continuous, categorical, and discrete dependent variables.

Independent Variables			Categorical DV		Discrete DVs (with repeated measures)				Discrete DVs (without repeated measures)						
			Continuous DV												
IV1	IV2	IV3	DV1	DV2	DV3_T1	DV4_T1	DV3_T2	DV4_T2	DV5	DV6	DV7	DV8	DV8	DV9	DV9
Friend	PhotoA	Positive	7.7	Yes	1	3	2	4	1	2	8	8	NA	1	NA
Friend	PhotoA	Positive	6.0	Yes	10	3	7	10	2	2	9	5	NA	5	NA
Stranger	PhotoB	Positive	8.4	Yes	9	10	5	7	6	0	5	NA	2	NA	10
Friend	PhotoC	Negative	6.5	No	4	6	4	4	10	10	4	2	NA	6	NA
Stranger	PhotoC	Negative	7.7	Yes	5	2	8	2	7	8	10	NA	1	NA	2
Friend	PhotoC	Positive	0.2	No	1	8	10	3	4	8	4	6	NA	8	NA
Stranger	PhotoC	Negative	0.9	Yes	5	5	10	5	3	0	3	NA	5	NA	10
Friend	PhotoB	Positive	3.9	Yes	2	2	4	4	0	0	7	8	NA	4	NA
Stranger	PhotoC	Positive	6.9	Yes	0	0	7	5	6	10	2	NA	9	NA	0
Stranger	PhotoA	Negative	1.8	No	9	6	5	6	9	7	6	NA	4	NA	4

Throughout this overview, I provide reproducible examples to illustrate the uses of the *bwrappers* functions. Both the sample data sheet ("bdata.xlsx") and the R script containing those examples ("Rscript.R") are available [here](#) on GitHub.

FUNCTIONS FOR DATA CLEANING

Splitting Data Frames

In many studies, you will need to conduct the same analyses separately within each of several conditions. The function **wrap.split** splits a data frame across all combinations of 1-3 independent variables and then assigns the new data frames to the Global Environment. The function delegates the primary computations to **split** {package: *base*}.

Usage & Arguments

`wrap.split(df, iv1, iv2 = NULL, iv3 = NULL)`

- `df`: The data frame that you intend to split.
- `iv1, iv2, iv3`: Names of the columns containing the independent variables.

Example (splitting a data frame by 1 independent variable)

`wrap.split(df = bdata, iv1 = IV1)`



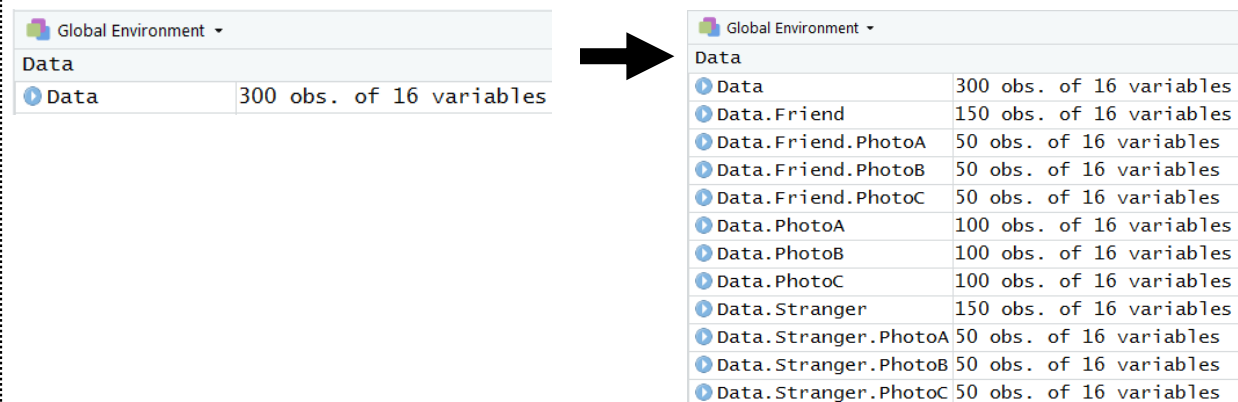
Global Environment	
Data	300 obs. of 16 variables

→

Global Environment	
Data	300 obs. of 16 variables
Data.Friend	150 obs. of 16 variables
Data.Stranger	150 obs. of 16 variables

Example (splitting a data frame by 2 independent variables)

`wrap.split(df = bdata, iv1 = IV1, iv2 = IV2)`



Global Environment	
Data	300 obs. of 16 variables

→

Global Environment	
Data	300 obs. of 16 variables
Data.Friend	150 obs. of 16 variables
Data.Friend.PhotoA	50 obs. of 16 variables
Data.Friend.PhotoB	50 obs. of 16 variables
Data.Friend.PhotoC	50 obs. of 16 variables
Data.PhotoA	100 obs. of 16 variables
Data.PhotoB	100 obs. of 16 variables
Data.PhotoC	100 obs. of 16 variables
Data.Stranger	150 obs. of 16 variables
Data.Stranger.PhotoA	50 obs. of 16 variables
Data.Stranger.PhotoB	50 obs. of 16 variables
Data.Stranger.PhotoC	50 obs. of 16 variables

Merging Columns

In some studies, participants in different conditions will complete the same dependent variables in separate survey blocks—one block per condition—and this may lead to a data sheet where responses on each dependent variable are disaggregated across several columns. The function **wrap.merge** searches the data frame for columns with identical names and then merges them into one column, preserving only the single entry within each row that is not blank or NA. The function outputs the revised data frame directly to the Global Environment.

Usage & Arguments


`wrap.merge(df)`

- `df`: The data frame that the function should search for identical column names.

Example

```
wrap.merge(df = bdata)
```

DV8	DV8	DV9	DV9
8	NA	1	NA
5	NA	5	NA
NA	2	NA	10
2	NA	6	NA
NA	1	NA	2
6	NA	8	NA
NA	5	NA	10
8	NA	4	NA
NA	9	NA	0
NA	4	NA	4



DV8	DV9
8	1
5	5
2	10
2	6
1	2
6	8
5	10
8	4
9	0
4	4

Generating Columns

You may need to compute sums, differences, or means between pairs of columns before analyzing your data: For example, in studies with repeated measures, you may need to compute differences between participants' responses at Time 1 versus Time 2. In studies involving pairs of participants, you may need to compute mean responses for each dependent measure across paired participants. The function **wrap.generate** searches the data frame for pairs of numeric columns whose names are identical except for specific strings that you've embedded within them—such as "T1" versus "T2" or "Participant1" versus "Participant2"—and then generates new columns by computing sums, differences, or means between pairs of corresponding columns.

Usage & Arguments

```
wrap.generate(df, string1, string2, operation, newString = operation)
```

- df: The data frame.
- string1, string2: Character strings embedded in the names of corresponding columns.
- operation: Character string to specify the operation ("sum", "difference", or "mean").
- newString: Optional character string to replace string1 and string2 in the names of the outputted column/s. By default, newString equals the operation ("sum", "difference", or "mean").

Example (computing differences between T1 responses and T2 responses)

```
wrap.generate(df = bdata, string1 = "T1", string2 = "T2", operation = "difference")
```

DV3_T1	DV4_T1	DV3_T2	DV4_T2	New columns	
DV3_T1	DV4_T1	DV3_T2	DV4_T2	DV3_difference	DV4_difference
1	3	2	4	-1	-1
10	3	7	10	3	-7
9	10	5	7	4	3
4	6	4	4	0	2
5	2	8	2	-3	0
1	8	10	3	-9	5
5	5	10	5	-5	0
2	2	4	4	-2	-2
0	0	7	5	-7	-5
9	6	5	6	4	0

FUNCTIONS FOR DATA ANALYSIS

Descriptive Statistics

The function **wrap.desc** computes descriptive statistics for one dependent variable, parsed by between 0 and 2 independent variables. Note that the function uses a t distribution to compute confidence intervals.

Usage & Arguments

```
wrap.desc(dv1, iv1 = NULL, iv2 = NULL)
```

- **dv1**: Column vector containing the dependent variable.
- **iv1, iv2**: Column vectors containing the independent variables (if applicable).

Example (parsing the dependent variable by 0 independent variables)

```
wrap.desc(dv1 = bdata$DV5)
```

```
# N = 300: M = 5.14, SD = 3.10, Var = 9.59, SE = 0.18, 95% CI = [4.79, 5.49]
```

Example (parsing the dependent variable by 1 independent variable)

```
wrap.desc(dv1 = bdata$DV5, iv1 = bdata$IV2)
```

```
# PhotoA (N = 100): M = 4.93, SD = 3.19, Var = 10.21, SE = 0.32, 95% CI = [4.30, 5.56]
```

```
# PhotoB (N = 100): M = 5.03, SD = 3.01, Var = 9.08, SE = 0.30, 95% CI = [4.43, 5.63]
```

```
# PhotoC (N = 100): M = 5.46, SD = 3.09, Var = 9.52, SE = 0.31, 95% CI = [4.85, 6.07]
```

T Tests (One Sample)

The function **wrap.t.one** performs one-sample t tests. The function delegates the primary operations to **t.test** {package: stats}.

Usage & Arguments

```
wrap.t.one(dv1, mu = 0)
```

- **dv1**: Column vector containing the dependent variable.
- **mu** (*default = 0*): Mean against which to compare responses on the dependent variable.

Example

```
wrap.t.one(dv1 = bdata$DV5, mu = 5)
```

```
# one-sample t(299) = 0.78, p = .434, 95% CI = [4.79, 5.49], d = 0.05
```

T Tests (Independent Samples)

The function **wrap.t.ind** performs independent-samples t tests. The function delegates the primary computations to **t.test** {package: stats}.

Usage & Arguments

```
wrap.t.ind(dv1, iv1, var.equal = TRUE)
```

- dv1: Column vector containing the dependent variable.
- iv1: Column vector containing the independent variable.
- var.equal (*default = TRUE*): A logical argument: If TRUE, the function assumes equal variance across conditions; if FALSE, the function does not assume equal variance.

Example

```
wrap.t.ind(dv1 = bdata$DV5, iv1 = bdata$IV1)
```

```
# t(298) = 0.15, p = .882, 95% CI difference = [-0.65, 0.76], d = 0.02
```

T Tests (Paired Samples)

The function **wrap.t.pair** performs paired-samples t tests. The function delegates the primary computations to **t.test** {package: *stats*}.

Usage & Arguments

```
wrap.t.pair(dv1, dv2)
```

- dv1, dv2: Column vectors containing the dependent variables.

Example

```
wrap.t.pair(dv1 = bdata$DV3_T1, dv2 = bdata$DV3_T2)
```

```
# paired t(299) = 1.12, p = .264, 95% CI difference = [-0.22, 0.81], d = 0.09
```

ANOVA (Main Effects & Interaction Effects)

The function **wrap.anova** computes main effects and interaction effects for ANOVA with up to 1 within-subjects factor and up to 3 between-subjects factors. The function delegates the primary computations to **ezANOVA** {package: *ez*}. Note that this function assumes categorical (i.e., unordered) independent variables and fixed effects, and does not apply sphericity corrections. In the output displayed below, hp2 denotes partial eta squared.

Usage & Arguments

```
wrap.anova(dv1, iv1 = NULL, iv2 = NULL, iv3 = NULL, type = 3)
```

- dv1: Column vector containing the between-subjects dependent variable OR multiple column vectors containing the within-subjects dependent variables.
- iv1, iv2, iv3: Column vectors containing between-subjects independent variables (if applicable).
- type (*default = 3*): Numeric argument representing sum-of-squares type (1, 2, or 3).

Example (1 within-subjects factor)

```
wrap.anova(dv1 = bdata[c(6, 8)])
```

```
# F(1, 299) = 1.25, p = .264, hp2 = .004
```

Example (2 between-subjects factors)

```
wrap.anova(dv1 = bdata$DV5, iv1 = bdata$IV1, iv2 = bdata$IV2)
```

```
# IV1 (main effect): F(1, 294) = 0.02, p = .882, hp2 = .0001
```

```
# IV2 (main effect): F(2, 294) = 0.82, p = .442, hp2 = .01
```

```
# IV1 x IV2 (2-way interaction): F(2, 294) = 0.37, p = .691, hp2 = .003
```

Example (1 within-subjects factor & 2 between-subjects factors)

```
wrap.anova(dv1 = bdata[c(6, 8)], iv1 = bdata$IV1, iv2 = bdata$IV2)
```

```
# IV1 (main effect): F(1, 294) = 2.08, p = .151, hp2 = .01
```

```
# IV2 (main effect): F(2, 294) = 0.33, p = .716, hp2 = .002
```

```
# Within-Subjects (main effect): F(1, 294) = 1.25, p = .264, hp2 = .004
```

```
# IV1 x IV2 (2-way interaction): F(2, 294) = 0.22, p = .801, hp2 = .002
```

```
# IV1 x Within-Subjects (2-way interaction): F(1, 294) = 0.001, p = .980, hp2 = .000002
```

```
# IV2 x Within-Subjects (2-way interaction): F(2, 294) = 0.30, p = .741, hp2 = .002
```

```
# IV1 x IV2 x Within-Subjects (3-way interaction): F(2, 294) = 2.40, p = .092, hp2 = .02
```

ANOVA (Planned Contrasts)

The function **wrap.planned** performs planned contrasts for a one-way, between-subjects ANOVA. This function assumes categorical (i.e., unordered) independent variables, fixed effects, and equal variance across conditions. Note that the confidence interval and Cohen's d use mean-square error to estimate variance.

Usage & Arguments

```
wrap.planned(dv1, iv1, levels, weights)
```

- **dv1**: Column vector containing the dependent variable.
- **iv1**: Column vector containing the between-subjects independent variable.
- **levels**: String vector containing two or more factor levels of the independent variable.
- **weights**: Numeric vector containing the contrast weights. Must sum to +1 and -1.

Example (contrast between two levels of the independent variable)

```
wrap.planned(dv1 = bdata$DV5, iv1 = bdata$IV2, levels = c("PhotoA", "PhotoB"), weights = c(-1, 1))
```

```
# t(297) = 0.23, p = .820, 95% CIdifference = [-0.76, 0.96], d = 0.03
```

Example (contrast between three levels of the independent variable)

```
wrap.planned(dv1 = bdata$DV5, iv1 = bdata$IV2, levels = c("PhotoA", "PhotoB", "PhotoC"), weights = c(-1, 0.5, 0.5))
```

```
# t(297) = 0.83, p = .407, 95% CIdifference = [-0.43, 1.06], d = 0.10
```

ANOVA (Simple Main Effects)

The function **wrap.simple** computes simple main effects for a two-way, between-subjects ANOVA. The function delegates the primary computations to **testInteractions** {package: *phia*}. This function assumes categorical (i.e., unordered) independent variables and fixed effects. In the output displayed below, hp2 denotes partial eta squared.

Usage & Arguments

```
wrap.simple(dv1, iv1, iv2, adjustment = "none")
```

- dv1: Column vector containing the dependent variable.
- iv1, iv2: Column vector containing the between-subjects independent variables. The function will test for simple main effects of iv1 separately at each level of iv2.
- adjustment (*default* = "none"): Character string representing the method of p value adjustment ("none", "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", or "fdr").

Example

```
wrap.simple(dv1 = bdata$DV5, iv1 = bdata$IV1, iv2 = bdata$IV2)
```

```
# PhotoA: F(1, 294) = 0.13, p = .724, hp2 = .0004
```

```
# PhotoB: F(1, 294) = 0.37, p = .542, hp2 = .001
```

```
# PhotoC: F(1, 294) = 0.26, p = .607, hp2 = .001
```

Linear Regression

The function **wrap.lm** performs linear regression analyses. The function delegates the primary computations to **lm** {package: *stats*} and **std_beta** {package: *sjstats*}. In the output, R² is not adjusted.

Usage & Arguments

```
wrap.lm(formula, standardized = FALSE)
```

- formula: The linear model.
- standardized (*default* = FALSE): A logical argument: if FALSE, the function returns unstandardized coefficients; if TRUE, the function returns standardized coefficients.

Example (unstandardized coefficients)

```
wrap.lm(formula = bdata$DV7 ~ bdata$DV5 * bdata$DV6, standardized = FALSE)
```

```
# Model: R^2 = .01, F(3, 296) = 1.02, p = .382
```

```
# (Intercept): b = 6.08, SE = 0.67, t(296) = 9.02, p < .001, 95% CI = [4.75, 7.41]
```

```
# DV5: b = -0.20, SE = 0.12, t(296) = -1.75, p = .081, 95% CI = [-0.43, 0.03]
```

```
# DV6: b = -0.15, SE = 0.12, t(296) = -1.21, p = .228, 95% CI = [-0.38, 0.09]
```

```
# DV5 x DV6: b = 0.03, SE = 0.02, t(296) = 1.45, p = .147, 95% CI = [-0.01, 0.07]
```

Example (standardized coefficients)

```
wrap.lm(formula = bdata$DV7 ~ bdata$DV5 * bdata$DV6, standardized = TRUE)
```

```
# Model: R^2 = .01, F(3, 296) = 1.02, p = .382
```

```
# (Intercept): Beta = 0.00
```

```
# DV5: Beta = -0.20, SE = 0.11, t(296) = -1.75, p = .081, 95% CI = [-0.42, 0.02]
```

```
# DV6: Beta = -0.14, SE = 0.11, t(296) = -1.21, p = .228, 95% CI = [-0.36, 0.09]
```

```
# DV5 x DV6: Beta = 0.23, SE = 0.16, t(296) = 1.45, p = .147, 95% CI = [-0.08, 0.53]
```

Correlation Tests

The function **wrap.cor** tests for a correlation between two dependent variables. The function delegates the primary computations to **cor.test** {package: stats}.

Usage & Arguments

```
wrap.cor(dv1, dv2)
```

- dv1, dv2: Column vectors containing the dependent variables.

Example

```
wrap.cor(dv1 = bdata$DV3_T1, dv2 = bdata$DV3_T2)
```

```
# r = -.07, t(298) = -1.21, p = .226, 95% CI = [-.18, .04]
```

Chi-Square Tests

The function **wrap.chi** performs one-way goodness-of-fit and two-way contingency tests. The function delegates the primary operations to **chisq.test** {package: stats}.

Usage & Arguments

```
wrap.chi(dv1, iv1 = NULL, p = rep(1/nlevels(factor(dv1)), nlevels(factor(dv1))), correct = FALSE)
```

- dv1: Column vector containing the dependent variable.
- iv1: Column vector containing the independent variable (*contingency tests, only*).
- p: A vector of probabilities containing expected cell frequencies (*goodness-of-fit tests, only*). By default, the function assumes equal expected cell frequencies for all levels of the dependent variable.
- correct (*default = FALSE*): A logical argument: If FALSE, the function does not apply Yates's correction for 2×2 contingency tables; if TRUE, the function applies Yates's correction for 2×2 contingency tables.

Example (one-way goodness-of-fit test)

```
wrap.chi(dv1 = bdata$DV2)
```

```
# X2(1, N = 300) = 0.48, p = .488
```

Example (two-way contingency test)

```
wrap.chi(dv1 = bdata$DV2, iv1 = bdata$IV2)
```

```
# X2(2, N = 300) = 0.24, p = .887
```

FUNCTIONS FOR DATA VISUALIZATION

Bar Plots

The function **wrap.bar** creates bar plots for numerical dependent variables, adds error bars, and prints descriptive statistics in a summary table. The function creates plots for up to 3 factors total, including 0-1 within-subjects factors and 0-3 between-subjects factors. The function delegates the primary computations to **ggplot** {package: *ggplot2*}. Error bars are $\pm 1 SE$.

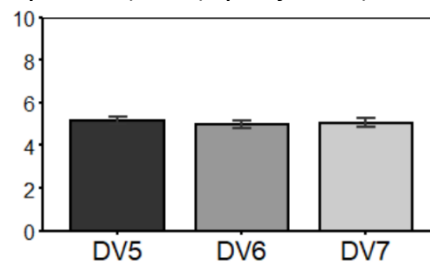
Usage & Arguments

`wrap.bar(dv1, iv1 = NULL, iv2 = NULL, iv3 = NULL)`

- `dv1`: Column vector containing the between-subjects dependent variable OR multiple column vectors containing the within-subjects dependent variables.
- `iv1`, `iv2`, `iv3`: Column vectors containing the independent variables (if applicable).
- *Optional formatting arguments are listed at the bottom of this section.*

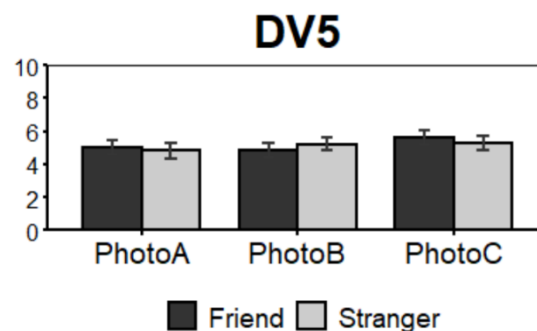
Example (1 within-subjects factor)

`wrap.bar(dv1 = bdata[c(10:12)], ylim = c(0, 10), ymajor = 2)`



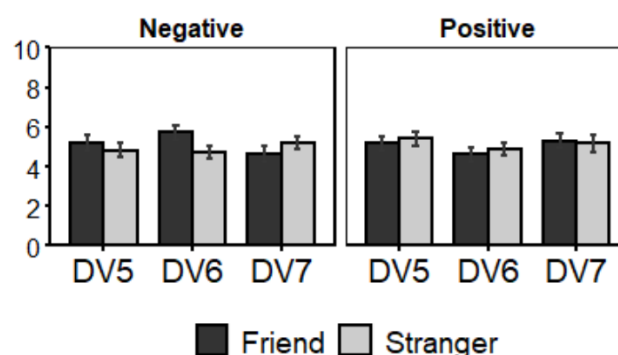
Example (2 between-subjects factors)

`wrap.bar(dv1 = bdata$DV5, iv1 = bdata$IV1, iv2 = bdata$IV2, ylim = c(0, 10), ymajor = 2)`



Example (1 within-subjects factor & 2 between-subjects factors)

`wrap.bar(dv1 = bdata[c(10:12)], iv1 = bdata$IV1, iv2 = bdata$IV3, ylim = c(0, 10), ymajor = 2)`



Optional formatting arguments

- `ylim`: Numeric vector containing lower and upper y-axis limits.
- `ymajor`: Numeric argument representing spacing of y-axis tick marks.
- `ylab`: Character string containing the y-axis label.
- `xlab`: Character string containing the x-axis label.
- `title`: Character string containing the plot title.
- `size.axis.text.y`, `size.axis.text.x`, `size.title`, `size.panel.title`, `size.legend.text`: Numeric arguments containing font sizes.
- `reposition`: Numeric vector to rearrange columns in the summary table and thus reposition factors within the plot itself. For example, `reposition = c(1, 3, 2)` reverses the order of the second and third columns in the summary table and repositions the corresponding factors within the plot. The length of the reposition vector must equal the total number of within-subjects and between-subjects factors.
- `rename1`, `rename2`, `rename3`: String vectors to rename the factor levels in the summary table and thus rename factor levels within the plot itself. For example, `rename1 = c("Close Friend", "Distant Stranger")` renames the levels in the first column of the summary table and renames the corresponding factor levels within the plot. (Note that the function applies the reposition argument to the summary table before applying the rename arguments.)
- `reorder1`, `reorder2`, `reorder3`: String vectors to reorder the factor levels in the summary table and thus reorder factor levels within the plot itself. For example, `reorder1 = c("Stranger", "Friend")` reorders the levels in the first column of the summary table and reorders the corresponding factor levels within the plot. (Note that the function applies the reposition and rename arguments to the summary table before applying the reorder arguments.)

Line Plots

The function **wrap.line** creates line plots for numerical dependent variables, adds error bars, and prints descriptive statistics in a summary table. The function creates plots for up to 3 factors total, including 1 within-subjects factor (corresponding to the default path of each line) and 0-2 between-subjects factors. The function delegates the primary computations to **ggplot** {package: *ggplot2*}. Error bars are $\pm 1 SE$.

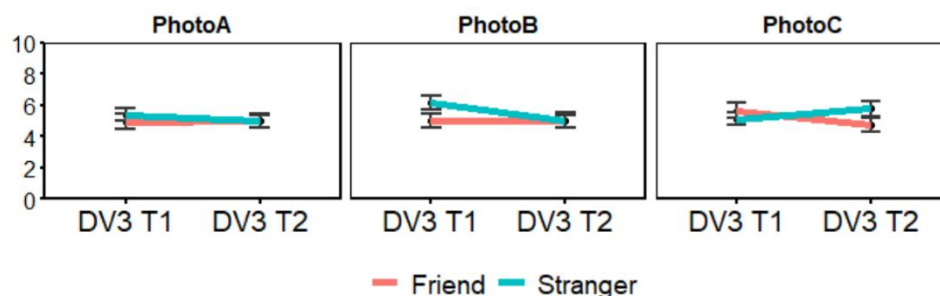
Arguments

`wrap.line(dv1, iv1 = NULL, iv2 = NULL)`

- `dv1`: Multiple column vectors containing the within-subjects dependent variables.
- `iv1`, `iv2`: Column vectors containing the independent variables (if applicable).
- *Optional formatting arguments are identical to those listed above for wrap.bar.*

Example (1 within-subjects factor & 2 between-subjects factors)

`wrap.line(dv1 = bdata[c(6, 8)], iv1 = bdata$IV1, iv2 = bdata$IV2, ylim = c(0, 10), ymajor = 2)`



Histogram Plots

The function **wrap.hist** creates histogram plots for numerical dependent variables and prints counts and percentages for each bin in a summary table. The function delegates the primary computations to **ggplot** {package: *ggplot2*}.

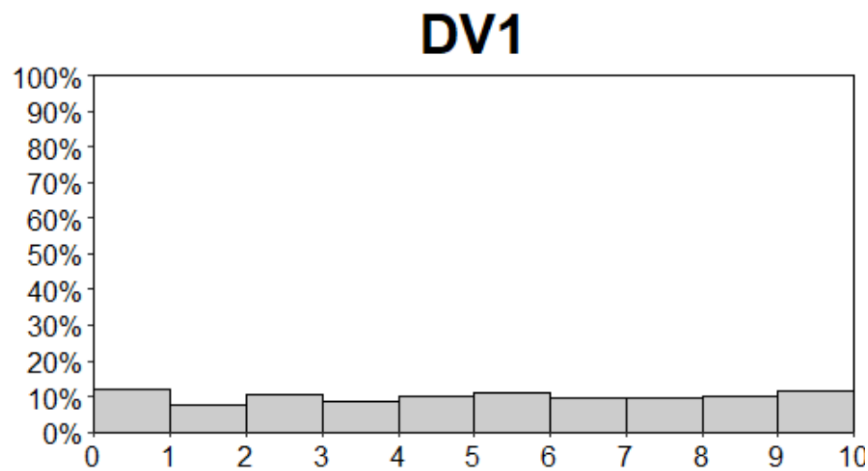
Arguments

`wrap.hist(dv1, likert = FALSE)`

- `dv1`: Column vector containing the dependent variable.
- `likert` (*default = FALSE*): A logical argument: If `FALSE`, the function assumes a continuous dependent variable; if `TRUE`, the function assumes a discrete dependent variable that takes on only integer values.
- *Optional formatting arguments are listed at the bottom of this section.*

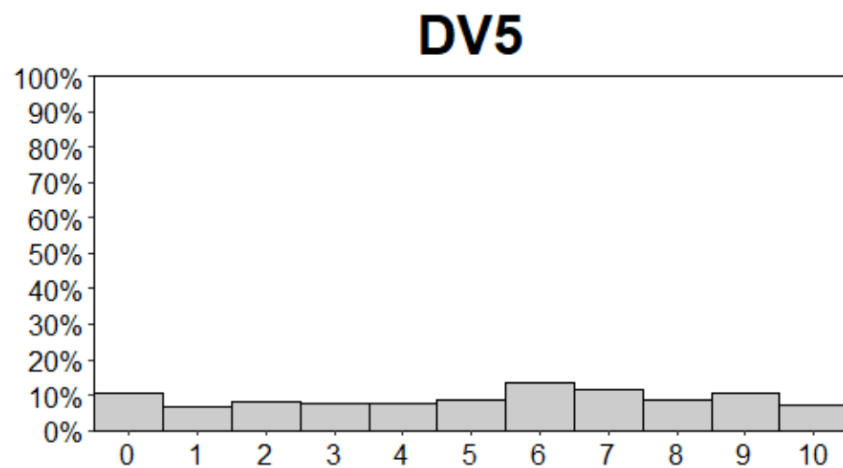
Example (continuous dependent variable)

`wrap.hist(dv1 = bdata$DV1, likert = FALSE)`



Example (discrete dependent variable)

`wrap.hist(dv1 = bdata$DV5, likert = TRUE)`



Optional formatting arguments

- `percent` (*default = TRUE*): A logical argument: If TRUE, the plot displays percentages along the y axis; if FALSE, the plot displays counts.
- `binwidth` (*default = 1*): Numeric argument representing bin width.
- `ylim`: Numeric vector containing y-axis limits.
- `ymajor`: Numeric argument representing spacing of y-axis tick marks.
- `ylab`: Character string containing the y-axis label.
- `title`: Character string containing plot title.
- `xlim`: Numeric vector containing x-axis limits.
- `xmajor`: Numeric argument representing spacing of x-axis labels.
- `xlab`: Character string containing the x-axis label.
- `size.axis.text.x`, `size.axis.text.y`, `size.title`: Numeric arguments containing font sizes.