

# Round robin analyses in R: How to use TripleR

Felix D. Schönbrodt\*    Stefan C. Schmukle†    Mitja D. Back‡

September 24, 2010

## Contents

<b>1</b>	<b>Getting the data into the right format</b>	<b>3</b>
<b>2</b>	<b>How to do the analyses</b>	<b>5</b>
2.1	Univariate manifest analysis	6
2.2	Univariate latent analyses	7
2.3	Bivariate manifest analysis	8
2.4	Bivariate latent analysis	8
2.5	Multiple groups	9
2.6	Missing values	10
2.7	Inspecting the results object	10
<b>3</b>	<b>Plots</b>	<b>12</b>
<b>4</b>	<b>Formatting the output</b>	<b>14</b>
<b>5</b>	<b>Subsequent analyses</b>	<b>16</b>
<b>6</b>	<b>FAQ</b>	<b>19</b>
6.1	This is an excellent introduction - but where can I get more information or pose a question?	19
6.2	How can I calculate a bivariate analysis between one manifest variable and a latent construct indicated by two variables?	19
6.3	This long data format really sounds good. But unfortunately my data already are in the wide format - how can I convert them into the long format?	20
6.4	I have to run many, many round robin analyses in a huge data set. What is the most convenient way to do this?	21
6.5	An error occurs: ‘Aggregation requires fun.aggregate: length used as default’	23
6.6	My original multi group data set has X participants - the effects of the RR analysis, however, only have Y ( $Y < X$ ) rows!	23

---

\*© September 24, 2010, Felix Schönbrodt, Department of Psychology, Ludwig-Maximilians-University, Germany. This package partly was written during a Google Summer of Code 2010 project. Comments on this document may be sent to the author at felix@nicebread.de.

†University of Münster, Germany

‡University of Mainz, Germany

6.7 An example from David Kenny - Comparison with SOREMO.exe	23
<b>References</b> . . . . .	<b>24</b>

TripleR<sup>1</sup> provides functions with a simple, yet powerful interface to calculate round robin analyses in R. We assume that you are already familiar with social relations analyses. If not, a good starter would be David Kenny’s website<sup>2</sup>, or some introductory articles (e.g., Back & Kenny, in press; Kenny, Kashy, & Cook, 2006, especially Ch. 8; Kenny, 1994, for detailed description of the model and the formulae).

If you have already done your round robin study, this document will explain how to get your data into the right format, how to tell TripleR what analyses to do, and how to work with the results. In social relations analyses (SRAs), two notations for the different roles are common. If the investigated phenomenon is a behavior, one usually speaks of *actors* and *partners*. If the investigated phenomenon is interpersonal perception, one speaks of *perceivers* and *targets*. Both groups of labels are interchangeable; in the remainder of this document (as well as in the help files), we will always call them actors and partners.

## 1 Getting the data into the right format

In dyadic data analyses, one often finds two data formats: either the “wide format”, in which each row is one participant, multiple variables or measurements are stored in multiple columns. Concerning round robin data, this would lead to a quadratic matrix with actors as rows and partners as columns. If we have a group of 5 people who rate how much they like each other, the data matrix would look like:

```

      A  B  C  D  E
A NA  3  1  0  5
B  2 NA  5  4  1
C  4  1 NA  6  4
D  0  1  0 NA  4
E  2  2  5  3 NA

```

The most flexible data format, however is the “long format”. In this format each observation is one row, which would look like:

```

actor.id partner.id value
1          A          A    NA
2          B          A     2
3          C          A     4
4          D          A     0
5          E          A     2
6          A          B     3
7          B          B    NA
8          C          B     1
9          D          B     1
10         E          B     2
11         A          C     1
12         B          C     5
13         C          C    NA
14         D          C     0
15         E          C     5
16         A          D     0

```

<sup>1</sup>When you use TripleR in your research, please cite it as Schmukle, S. C., Schönbrodt, F.D., & Back, M. D. (2010). TripleR: A package for round robin analyses using R (version 0.4.5). Retrieved from <http://www.persoc.net/ToolBox/TripleR>.

<sup>2</sup><http://davidakenny.net/kenny.htm>

17	B	D	4
18	C	D	6
19	D	D	NA
20	E	D	3
21	A	E	5
22	B	E	1
23	C	E	4
24	D	E	4
25	E	E	NA

The long format has several advantages:

- Several variables can be stored in one data structure (instead of putting each variable into another quadratic matrix)
- Several groups can be stored in the same data structure by a column indicating the group id
- The data format can be more efficient, as missing values just are missing, and do not occupy a NA place in the matrix (however, as actor ids and partner ids need their own column, the long format has some overhead)
- Data input can be easier, as the order of rows in long format is arbitrary. Each data row is uniquely identified by their actor id and partner id, hence it does not matter whether data entries are grouped along the partner id (as in the example above). You can also group them along the actor id (which could be favorable, as for example the data from one perceiver are typed in one block), or do not group them at all. If you find a lost questionnaire, you can just append it at the end of the long format data frame, regardless of what happened in between.

If the example data set from above would be extended to multiple groups and multiple variables, it would look like:

	actor.id	partner.id	value	value2	group.id
1	A	A	NA	NA	1
2	B	A	2	6	1
3	C	A	4	1	1
4	D	A	0	4	1
5	E	A	2	3	1
6	A	B	3	2	1
7	B	B	NA	NA	1
8	C	B	1	5	1
9	D	B	1	3	1
10	E	B	2	3	1
11	A	C	1	2	1
12	B	C	5	6	1
13	C	C	NA	NA	1
14	D	C	0	4	1
15	E	C	5	3	1
16	A	D	0	2	1
17	B	D	4	3	1
18	C	D	6	5	1
19	D	D	NA	NA	1
20	E	D	3	3	1
21	A	E	5	2	1
22	B	E	1	6	1
23	C	E	4	1	1
24	D	E	4	4	1

25	E	E	NA	NA	1
26	F	F	NA	NA	2
27	G	F	6	3	2
28	H	F	2	5	2
29	I	F	3	3	2
30	J	F	5	3	2
31	F	G	3	2	2
32	G	G	NA	NA	2
33	H	G	3	1	2
34	I	G	6	4	2
35	J	G	2	3	2
36	F	H	5	2	2
37	G	H	4	3	2
38	H	H	NA	NA	2
39	I	H	2	3	2
40	J	H	0	3	2
41	F	I	1	2	2
42	G	I	6	6	2
43	H	I	4	1	2
44	I	I	NA	NA	2
45	J	I	5	3	2
46	F	J	5	2	2
47	G	J	1	3	2
48	H	J	1	5	2
49	I	J	6	3	2
50	J	J	NA	NA	2

Note: The rows where actors == partners (which contain NAs in all measured variables) could have been omitted in the long format. They are only kept for illustration. Furthermore, if you assess self ratings (which would naturally be stored in these fields) they can stay in the data set. These values are automatically set to NA prior to performing the SRAs.

To summarize, for TripleR we need data in the long format. We need at least 3 columns: the actor id, the partner id, and the variable. If multiple variables are assessed, they are coded in a separate column. If multiple groups are assessed, the group id goes into another column. Actor and partner ids have to be unique within each group (i.e., person in different groups can have the same id. To avoid confusions, however, it might be preferable to assign person ids which are unique for the whole data set). Throughout this documentation, the column indicating the actor id is called `actor.id` (the other id columns respectively). Note, however, that you can assign any other name to these columns.

## 2 How to do the analyses

TripleR is capable of doing 4 different types of analyses<sup>3</sup>:

- Univariate manifest analyses (i.e., one measured variable)

<sup>3</sup>Please make sure that you use the most recent version of TripleR (this document was built using TripleR 0.4.5). You can check the installed version using `sessionInfo()`. You can install the latest stable version from CRAN: `install.packages("TripleR", dependencies=TRUE)`. Or, if you are brave, you can install the current developer version from RForge: `install.packages('TripleR', repos='http://www.rforge.net/')`. But be aware that these developer versions might be buggy or produce wrong results. For productive use, we only recommend to use the stable version on CRAN. TripleR depends on some other packages (`reshape`, `plyr`, and `ggplot2`), which have to be installed on your system as well. The parameter `dependencies=TRUE` in the install command forces R to install these additional packages automatically.

- Univariate latent analyses, where two manifest variables are indicators for one latent construct (in the current version, only two manifest variables are possible. Future versions may be able to process an unlimited number of indicators)
- Bivariate manifest analyses (i.e., two measured variables, which are correlated within the SRM)
- Bivariate latent analyses, where each two manifest variables define one latent construct

All of these analyses are possible in a single group (in this case, within group tests for significance are employed), or with multiple groups (in this case, between group t-tests, weighted for group size, are employed).

In the following paragraphs, all four analyses will be shown. Therefore, we load a built in data set from the package. This data set comes from the ‘Mainz Freshman Study’, which assessed liking (‘How much do you like X?’) and meta-liking (‘How much, do you think, does X like you?’) in a group of 54 freshmen, at zero acquaintance:

```
> # load the package
> library(TripleR)
> # load a data set in long format
> data(likingLong)
> #inspect the data set
> head(likingLong, 15)
```

	actor.id	partner.id	liking_a	liking_b	metaliking_a	metaliking_b
1	1	1	NA	NA	NA	NA
2	2	1	4	5	3	2
3	3	1	4	4	4	4
4	4	1	3	3	3	3
5	5	1	5	5	3	3
6	6	1	3	4	4	3
7	7	1	5	4	3	3
8	8	1	4	3	3	3
9	9	1	3	4	3	3
10	10	1	3	3	2	2
11	11	1	3	3	3	3
12	12	1	3	3	3	3
13	13	1	3	3	3	3
14	14	1	5	4	3	3
15	15	1	4	3	3	3

As we can see, both liking and meta-liking have been assessed with two indicators, which allows a latent analyses. But first let’s do an univariate analysis:

## 2.1 Univariate manifest analysis

All analyses can be run with one function: `RR`. For details, you definitely should check the help entry for this function (type `?RR` into the R console). Most parameters of the function are specified via a formula interface. The formula for the current analysis would be: `liking_a ~actor.id * partner.id`. The measured variables are defined in the left part of the formula (left of the `~` sign). The right part defines, which columns in the data frame indicate the actor, the partner, and the group id. These three variables are always given in this order. Actor and partner id are separated by a `*`, which indicates that these factors

are fully crossed (as in the `lm` notation). The group id is separated by a `|`, as in the `lattice` notation.

After the formula, the data frame has to be specified, on which the formula will be applied. Unlike as in the `lm` notation, the data object has to be specified explicitly by `data=...`. Hence, the final command for a univariate manifest analysis is:

```
RR1 <- RR(liking_a ~ actor.id * partner.id, data=likingLong)
```

Please note: all variable names in the formula (i.e., `liking_a`, `actor.id`, and `partner.id`) refer to column names in the specified data frame. They do not have to be like this - if your data frame has other column names your formula might look like `DV a*p`, or anything else.

When we run the command, an object of the class `RR` is returned. If we print the object, a summary of the analysis is printed:

```
> RR1 <- RR(liking_a ~ actor.id * partner.id, data=likingLong)
> RR1
[1] "Round-Robin object ('RR'), calculated by TripleR"
[1] "Univariate analysis of one round robin variable"
[1] "Univariate analyses for: liking_a"
      estimate standardized   se t.value p.value
actor variance      0.172      0.194 0.035  4.914  0.000
partner variance    0.105      0.119 0.022  4.727  0.000
relationship variance 0.609      0.687 0.017 36.827  0.000
error variance      NA         NA    NA    NA    NA
actor-partner covariance 0.014      0.105 0.020  0.703  0.618
relationship covariance 0.080      0.131 0.017  4.809  0.000
[1] "Actor effect reliability: 0.937"
[1] "Partner effect reliability: 0.901"
```

## 2.2 Univariate latent analyses

If you have two indicators to assess a latent construct, error variance can be separated from relationship variance (in the univariate manifest case, error variance is mixed up in the relationship variance component). Two indicators for one latent construct are separated by a `/`. In the current data set, we have two indicators for liking, hence the analysis would look like:

```
> RR2 <- RR(liking_a/liking_b ~ actor.id * partner.id, data=likingLong)
> RR2
[1] "Round-Robin object ('RR'), calculated by TripleR"
[1] "Latent construct analysis of one construct measured by two round robin variables"
[1] "Univariate analyses for: liking_a/liking_b"
      estimate standardized   se t.value p.value
actor variance      0.161      0.164 0.036  4.525  0.000
partner variance    0.105      0.107 0.023  4.678  0.000
relationship variance 0.507      0.518 0.016 31.294  0.000
error variance      0.206      0.211  NA    NA    NA
actor-partner covariance 0.012      0.094 0.021  0.573  0.672
relationship covariance 0.079      0.156 0.016  4.887  0.000
[1] "Actor effect reliability: 0.865"
[1] "Partner effect reliability: 0.893"
[1] "Relationship effect reliability: 0.852"
```

As you can see, the error variance component changed from NA to a meaningful value. For the error component no significance tests are provided. Furthermore, in the single group case we are not aware of an approach to calculate the significance of the latent covariances (in the multi group case, however, they can be calculated, see below).

### 2.3 Bivariate manifest analysis

If you have two different variables (each assessing another construct), bivariate SRAs can be performed. Two different variables are separated by a + on the left hand side of the formula. In the current example, we can examine the relationship between liking and meta-liking, by typing:

```
> RR3 <- RR(liking_a+metaliking_a ~ actor.id * partner.id, data=likingLong)
> RR3
[1] "Round-Robin object ('RR'), calculated by TripleR"
[1] "Bivariate analysis of two variables, each measured by one round robin variable"
[1] "Univariate analyses for: liking_a"
      estimate standardized      se t.value p.value
actor variance      0.172      0.194 0.035  4.914  0.000
partner variance    0.105      0.119 0.022  4.727  0.000
relationship variance 0.609      0.687 0.017 36.827  0.000
error variance      NA          NA    NA    NA    NA
actor-partner covariance 0.014      0.105 0.020  0.703  0.618
relationship covariance 0.080      0.131 0.017  4.809  0.000
[1] "Actor effect reliability: 0.937"
[1] "Partner effect reliability: 0.901"
[1] "Univariate analyses for: metaliking_a"
      estimate standardized      se t.value p.value
actor variance      0.140      0.233 0.028  4.953  0.000
partner variance    0.027      0.044 0.007  4.005  0.000
relationship variance 0.436      0.723 0.012 36.767  0.000
error variance      NA          NA    NA    NA    NA
actor-partner covariance 0.002      0.031 0.010  0.195  0.779
relationship covariance 0.062      0.143 0.012  5.247  0.000
[1] "Actor effect reliability: 0.944"
[1] "Partner effect reliability: 0.764"
[1] "Bivariate analyses:"
      estimate standardized      se t.value p.value
actor-actor covariance 0.072      0.462 0.025  2.900  0.015
partner-partner covariance 0.049      0.920 0.011  4.310  0.000
actor-partner covariance 0.014      0.206 0.011  1.258  0.359
partner-actor covariance 0.000      0.003 0.018  0.021  0.794
intrapersonal relationship covariance 0.289      0.560 0.011 25.498  0.000
interpersonal relationship covariance 0.067      0.129 0.011  5.893  0.000
```

In this case, we get three different outputs: univariate analyses for each of the both variables, and a third section containing the bivariate analyses (i.e., all possible covariances between the social relations effects from both variables).

### 2.4 Bivariate latent analysis

In this case, two latent constructs are measured by two indicators each. In the current example, we have two indicators for liking and for metaliking. Applying the same logic as before, the command now is:

```

> RR4 <- RR(liking_a/liking_b + metaliking_a/metaliking_b
+          ~ actor.id * partner.id, data=likingLong)
> RR4
[1] "Round-Robin object ('RR'), calculated by TripleR"
[1] "Bivariate analysis of two constructs, each measured by two round robin variables"
[1] "Univariate analyses for: liking_a/liking_b"
      estimate standardized   se t.value p.value
actor variance      0.161      0.164 0.036  4.525  0.000
partner variance    0.105      0.107 0.023  4.678  0.000
relationship variance 0.507      0.518 0.016 31.294  0.000
error variance      0.206      0.211  NA      NA      NA
actor-partner covariance 0.012      0.094 0.021  0.573  0.672
relationship covariance 0.079      0.156 0.016  4.887  0.000
[1] "Actor effect reliability: 0.865"
[1] "Partner effect reliability: 0.893"
[1] "Relationship effect reliability: 0.852"
[1] "Univariate analyses for: metaliking_a/metaliking_b"
      estimate standardized   se t.value p.value
actor variance      0.148      0.217 0.031  4.730  0.000
partner variance    0.026      0.038 0.007  3.980  0.000
relationship variance 0.357      0.522 0.012 30.776  0.000
error variance      0.153      0.223  NA      NA      NA
actor-partner covariance 0.000      0.002 0.011  0.014  0.794
relationship covariance 0.071      0.197 0.012  6.075  0.000
[1] "Actor effect reliability: 0.899"
[1] "Partner effect reliability: 0.761"
[1] "Relationship effect reliability: 0.841"
[1] "Bivariate analyses:"
      estimate standardized   se t.value p.value
actor-actor covariance    0.092      0.593 0.027  3.370  0.004
partner-partner covariance 0.049      0.928 0.011  4.287  0.000
actor-partner covariance  0.007      0.114 0.011  0.676  0.630
partner-actor covariance  0.004      0.032 0.019  0.209  0.777
intrapersonal relationship covariance 0.330      0.774 0.012 28.570  0.000
interpersonal relationship covariance 0.075      0.177 0.012  6.532  0.000

```

Now we get a comparable output to the bivariate manifest analysis, only that now the error variance can be separated from the relationship variance.

## 2.5 Multiple groups

Using the formula interface, analyses with multiple groups can be performed as well. The only extension is, that the variable which identifies group membership is specified at the end of the formula after a | sign. For example, we load another built in data set which consists of 10 groups. Two variables are measured: **ex** is a round robin extraversion rating, **ne** is a neuroticism rating (self ratings for both variables also are included).

```

> data(multiGroup)
> RR1m <- RR(ex~actor.id*partner.id|group.id, data=multiGroup, na.rm=TRUE)
> RR1m
[1] "Round-Robin object ('RR'), calculated by TripleR"
[1] "Univariate analysis of one round robin variable in multiple groups"
[1] "Group descriptives: n = 10 ; average group size = 19.4 ; range: 15 - 24"
[1] "Univariate analyses for: ex"
      estimate standardized   se t.value p.value
actor variance      0.242      0.100 0.033  7.238  0.000
partner variance    0.898      0.373 0.147  6.103  0.000
relationship variance 1.270      0.527 0.056 22.634  0.000
error variance      NA      NA      NA      NA      NA

```

```

actor-partner covariance    0.018          0.039 0.051   0.353   0.732
relationship covariance    0.112          0.088 0.041   2.706   0.024
[1] "Actor effect reliability: 0.777"
[1] "Partner effect reliability: 0.928"

```

Any formula explained above can be extended by the multi group parameter. Concerning the output, no differences can be seen (except the second line of the output, which always displays the type of analysis: "Univariate analysis of one round robin variable in multiple groups").

As already described, one computational difference is the usage of between group t-tests, instead of the within group method. Another difference is the results object: all univariate analyses are contained (although, not displayed by the `print` function) in the results. More details on the results object can be found in the section [2.7](#).

## 2.6 Missing values

Missing values can be handled in TripleR. By default, calculations are aborted if missing values are outside the diagonale of the round robin matrix. To allow missing values, add the argument `na.rm="remove"`, or `na.rm="impute"`.

You can inspect the distribution of missing values by using the `plot_missings` command. It takes the same parameters as an univariate manifest RR analysis; for details see the help files.

## 2.7 Inspecting the results object

When a round robin analysis is performed (and stored in an object), not all information is displayed. When the object is printed (either by `print(object)`, or by simple writing the name of the object, e.g. `RR1`), a custom `print` function is called, which displays the table of variance components, effects reliability estimates, and some other information. During the calculation, however, much more results are computed and stored in the object.

To see the structure of the object type `str(object)`:

```

> str(RR1)
List of 9
 $ effects      :'data.frame':      54 obs. of  3 variables:
  ..$ id        : Factor w/ 54 levels "1","10","11",...: 1 12 23 34 45 51 52 53 54 2 ...
  ..$ liking_a.a: atomic [1:54] -0.477 0.276 -0.324 -0.323 0.198 ...
  .. ..- attr(*, "reliability")= num 0.937
  ..$ liking_a.p: atomic [1:54] 0.2639 -0.854 0.3611 0.4177 0.0125 ...
  .. ..- attr(*, "reliability")= num 0.901
 $ effectsRel  :'data.frame':      2862 obs. of  4 variables:
  ..$ actor.id   : int [1:2862] 1 2 1 3 1 4 1 5 1 6 ...
  ..$ partner.id : int [1:2862] 2 1 3 1 4 1 5 1 6 1 ...
  ..$ dyad       : chr [1:2862] "01_02" "01_02" "01_03" "01_03" ...
  ..$ relationship: num [1:2862] 1.1509 0.2805 -0.0642 0.8802 -1.1208 ...
 $ effects.gm  :'data.frame':      54 obs. of  3 variables:
  ..$ id        : Factor w/ 54 levels "1","10","11",...: 1 12 23 34 45 51 52 53 54 2 ...
  ..$ liking_a.a: num [1:54] 2.7 3.46 2.86 2.86 3.38 ...
  ..$ liking_a.p: num [1:54] 3.44 2.33 3.54 3.6 3.19 ...
 $ varComp    :'data.frame':      6 obs. of  6 variables:
  ..$ type       : Factor w/ 6 levels "actor variance",...: 1 4 6 3 2 5
  ..$ estimate   : num [1:6] 0.1717 0.1053 0.6088 NA 0.0141 ...
  ..$ standardized: num [1:6] 0.194 0.119 0.687 NA 0.105 ...

```

```

..$ se          : num [1:6] 0.0349 0.0223 0.0165 NA 0.02 ...
..$ t.value     : num [1:6] 4.914 4.727 36.827 NA 0.703 ...
..$ p.value     : num [1:6] 1.57e-05 2.98e-05 1.35e-39 NA 6.18e-01 ...
$ relMat.av    : num [1:54, 1:54] NA 0.716 0.408 -0.621 0.321 ...
..- attr(*, "group.id")= chr "1"
..- attr(*, "varname")= chr "liking_a"
..- attr(*, "dimnames")=List of 2
.. ..$ : chr [1:54] "1" "2" "3" "4" ...
.. ..$ : chr [1:54] "1" "2" "3" "4" ...
$ relMat.diff: num [1:54, 1:54] NA -0.87 0.944 1 2.074 ...
..- attr(*, "group.id")= chr "1"
..- attr(*, "varname")= chr "liking_a"
..- attr(*, "dimnames")=List of 2
.. ..$ : chr [1:54] "1" "2" "3" "4" ...
.. ..$ : chr [1:54] "1" "2" "3" "4" ...
$ group.size   : int 54
$ latent       : logi FALSE
$ anal.type    : chr "Univariate analysis of one round robin variable"
- attr(*, "class")= chr "RRuni"
- attr(*, "group.size")= int 54
- attr(*, "varname")= chr "liking_a"

```

Multiple data structures are stored in the object in list mode. Some objects are for internal use, others, however, are very important for subsequent analyses (see section 5). You can access all stored objects via the `$` operator. For example, the actor and partner effects are stored in the `effects` object:

```

> head(RR1$effects)
  id liking_a.a liking_a.p
1  1 -0.4768519  0.26388889
2  2  0.2756410 -0.85398860
3  3 -0.3240741  0.36111111
4  4 -0.3230057  0.41773504
5  5  0.1976496  0.01246439
6  6  1.0544872  0.42485755

```

Following data objects might be relevant for subsequent analyses:

**effects** The actor and partner effects. You access each effect by another `$` operator; the effects have the same name like the original variable with a suffix for actor and partner effect. Default suffixes are ‘.a’ for actor and ‘.p’ for partner effect. For example, if your original variable is called `liking`, you can access the actor effect by `RR1$effects$liking.a`. If self ratings are present in the data set, they are also returned with the default suffix `.s`. You can inspect the effects by typing `str(RR1$effects)`.

**effects.gm** Actor and partner effects with group mean added

**effectsRel** A data frame in long format which corresponds to the `n x n` matrix of relationship effects

**varComp** A data frame with the absolute and standardized variance components and their respective significance tests (this object is printed into the `print` function of an `RR` object)

**group.var** In the multi group case: display group variance

In section 5 (Subsequent Analyses) it is explained how follow up analyses using the actor and partner effects, and the variance components can be done.

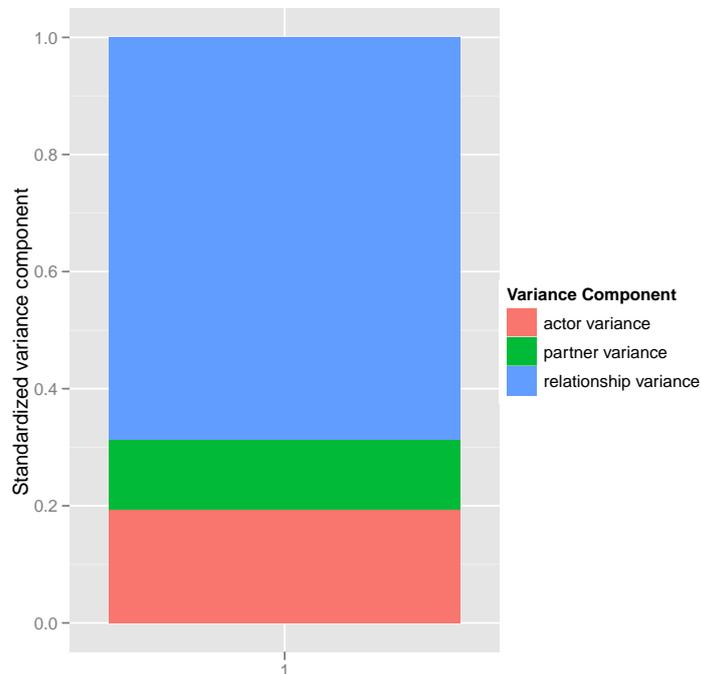


Figure 1: Variance decomposition of a single round robin group

### 3 Plots

Several plots can be made from the result objects. Simply type `plot(RR_object)` to see the standard variance plot associated with each analysis. The main difference between plots is whether you have multiple groups or a single round robin group.

```
> # see Figure 1
> plot(RR1)
```

```
> # see Figure 2
> plot(RR1m)
```

You can also try different parameters:

**measure** = `behavior` (default) or `perception`: changes the labels of the plots

**geom (single groups)** = `bar` (default) or `pie`: show variance components as stacked bars or as a pie chart

**geom (multiple groups)** = `scatter` (default) or `bar`: show variance components of all groups as scatter plots with confidence intervals or as a bar charts

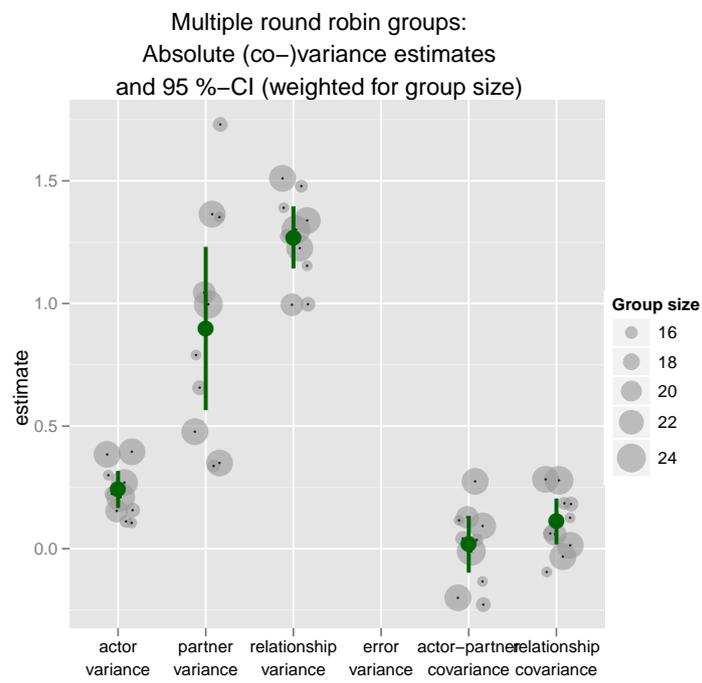


Figure 2: Variance decomposition of multiple round robin groups

**connect (multiple groups)** = FALSE (default) or TRUE: connect the dots of each group in the scatter plot (usually this looks very cluttered and should not be turned on)

**conf.level (multiple groups)** (defaults to 0.95) defines the size of the confidence interval in the scatter plot

Hence you can try several combinations of these parameters, e.g.:

```
> plot(RR1, measure="perception", geom="pie")
> plot(RR1, measure="behavior", geom="pie")
> plot(RR1m, measure="perception", geom="bar")
> plot(RR1m, conf.level=0.5, connect=TRUE)
```

The plot function returns a `ggplot2` object, which in turn can be altered (e.g., you can change the title, the axes labels, the colors, etc.). For more information, please consult the `ggplot2` documentation.

## 4 Formatting the output

As mentioned above, two nomenclatures have been established, depending on whether behaviors or interpersonal perceptions are assessed. While internally always the labels *actor* and *partner* are used, the summary output can be customized by specifying whether the measure is a **behavior** or a **perception** (default is *behavior*). In bivariate analyses, both variables can be specified, e.g. `measure1='behavior', measure2='perception'`, or all other combinations, e.g.:

```
> print(RR1, measure1="perception")
[1] "Round-Robin object ('RR'), calculated by TripleR"
[1] "Univariate analysis of one round robin variable"
[1] "Univariate analyses for: liking_a"
              estimate standardized   se t.value p.value
perceiver variance      0.172      0.194 0.035   4.914  0.000
target variance         0.105      0.119 0.022   4.727  0.000
relationship variance   0.609      0.687 0.017  36.827  0.000
error variance           NA         NA    NA     NA     NA
perceiver-target covariance 0.014      0.105 0.020   0.703  0.618
relationship covariance   0.080      0.131 0.017   4.809  0.000
[1] "Perceiver effect reliability: 0.937"
[1] "Target effect reliability: 0.901"
> print(RR4, measure1="behavior", measure2="perception")
[1] "Round-Robin object ('RR'), calculated by TripleR"
[1] "Bivariate analysis of two constructs, each measured by two round robin variables"
[1] "Univariate analyses for: liking_a/liking_b"
              estimate standardized   se t.value p.value
actor variance      0.161      0.164 0.036   4.525  0.000
partner variance    0.105      0.107 0.023   4.678  0.000
relationship variance 0.507      0.518 0.016  31.294  0.000
error variance       0.206      0.211    NA     NA     NA
actor-partner covariance 0.012      0.094 0.021   0.573  0.672
relationship covariance 0.079      0.156 0.016   4.887  0.000
[1] "Actor effect reliability: 0.865"
[1] "Partner effect reliability: 0.893"
[1] "Relationship effect reliability: 0.852"
[1] "Univariate analyses for: metaliking_a/metaliking_b"
              estimate standardized   se t.value p.value
```

```

perceiver variance          0.148          0.217 0.031   4.730   0.000
target variance            0.026          0.038 0.007   3.980   0.000
relationship variance      0.357          0.522 0.012  30.776   0.000
error variance             0.153          0.223   NA      NA      NA
perceiver-target covariance 0.000          0.002 0.011   0.014   0.794
relationship covariance    0.071          0.197 0.012   6.075   0.000
[1] "Perceiver effect reliability: 0.899"
[1] "Target effect reliability: 0.761"
[1] "Relationship effect reliability: 0.841"
[1] "Bivariate analyses:"
                                estimate standardized   se t.value p.value
actor-perceiver covariance      0.092          0.593 0.027   3.370   0.004
partner-target covariance       0.049          0.928 0.011   4.287   0.000
actor-target covariance         0.007          0.114 0.011   0.676   0.630
partner-perceiver covariance    0.004          0.032 0.019   0.209   0.777
intrapersonal relationship covariance 0.330          0.774 0.012  28.570   0.000
interpersonal relationship covariance 0.075          0.177 0.012   6.532   0.000
> print(RR4, measure1="perception", measure2="metaperception")
[1] "Round-Robin object ('RR'), calculated by TripleR"
[1] "Bivariate analysis of two constructs, each measured by two round robin variables"
[1] "Univariate analyses for: liking_a/liking_b"
                                estimate standardized   se t.value p.value
perceiver variance otherperception 0.161          0.164 0.036   4.525   0.000
target variance otherperception    0.105          0.107 0.023   4.678   0.000
relationship variance otherperception 0.507          0.518 0.016  31.294   0.000
error variance otherperception      0.206          0.211   NA      NA      NA
generalized reciprocity otherperception 0.012          0.094 0.021   0.573   0.672
dyadic reciprocity otherperception  0.079          0.156 0.016   4.887   0.000
[1] "Perceiver effect reliability: 0.865"
[1] "Target effect reliability: 0.893"
[1] "Relationship effect reliability: 0.852"
[1] "Univariate analyses for: metaliking_a/metaliking_b"
                                estimate standardized   se t.value p.value
perceiver variance metaperception  0.148          0.217 0.031   4.730   0.000
target variance metaperception     0.026          0.038 0.007   3.980   0.000
relationship variance metaperception 0.357          0.522 0.012  30.776   0.000
error variance metaperception      0.153          0.223   NA      NA      NA
generalized reciprocity metaperception 0.000          0.002 0.011   0.014   0.794
dyadic reciprocity metaperception  0.071          0.197 0.012   6.075   0.000
[1] "Perceiver effect reliability: 0.899"
[1] "Target effect reliability: 0.761"
[1] "Relationship effect reliability: 0.841"
[1] "Bivariate analyses:"
                                estimate standardized   se t.value p.value
Perceiver assumed reciprocity      0.092          0.593 0.027   3.370   0.004
Generalized assumed reciprocity     0.049          0.928 0.011   4.287   0.000
Perceiver meta-accuracy             0.007          0.114 0.011   0.676   0.630
Generalized meta-accuracy           0.004          0.032 0.019   0.209   0.777
Dyadic assumed reciprocity          0.330          0.774 0.012  28.570   0.000
Dyadic meta-accuracy                0.075          0.177 0.012   6.532   0.000

```

Possible combinations are for the univariate case: `measure=c('behavior', 'perception')`; and for the bivariate case: `measure1 = c('behavior', 'perception'), measure2 = c('behavior', 'perception')`, and the special case `measure1='perception', measure2='metaperception'`.

As you can see, typical labels from different research traditions, like 'generalized reciprocity metaperception' or 'perceiver meta-accuracy' are automatically printed to ease interpretation of the results.

A convenient short cut to achieve this styling is the function `RR.style`. You can call this function once at the beginning of your script, and all subsequent

analyses will be labelled accordingly.

## 5 Subsequent analyses

Usually one does not only want to know about the variance components and the within-SRM correlations. Often, we want to correlate the actor and partner effects with the self-ratings, with external personality questionnaires, or demographic variables. To do this, we can extract the actor/ partner effects from the RR-object, combine them with the other data (e.g., self ratings) in another data frame, and do which ever analysis we like.

*Be careful:* in RR objects one cannot be sure about the order and the completeness of actor/ partner effects. That means, actors can be reordered and their order might be different from that in the original data set. Furthermore, if some participants are only actors or only partners they are removed prior to the social relations analyses, and do not appear in the actor/ partner effects. Hence, merging of RR effects and other data *always* has to be done using the **merge** command. As non-round robin variables usually are assigned to the actor id, consequently merging should be done along the actor id).

The data set **multiGroup** contains round robin ratings and self ratings of extraversion, which will serve as an extended example:

```
> # calculate the SRM
> data(multiGroup)
> RR.style("perception")
> RRlm <- RR(ex~actor.id*partner.id|group.id, data=multiGroup, na.rm=TRUE)
> RRlm

[1] "Round-Robin object ('RR'), calculated by TripleR"
[1] "Univariate analysis of one round robin variable in multiple groups"
[1] "Group descriptives: n = 10 ; average group size = 19.4 ; range: 15 - 24"
[1] "Univariate analyses for: ex"

```

	estimate	standardized	se	t.value	p.value
perceiver variance	0.242	0.100	0.033	7.238	0.000
target variance	0.898	0.373	0.147	6.103	0.000
relationship variance	1.270	0.527	0.056	22.634	0.000
error variance	NA	NA	NA	NA	NA
perceiver-target covariance	0.018	0.039	0.051	0.353	0.732
relationship covariance	0.112	0.088	0.041	2.706	0.024

```
[1] "Perceiver effect reliability: 0.777"
[1] "Target effect reliability: 0.928"
> # extract the actor and partner effects
> eff <- RRlm$effects
> head(eff)

```

	id	ex.p	ex.t	ex.s	group.id
90201	90201	-0.721568627	0.8078431	1.6654412	2
90205	90205	-0.227450980	0.7137255	0.6654412	2
90207	90207	-0.007843137	-1.7725490	-1.3345588	2
90209	90209	0.003921569	2.4156863	2.6654412	2
90210	90210	-0.066666667	1.2862745	1.6654412	2
90212	90212	-0.058823529	-0.5882353	1.6654412	2

As actor and partner effects are corrected for group membership in  $g$  groups, according to Kenny et al. (2006) partial correlations should be used when these effects are correlated with external (non-SRM) variables (i.e. external variables like self ratings also have to be controlled for group membership). ‘Controlling for group membership’ by  $g-1$  dummy variables is equivalent to group centering

all measures. As the self ratings returned by `RR$effects` already are centered on group level, all variables (actor & partner effects, self ratings) already are controlled for group membership.

Correlations between group centered variables and partial correlations between their non-centered counterparts *controlled for group membership* are exactly the same. However, when controlling for group membership, one loses  $g-1$  degrees of freedom, hence their test of significance is more conservative.

Practically, you can run a simple correlation between the group centered measures and calculate the p-value 'by hand' and adjust the degrees of freedom (see example below). Alternatively, you can use specialized packages for partial correlations (e.g. the functions `pcor` and `pcor.test` in the package `ggm`) to run these analyses.

Here is a step-by-step example for the calculation of bivariate correlations between the target effect and the self rating:

```
> # correlate effects and self ratings
> c1 <- cor(eff$ex.t, eff$ex.s, use="p")
> print(round(c1, 3))
[1] 0.628
> # Be careful: when calculating partial correlations,
> # the degrees of freedom have to be adjusted by the number of groups - 1
>
> #Calculate the t value by hand:
>
> # k = number of control parameters: number of groups - 1
> k <- length(levels(factor(multiGroup$group.id)))-1
> n <- nrow(eff) # n = number of participants
> df <- n-2-k
> t.value <- c1*sqrt((n-2-k)/(1-c1^2))
> p.value <- dt(t.value, df=df)
> round(p.value, 4)
[1] 0
```

In this analysis, we find a considerable self-other agreement of extraversion ratings  $r_{ex.target,ex.self} = 0.628$ .

Correlations which are calculated by `SOREMO.exe` are by default disattenuated for actor and/or partner effect unreliability. To replicate these results, you have to disattenuate the obtained correlations by following formula:

$$r_{disatt} = r_{raw} * \frac{1}{\sqrt{Rel_{targeteffect}}}$$

Hence, the disattenuated correlation  $r_{ex.target,ex.self}$  would be  $0.628 * \frac{1}{\sqrt{0.928}} = 0.652$ .

If you have other external variables (except the self rating), you have to group center them before doing this approach. The variable `narc` (= narcissism) in the data set `multiNarc` is such a variable: it is a self rating of narcissism.

```
> # calculate the SRM
> data(multiGroup)
> data(multiNarc)
> RR.style("perception")
> RR1m <- RR(ex~actor.id*partner.id|group.id, data=multiGroup, na.rm=TRUE)
> RR1m
[1] "Round-Robin object ('RR'), calculated by TripleR"
[1] "Univariate analysis of one round robin variable in multiple groups"
[1] "Group descriptives: n = 10 ; average group size = 19.4 ; range: 15 - 24"
[1] "Univariate analyses for: ex"
```

```

              estimate standardized   se t.value p.value
perceiver variance      0.242          0.100 0.033   7.238  0.000
target variance         0.898          0.373 0.147   6.103  0.000
relationship variance    1.270          0.527 0.056  22.634  0.000
error variance          NA             NA     NA     NA     NA
perceiver-target covariance 0.018        0.039 0.051   0.353  0.732
relationship covariance  0.112          0.088 0.041   2.706  0.024
[1] "Perceiver effect reliability: 0.777"
[1] "Target effect reliability: 0.928"
> # extract the actor and partner effects
> eff <- RRlm$effects
> dataset <- merge(eff, multiNarc, by=c("id", "group.id"))
> # group center narcissism
> dataset$narc.gc <- lm(narc~factor(group.id), dataset)$resid
> c1 <- cor(dataset$ex.t, dataset$narc.gc)
> df <- nrow(dataset) - 2 - (length(RRlm$groups)-1)
> (p.value <- dt(c1*sqrt((df)/(1-c1^2)), df=df))
[1] 1.094988e-36

```

Using the approach of group centering, groups are treated as fixed factors. Both conceptually and by means of computations it might be preferable to treat groups as random factors (which, however, requires a sufficient number of groups). When using a multilevel approach, we would like to keep the group variance in our dependent variable (as the multilevel modeling takes care of this), hence we use the effects with group mean added (`effects.gm`) and the raw self ratings. Using a multilevel modeling approach, the calculation would look like the following:

```

> library(lme4)
> eff.gm <- RRlm$effects.gm
> # scale all continuous variables to obtain standardized estimates
> eff.gm[,2:4] <- apply(eff.gm[,2:4], 2, scale)
> # Allow the intercept to vary between groups
> # (this is equivalent to the fixed effects approach, only with random effects).
> # Additionally, allow slopes to vary:
> lmer(ex.s~ex.t + (ex.t|group.id), eff.gm)
Linear mixed model fit by REML
Formula: ex.s ~ ex.t + (ex.t | group.id)
Data: eff.gm
   AIC   BIC logLik deviance REMLdev
471.5 491.1 -229.7   451.6   459.5
Random effects:
 Groups   Name      Variance Std.Dev. Corr
group.id (Intercept) 0.00000  0.00000
          ex.t       0.00000  0.00000   NaN
Residual              0.60677  0.77895
Number of obs: 194, groups: group.id, 10

Fixed effects:
              Estimate Std. Error t value
(Intercept) -2.803e-16  5.593e-02   0.00
ex.t         6.296e-01  5.607e-02  11.23

Correlation of Fixed Effects:
      (Intr)
ex.t 0.000

```

The multilevel analysis reveals a self-other agreement of extraversion ratings  $\beta_{ex.target,ex.self} = .630$ . As there is no random variance of the group level in

this analysis (and also no random variance of the slopes), the result is virtually the same as in the fixed effects analysis.

For principal reasons, the `lme4` package does not report p values, as it is not clear how to compute the degrees of freedoms in multilevel models<sup>4</sup>. For practical reasons, however, with sufficient degrees of freedom the  $t$  distribution converges to the  $z$  distribution. Hence, the reported  $t$  value still can be examined. Some authors argue that absolute  $t$  values  $> 2$  can be judged as significant, regardless of the actual  $df$  (e.g., Baayen, Davidson, & Bates, 2008; Kliegl, Masson, & Richter, 2010).

For subsequent analyses of relationship effects, please note that in contrast to actor and partner effects, relationship effects have another structure: they are nested in each dyad. Hence, in this case a dyadic data analysis such as the actor-partner interdependence model (APIM) has to be conducted (see Kenny, Kashy & Cook, 2006, p. 210). Relationship effects can be retrieved from the RR object by typing `RR1m$effectsRel`.

## 6 FAQ

### 6.1 This is an excellent introduction - but where can I get more information or pose a question?

The best way is to join the tripler-info mailing list on R-Forge. Bug reports, questions, or praise can be put on this list; important announcements (new versions, functions, etc.) also are posted on this list:

<http://lists.r-forge.r-project.org/mailman/listinfo/tripler-info>

### 6.2 How can I calculate a bivariate analysis between one manifest variable and a latent construct indicated by two variables?

A natural application of the formula interface would be:

```
RR1 <- RR(liking_a + metaliking_a / metaliking_b ~actor.id *
          partner.id, data=likingLong)
```

This approach, however, does not work in the current version of TripleR. However, you can do the analysis by first creating a new variable for the latent construct by taking the mean of both indicators for metaliking. Then, you can perform a normal bivariate manifest analysis:

```
RR1 <- RR(liking_a + metaliking_latent ~actor.id * partner.id,
          data=likingLong)
```

---

<sup>4</sup><https://stat.ethz.ch/pipermail/r-help/2006-May/094765.html>, also see several lengthy discussions on the R-sig-ME mailing list

### 6.3 This long data format really sounds good. But unfortunately my data already are in the wide format - how can I convert them into the long format?

Converting data from wide to long is relatively easy in R. If you have quadratic matrices, TripleR provides a function which converts these data into long format. For example, in the package is a built in data set (`liking_a`), which is in wide format:

```
> data(liking_a)
> head(liking_a)
  V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21 V22 V23 V24
1 NA 3 3 2 2 4 3 3 2 3 3 2 2 3 2 3 2 3 2 3 2 2 3 3
2 4 NA 3 4 3 4 3 2 2 3 2 3 3 3 4 3 2 3 3 4 4 4 3 4
3 4 3 NA 3 3 3 4 3 2 3 2 3 1 4 2 4 0 3 2 3 2 3 3 2
4 3 3 3 NA 4 2 1 2 3 2 2 4 2 3 2 3 2 4 4 3 3 3 2 2
5 5 4 4 4 NA 4 3 2 3 3 4 3 2 4 3 4 3 4 4 4 2 3 3 4
6 3 3 4 4 3 4 NA 5 5 3 4 5 4 4 5 4 5 4 4 5 5 4 5 4 3
  V25 V26 V27 V28 V29 V30 V31 V32 V33 V34 V35 V36 V37 V38 V39 V40 V41 V42 V43 V44 V45 V46
1 3 3 3 3 3 2 2 3 1 3 3 3 2 2 3 3 3 3 3 3 2 3
2 3 4 4 4 3 4 4 4 4 4 4 2 3 4 4 4 4 4 4 4 3 4 3
3 1 2 3 2 3 2 4 2 4 4 3 2 3 3 3 2 4 3 2 4 3 2
4 3 3 3 3 3 3 2 3 4 3 3 3 2 4 3 3 3 3 3 3 4 3 2
5 3 4 4 4 3 3 3 4 4 2 4 4 4 4 3 3 4 4 4 3 3 3
6 3 4 5 5 4 4 5 4 3 5 4 5 5 4 4 4 5 4 4 5 3 4
  V47 V48 V49 V50 V51 V52 V53 V54
1 3 3 3 3 3 3 3
2 4 4 3 4 3 4 4
3 3 4 4 3 3 4 4
4 3 3 3 3 3 3 2
5 3 2 4 3 2 3 3
6 3 5 4 4 5 5 5
```

To convert this into long format you can use the function `matrix2long`:

```
> long <- matrix2long(liking_a)
> str(long)
'data.frame':      2916 obs. of  3 variables:
 $ actor.id : int  1 2 3 4 5 6 7 8 9 10 ...
 $ partner.id: int  1 1 1 1 1 1 1 1 1 1 ...
 $ value    : int  NA 4 4 3 5 3 5 4 3 3 ...
```

Now you can run the SRAs as usual using the data frame `long`. If you assessed multiple variables (and now have a separate matrix for each variable), you have to get each variable into long format and then combine all long data frames using `merge` (in the final data frame, each variable should be a separate column):

```
> data(liking_a)
> data(liking_b)
> long_a <- matrix2long(liking_a, var.id="liking_a")
> long_b <- matrix2long(liking_b, var.id="liking_b")
> long <- merge(long_a, long_b, by=c("actor.id", "partner.id"))
> str(long)
'data.frame':      2916 obs. of  4 variables:
 $ actor.id : int  1 1 1 1 1 1 1 1 1 1 ...
 $ partner.id: int  1 10 11 12 13 14 15 16 17 18 ...
 $ liking_a  : int  NA 3 3 2 2 3 2 3 2 3 ...
 $ liking_b  : int  NA 2 2 1 2 3 3 3 2 3 ...
```

If you have multiple groups, all transformed long data frames are combined *row wise* and an additional column is necessary to indicate the group id. In lack of appropriate demo data, for the following example imagine that `liking_a` is the liking rating in group A, and `liking_b` is the liking rating in another group B. Hence, one would combine both as following:

```
> data(liking_a)
> data(liking_b)
> long_a <- matrix2long(liking_a, var.id="liking")
> long_b <- matrix2long(liking_b, var.id="liking")
> # add group id
> long_a$group.id <- 1
> long_b$group.id <- 2
> long2 <- rbind(long_a, long_b)
> str(long2)
'data.frame':      5832 obs. of  4 variables:
 $ actor.id  : int  1 2 3 4 5 6 7 8 9 10 ...
 $ partner.id: int  1 1 1 1 1 1 1 1 1 1 ...
 $ liking    : int  NA 4 4 3 5 3 5 4 3 3 ...
 $ group.id  : num  1 1 1 1 1 1 1 1 1 1 ...
```

Be careful: `rbind` only works if all column names are identical in the data frames which are combined. Hence, you have to make sure that all long data frames have the same structure before applying `rbind` to them. Furthermore, you should note that performing `RR` in this last example is not overly sensible, as running a between group t-test with only two groups is rather debatable.

The function `matrix2long` essentially is a wrapper for the much more powerful functions from the `reshape` package. If you do a lot of data manipulation and conversions from wide to long format or vice versa, you definitely should dig into this package.

## 6.4 I have to run many, many round robin analyses in a huge data set. What is the most convenient way to do this?

Imagine you assessed 50 variables in round robin style, and want to extract the effects for all variables and to store them in a new data frame (e.g., for subsequent analyses). Of course, you can type the `RR` command 50 times, but there are more convenient ways to do this.

You can construct the formula by a loop, and iterate through all measured variables, and combine the results at the end. As an example, let's take the `likingLong` data set, which has 4 round robin variables:

```
> data(likingLong)
> str(likingLong)
'data.frame':      2916 obs. of  6 variables:
 $ actor.id   : int  1 2 3 4 5 6 7 8 9 10 ...
 $ partner.id : int  1 1 1 1 1 1 1 1 1 1 ...
 $ liking_a   : int  NA 4 4 3 5 3 5 4 3 3 ...
 $ liking_b   : int  NA 5 4 3 5 4 4 3 4 3 ...
 $ metaliking_a: int  NA 3 4 3 3 4 3 3 3 2 ...
 $ metaliking_b: int  NA 2 4 3 3 3 3 3 3 2 ...
```

If we want to extract the effects for all 4 variables, we could either type:

```

> RR(liking_a~actor.id*partner.id, data=likingLong)
> RR(liking_b~actor.id*partner.id, data=likingLong)
> RR(metaliking_a~actor.id*partner.id, data=likingLong)
> RR(metaliking_b~actor.id*partner.id, data=likingLong)

```

Or, we do it in a loop, store the results and combine them at the end:

```

> varnames <- colnames(likingLong)[3:6]
> # run a RR analysis for each variable and store results in a list
> res_list <- list()
> for (v in 1:length(varnames)) {
+   fl <- formula(paste(varnames[v], "~actor.id*partner.id"))
+   RR1 <- RR(fl, data=likingLong)
+   res_list <- c(res_list, list(RR1$effects))
+ }
> # now combine all effects in a single data frame; merge by id
> library(reshape)
> res <- merge_recurse(res_list, by="id")

```

As you can see, there's a new data frame with all actor and partner effects. On this data frame you can run subsequent analyses, for example correlations:

```

> str(res)
'data.frame':      54 obs. of  9 variables:
 $ id              : Factor w/ 54 levels "1","10","11",...: 1 12 23 34 45 51 52 53 54 2 ...
 $ liking_a.p      : num  -0.477 0.276 -0.324 -0.323 0.198 ...
 $ liking_a.t      : num   0.2639 -0.854 0.3611 0.4177 0.0125 ...
 $ liking_b.p      : num  -0.2283 0.2571 -0.4915 -0.0395 -0.2411 ...
 $ liking_b.t      : num   0.253 -1.021 0.305 0.257 -0.426 ...
 $ metaliking_a.p  : num  -0.2507 -0.3333 0.0338 0.0499 -0.5577 ...
 $ metaliking_a.t  : num   0.00855 -0.37037 0.08939 -0.06125 -0.2614 ...
 $ metaliking_b.p  : num  -0.0958 -0.3123 0.036 0.1303 -0.9127 ...
 $ metaliking_b.t  : num   0.0524 -0.4234 0.036 0.0377 -0.2461 ...
> round(cor(res[,2:9]), 2)
      liking_a.p liking_a.t liking_b.p liking_b.t metaliking_a.p metaliking_a.t
liking_a.p      1.00      0.11      0.85      0.14      0.47      0.19
liking_a.t      0.11      1.00      0.04      0.95      0.01      0.85
liking_b.p      0.85      0.04      1.00      0.08      0.55      0.12
liking_b.t      0.14      0.95      0.08      1.00      0.03      0.88
metaliking_a.p  0.47      0.01      0.55      0.03      1.00      0.04
metaliking_a.t  0.19      0.85      0.12      0.88      0.04      1.00
metaliking_b.p  0.43      0.03      0.63      0.07      0.90      0.08
metaliking_b.t  0.10      0.77      0.01      0.84     -0.05      0.92
      metaliking_b.p metaliking_b.t
liking_a.p           0.43           0.10
liking_a.t           0.03           0.77
liking_b.p           0.63           0.01
liking_b.t           0.07           0.84
metaliking_a.p       0.90          -0.05
metaliking_a.t       0.08           0.92
metaliking_b.p       1.00          -0.03
metaliking_b.t      -0.03           1.00

```

For convenience, this short script is also implemented in TripleR (`?getEffects`), which reduces the code to one or two lines:

```

> res <- getEffects(~actor.id*partner.id, data=likingLong,
+                  varlist=c("liking_a", "liking_b", "metaliking_a", "metaliking_b"))
[1] "Calculate: liking_a"
[1] "Calculate: liking_b"
[1] "Calculate: metaliking_a"
[1] "Calculate: metaliking_b"

```

```

> str(res)
'data.frame':      54 obs. of  9 variables:
 $ id      : Factor w/ 54 levels "1","10","11",...: 1 12 23 34 45 51 52 53 54 2 ...
 $ liking_a.p : num  -0.477 0.276 -0.324 -0.323 0.198 ...
 $ liking_a.t : num  0.2639 -0.854 0.3611 0.4177 0.0125 ...
 $ liking_b.p : num  -0.2283 0.2571 -0.4915 -0.0395 -0.2411 ...
 $ liking_b.t : num  0.253 -1.021 0.305 0.257 -0.426 ...
 $ metaliking_a.p: num  -0.2507 -0.3333 0.0338 0.0499 -0.5577 ...
 $ metaliking_a.t: num  0.00855 -0.37037 0.08939 -0.06125 -0.2614 ...
 $ metaliking_b.p: num  -0.0958 -0.3123 0.036 0.1303 -0.9127 ...
 $ metaliking_b.t: num  0.0524 -0.4234 0.036 0.0377 -0.2461 ...

```

## 6.5 An error occurs: ‘Aggregation requires fun.aggregate: length used as default’

This error most probably occurs when you specify a data set which has a multi group structure, but you forgot to define the group id in the formula (i.e., the `| group.id` part is missing).

## 6.6 My original multi group data set has X participants - the effects of the RR analysis, however, only have Y (Y < X) rows!

This happens, whenever single groups are excluded from the SRA. SRAs need a minimum group size of 4 participants. If your data set contains groups with 3 or fewer members, this group is excluded from the analyses, and no effects are calculated. A warning message informs you which groups have been excluded.

## 6.7 An example from David Kenny - Comparison with SOREMO.exe

David Kenny describes how to estimate SRMs with other software programs (<http://www.davidakenny.net/doc/srmssoftware.doc>) and also provides a data set. We can do the analysis in TripleR as well:

```

> library(TripleR)
> library(foreign)
> dat <- read.spss("http://www.davidakenny.net/doc/contribute.sav", to.data.frame=TRUE)
> RR.Kenny <- RR(l1~Actor*Partner|Group, data=dat)
> RR.Kenny
[1] "Round-Robin object ('RR'), calculated by TripleR"
[1] "Univariate analysis of one round robin variable in multiple groups"
[1] "Group descriptives: n = 24 ; average group size = 4 ; range: 4 - 4"
[1] "Univariate analyses for: l1"

```

	estimate	standardized	se	t.value	p.value
perceiver variance	0.233	0.335	0.054	4.307	0.000
target variance	0.240	0.345	0.045	5.330	0.000
relationship variance	0.222	0.320	0.030	7.316	0.000
error variance	NA	NA	NA	NA	NA
perceiver-target covariance	0.059	0.250	0.047	1.244	0.226
relationship covariance	0.014	0.063	0.034	0.414	0.682

```

[1] "Perceiver effect reliability: 0.732"
[1] "Target effect reliability: 0.738"

```

Group variance is not printed in the standard `RR`-output, but it can be accessed by:

```
> RR.Kenny$group.var  
[1] -0.09060487
```

If you compare these results with Table 1 from the `srmsoftware.doc` document, you will see that all results are identical to SOREMO.

## References

- Baayen, R., Davidson, D., & Bates, D. (2008). Mixed-effects modeling with crossed random effects for subjects and items. *Journal of Memory and Language*, 59(4), 390-412.
- Back, M., & Kenny, D. (in press). The social relations model: How to understand dyadic processes. *Social and Personality Psychology Compass*.
- Kenny, D. (1994). *Interpersonal perceptions: A social relations analysis*. New York: Guilford Press.
- Kenny, D., Kashy, D., & Cook, W. (2006). *Dyadic data analysis*. New York: Guilford.
- Kliegl, R., Masson, M. E. J., & Richter, E. M. (2010). A linear mixed model analysis of masked repetition priming. *Visual Cognition*, 18(5), 655-681.