

lavaanExtra: Convenience Functions for Package *lavaan*

Rémi Thériault ¹

¹ Department of Psychology, Université du Québec à Montréal, Québec, Canada

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

`{lavaanExtra}` is an R package that offers an alternative and vector-based syntax to the package `{lavaan}`, as well as other convenience functions such as naming paths and defining indirect effects automatically. It also offers convenience formatting optimized for publication and script sharing workflows.

Statement of need

`{lavaan}` ([Rosseel, 2012](#)) is a very popular R package for structural equation modeling (SEM). The package requires familiarizing oneself with a specific syntax to define latent variables, regressions, covariances, indirect effects, and so on.

`{lavaanExtra}` does mainly two things. First, it offers an alternative, code-efficient syntax. Second, it facilitates the analysis-to-publication workflow by providing publication-ready tables and figures following the style requirements of the American Psychological Association (APA).

Usage

There is a single function at the center of the proposed alternative syntax, `write_lavaan()`. The idea behind `write_lavaan()` is to define individual components (regressions, covariances, latent variables, etc.), provide them to the function, and have it write the `lavaan` model, so the user does not have to worry about making typos in the specific symbols required for each aspect of the model.

There are several benefits to this approach. Some `lavaan` models can become very large. By defining the entire model every time, such as is typical with `{lavaan}` users, not only do we break the DRY (Don't Repeat Yourself) principle, but our scripts can also become long and unwieldy. This problem gets worse in the scenario where we want to compare several variations of the same general model. `write_lavaan()` allows the user to reuse code components, say, only the latent variables, for future models.

This aspect also allows better control over the user's code. If the user makes a mistake in one of, say, five SEM models definition, the user will have to change it at all five places within the script. With `write_lavaan()`, the user only needs to change it once, at the appropriate location, and it will update future occurrences automatically since it relies on reusable components.

The vector-based approach also allows the use of functions to define components. For example, if all scale items are named consistently, say `x1` to `x50`, one can use `paste0("x", 1:50)` instead of typing all the items by hand and risk making mistakes.

Another issue with `lavaan` models is the readability of the code defining the model. One can go to lengths to make it pretty, but not everyone does, and many people do not use the same

38 strategies to organize the information of the model definition. With `write_lavaan()`, not
 39 only is the model information standardized, but it is also neatly divided into clear and useful
 40 categories.

41 Finally, for beginners, it can be difficult to remember the correct lavaan symbols for each
 42 specific operation. `write_lavaan()` uses intuitive names to convert the information to the
 43 correct symbols, meaning the user does not have to rely on memory as much. Even for
 44 people familiar with lavaan syntax, this approach can save time. The function also offers the
 45 possibility to define the named paths automatically with clear and intuitive names.

46 I provide a simple Confirmatory Factor Analysis (CFA) example below using the
 47 HolzingerSwineford1939 dataset (Holzinger & Swineford, 1939). The dataset con-
 48 tains the mental ability test scores of children. In this example, we want to define the latent
 49 variables `visual` (visual perception ability), `textual` (reading and writing ability), and `speed`
 50 (processing speed ability), which are defined by items 1 to 9, respectively. We can then use
 51 the `cat()` function on the resulting object (of type character) to read it in the traditional way
 52 and make sure we have not made any mistake.

```
library(lavaanExtra)

latent <- list(visual = paste0("x", 1:3),
              textual = paste0("x", 4:6),
              speed = paste0("x", 7:9))

model.cfa <- write_lavaan(latent = latent)
cat(model.cfa)

## #####
## # [-----Latent variables (measurement model)-----]
##
## visual =~ x1 + x2 + x3
## textual =~ x4 + x5 + x6
## speed =~ x7 + x8 + x9
```

59 Should we want to use these latent variables in a full SEM model, we do not need to define
 60 the latent variables again, only the new components. In the example below, I add regressions,
 61 covariances, and indirect effects to the model. Two of our latent variables (`textual` and `speed`)
 62 are now predicted by our mediating variable, `visual`. In turn, `visual` is now predicted by our
 63 independent variables, `grade` (the students' grade) and `ageyr` (the students' age, in years).

64 With the `lavaanExtra` syntax, when defining our lists of components, we can think of the
 65 `=` sign as "predicted by", a bit like `~` for regression. There is an exception to this for the
 66 indirect object, which also allows specifying our variables directly instead. When such is the
 67 case, `write_lavaan()` will define all indirect paths automatically.

```
DV <- c("textual", "speed")
M <- "visual"
IV <- c("grade", "ageyr")

mediation <- list(speed = M, textual = M, visual = IV)
regression <- list(speed = IV, textual = IV)
covariance <- list(speed = "textual", ageyr = "grade", x4 = c("x5", "x6"))
indirect <- list(IV = IV, M = M, DV = DV)

model.sem <- write_lavaan(mediation = mediation,
                          regression = regression,
                          covariance = covariance,
                          indirect = indirect,
```

```

latent = latent,
label = TRUE)

cat(model.sem)

68 ## #####
69 ## # [-----Latent variables (measurement model)-----]
70 ##
71 ## visual =~ x1 + x2 + x3
72 ## textual =~ x4 + x5 + x6
73 ## speed =~ x7 + x8 + x9
74 ##
75 ## #####
76 ## # [-----Mediations (named paths)-----]
77 ##
78 ## speed ~ visual_speed*visual
79 ## textual ~ visual_textual*visual
80 ## visual ~ grade_visual*grade + ageyr_visual*ageyr
81 ##
82 ## #####
83 ## # [-----Regressions (Direct effects)-----]
84 ##
85 ## speed ~ grade + ageyr
86 ## textual ~ grade + ageyr
87 ##
88 ## #####
89 ## # [-----Covariances-----]
90 ##
91 ## speed ~~ textual
92 ## ageyr ~~ grade
93 ## x4 ~~ x5 + x6
94 ##
95 ## #####
96 ## # [-----Mediations (indirect effects)-----]
97 ##
98 ## grade_visual_textual := grade_visual * visual_textual
99 ## grade_visual_speed := grade_visual * visual_speed
100 ## ageyr_visual_textual := ageyr_visual * visual_textual
101 ## ageyr_visual_speed := ageyr_visual * visual_speed

```

102 Tables

103 The `nice_fit()` function extracts only some of the most popular fit indices and organize them
 104 such that it is easy to compare models. There is an option to format the table as an APA
 105 {flextable} (Gohel & Skintzos, 2023), through the {rempsyc} package (Thériault, 2022), using
 106 option `nice_table = TRUE`. This flextable object can then be easily exported to Microsoft
 107 Word. Below we fit our two earlier models and feed them to `nice_fit()` as a named list:

```

library(lavaan)
fit.cfa <- cfa(model.cfa, data = HolzingerSwineford1939)
fit.sem <- sem(model.sem, data = HolzingerSwineford1939)

list.mods <- list(fit.cfa = fit.cfa, fit.sem = fit.sem)
fit_table <- nice_fit(list.mods, nice_table = TRUE)

fit_table

```

Model	χ^2	df	χ^2/df	p	CFI	TLI	RMSEA [90% CI]	SRMR	AIC	BIC
fit.cfa	85.31	24	3.55	< .001	.93	.90	.09 [.07, .11]	.06	7,517.49	7,595.34
fit.sem	114.20	34	3.36	< .001	.93	.88	.09 [.07, .11]	.06	8,640.07	8,758.59
Ideal Value ^a	—	—	< 2 or 3	> .05	≥ .95	≥ .95	< .05 [.00, .08]	≤ .08	Smaller	Smaller

^aAs proposed by Schreiber (2017).

The table can then be saved to word simply using `flextable::save_as_docx()` on the resulting `flextable` object.

```
flextable::save_as_docx(fit_table, path = "fit_table.docx")
```

It is similarly possible to prepare APA tables in Word with the regression coefficients (`lavaan_reg()`), covariances (`lavaan_cov()`), correlations (`lavaan_cor()`), or indirect effects (`lavaan_ind()`). For example, for indirect effects:

```
lavaan_ind(fit.sem, nice_table = TRUE)
```

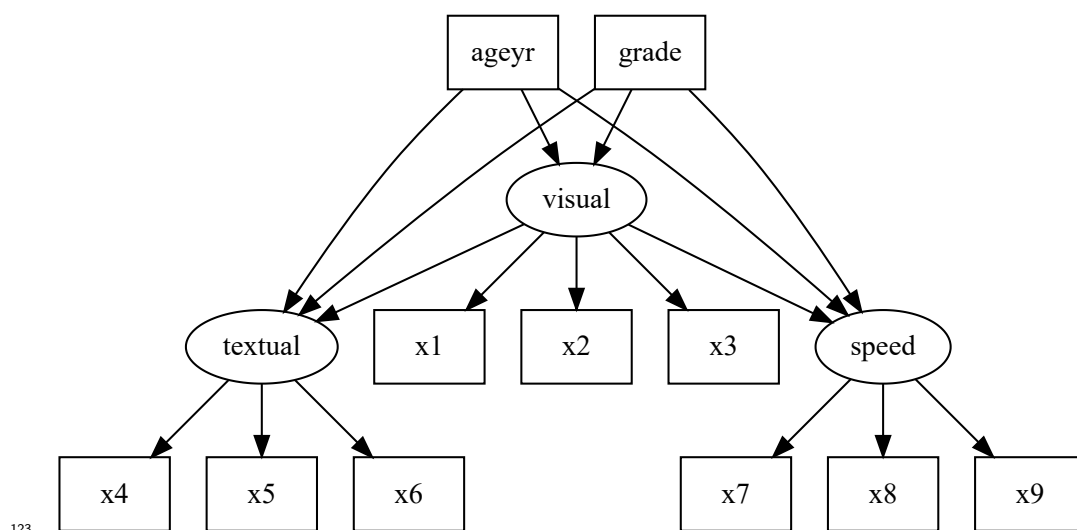
Indirect Effect	Paths	p	β	95% CI
grade → visual → textual	grade_visual*visual_textual	.001***	0.13	[0.05, 0.20]
grade → visual → speed	grade_visual*visual_speed	.001**	0.13	[0.05, 0.21]
ageyr → visual → textual	ageyr_visual*visual_textual	.012*	-0.08	[-0.14, -0.02]
ageyr → visual → speed	ageyr_visual*visual_speed	.016*	-0.08	[-0.15, -0.02]

Figures

There are several packages designed to plot SEM models, but few that people consider satisfying or sufficiently good for publication. There are two packages that stand out however, `{lavaanPlot}` (Lishinski, 2021) and `{tidySEM}` (van Lissa, 2023b). Yet, even for those excellent packages, most people do not view them as publication-ready or at least optimized in the best possible way.

This is what `nice_lavaanPlot` and `nice_tidySEM` aim to correct. Let's compare the default `lavaanPlot()` and `nice_lavaanPlot()` outputs side-by-side for demonstration purposes.

```
lavaanPlot::lavaanPlot(fit.sem)
```



`nice_lavaanPlot(fit.sem)`

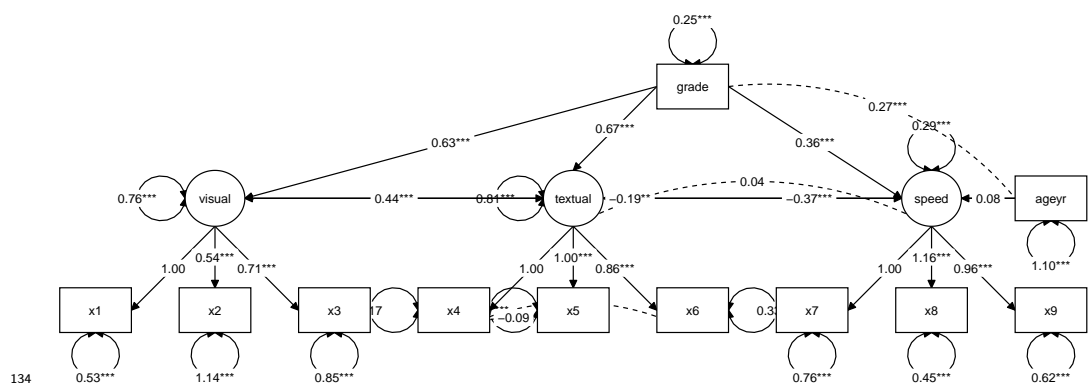


As these figures demonstrate, `nice_lavaanPlot()` has several elements frequently requested by researchers (especially in psychology): (a) a horizontal, rather than vertical, layout; (b) the coefficients appear by default (but only significant ones); (c) significance stars; and (d) the use of a sans serif font (as required by APA style for figures).

Even so, `nice_lavaanPlot` is not perfectly optimal for publication, for example for the use of curved lines, which many researchers dislike. Nonetheless, it will still yield excellent and satisfying results for a quick and easy check.

The best option for publication then is `nice_tidySEM`. Let's first look at the default output of the base `tidySEM::graph_sem()` for reference.

`tidySEM::graph_sem(fit.sem)`

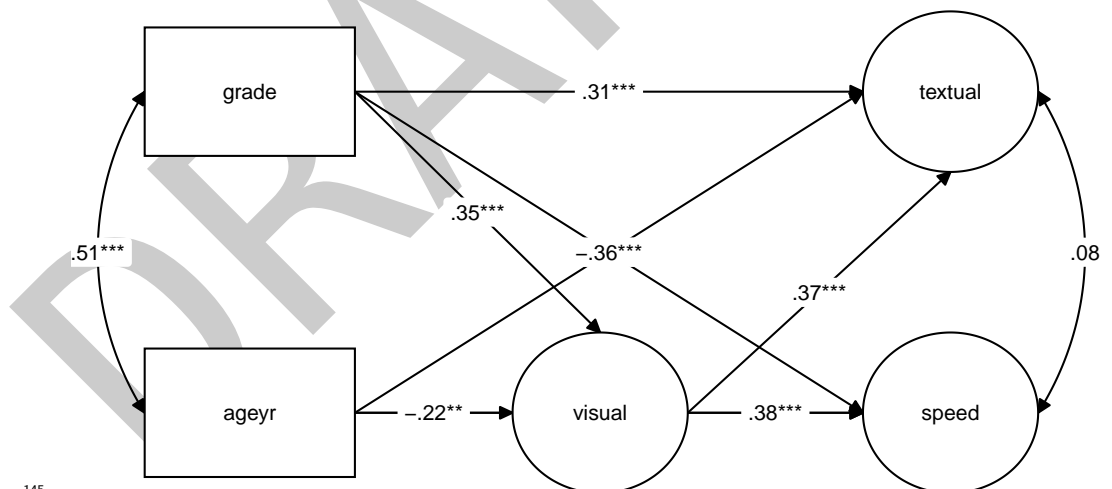


The author of the `{tidySEM}` package notes that

This uses a default layout, provided by the `igraph` package. However, the node placement is not very aesthetically pleasing. One of the areas where `tidySEM` really excels is customization. (van Lissa, 2023a)

In this sense, most of the time, both `tidySEM` and `nice_tidySEM` will need a layout in order to yield the best result. One of the benefits of `nice_tidySEM` is that when our model is simply made of three “levels”: independent variables, mediators, and dependent variables (e.g., for a path analysis, or if we do not want to draw the items for a full SEM), it is possible to automatically specify a proper layout by simply feeding it the indirect object that we created earlier.

```
nice_tidySEM(fit.sem, layout = indirect)
```



For the time being, `nice_tidySEM` only supports this three-level automatic layout, but designs with more levels are in the works. In the meantime, when the model is more complex (or that we want to include items), it is necessary to specify the layout manually using a matrix or data frame, which allows fine-grained control over the generated figure.

```
mylayout <- data.frame(
  IV = c("x1", "grade", "", "ageyr", ""),
  M = c("x2", "", "visual", "", ""),
  DV = c("x3", "textual", "", "speed", "x9"),
  DV.items = c(paste0("x", 4:8)))
as.matrix(mylayout)
```

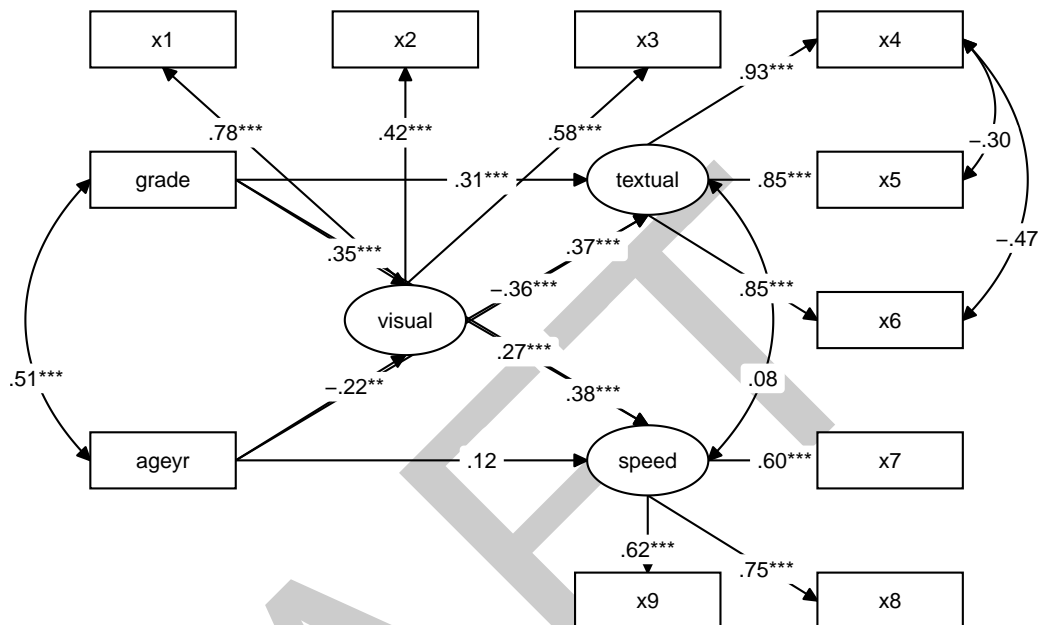
```
##      IV      M      DV      DV.items
## [1,] "x1"    "x2"    "x3"    "x4"
## [2,] "grade" ""      "textual" "x5"
```

```

153 ## [3,] ""      "visual" ""      "x6"
154 ## [4,] "ageyr" ""      "speed" "x7"
155 ## [5,] ""      ""      "x9"    "x8"

nice_tidySEM(fit.sem, layout = mylayout, label_location = 0.70)

```



```

156
157 If the figure is still not sufficiently satisfying, it is possible to store the output as a tidy_sem
158 object (by using plot = FALSE), which can then be modified according to regular tidySEM
159 syntax. This can be useful to fine-tune and finalize the figure.

```

```

x <- nice_tidySEM(fit.sem, layout = mylayout, label_location = 0.65,
  reduce_items = c(x = 0.4, y = 0.2), plot = FALSE)
from <- x$edges$from
to <- x$edges$to
x$edges[from == "grade" & to == "speed", "curvature"] <- 40
x$edges[from == "ageyr" & to == "textual", "curvature"] <- -40
plot(x)

```



160

161 The resulting figure can be saved using `ggplot2::ggsave()` (Wickham, 2016):

```
ggplot2::ggsave("my_semPlot.pdf", width = 8, height = 6)
```

162 Other differences between `{tidySEM}` and `nice_tidySEM()` are that: (a) the latter displays
163 standardized coefficients by default (but unstandardized coefficients can be specified with
164 `est_std = FALSE`), (b) if using standardized coefficients, the leading zero is omitted (as per
165 APA requirements); (c) does not plot the variances by default, (d) uses full double-headed
166 arrows instead of dashed lines with no arrows for covariances, (e) has further arguments for easy
167 customization (e.g., `reduce_items`), and (f) allows defining an automatic layout in specific
168 cases (as described earlier).

169 Finally, the base function, `tidySEM::graph_sem()`, is difficult to customize in depth. For
170 the aesthetics of `nice_tidySEM()`, for example, we need to rely instead on the `{tidySEM}`'s
171 `prepare_graph()`, `edit_graph()`, and numerous conditional formatting functions. In contrast
172 to `nice_tidySEM()`, these `tidySEM` functions act more like a grammar of SEM plotting, akin to
173 the popular grammar of graphics, `{ggplot2}` (Wickham, 2016). This provides great flexibility,
174 but for the occasional user, also comes with an additional burden, as users may for example
175 need to skim through almost 400 undocumented functions, should they want to conditionally
176 edit the resulting `tidy_sem` object.

177 Availability

178 The `{lavaanExtra}` package is licensed under the MIT License. It is available on CRAN, and
179 can be installed using `install.packages("lavaanExtra")`. The full tutorial website can be
180 accessed at: <https://lavaanExtra.remi-theriault.com/>. All code is open-source and hosted on
181 GitHub, and bugs can be reported at <https://github.com/rempsyc/lavaanExtra/issues/>.

182 Acknowledgements

I would like to thank Hugues Leduc, Jany St-Cyr, Andreea Gavrila, Patrick Coulombe, Jay Olson, Charles-Étienne Lavoie, and Björn Büdenbender for statistical or technical advice that helped inform some functions of this package and/or useful feedback on this manuscript. I would also like to acknowledge funding from the Social Sciences and Humanities Research Council of Canada.

188 References

- 189 Gohel, D., & Skintzou, P. (2023). *flextable: Functions for tabular reporting*. <https://CRAN.R-project.org/package=flextable>
- 190
- 191 Holzinger, K. J., & Swineford, F. (1939). A study in factor analysis: The stability of a bi-factor solution. *Supplementary Educational Monographs*.
- 192
- 193 Lishinski, A. (2021). *lavaanPlot: Path diagrams for 'lavaan' models via 'DiagrammeR'*.
194 <https://CRAN.R-project.org/package=lavaanPlot>
- 195 Rosseel, Y. (2012). lavaan: An R package for structural equation modeling. *Journal of Statistical Software*, 48(2), 1–36. <https://doi.org/10.18637/jss.v048.i02>
- 196
- 197 Thériault, R. (2022). *rempsyc: Convenience functions for psychology*. <https://rempsyc.remi-theriault.com>
- 198
- 199 van Lissa, C. J. (2023a). *Plotting graphs for structural equation models*. https://cjvanlissa.github.io/tidySEM/articles/Plotting_graphs.html
- 200
- 201 van Lissa, C. J. (2023b). *tidySEM: Tidy structural equation modeling*. <https://CRAN.R-project.org/package=tidySEM>
- 202
- 203 Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York.
204 <https://ggplot2.tidyverse.org>