The Journal of Open Source Software

lavaanExtra: Convenience Functions for Package *lavaan*

Rémi Thériault [1]

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

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

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

Statement of need

{lavaan} (Rosseel, 2012) is a very popular R package for structural equation modeling (SEM). {lavaan} 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.

Alternative Syntax

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, 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 relevant 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 readability of the code defining the model. One can go in lengths to make it pretty, but not everyone does, and the model formatting is certainly not

standardized. With `write_lavaan()`, not only is the model standardized, but it is also neatly divided in clear and useful categories.

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

I provide a simple CFA example below, where the latent variables `visual`, `textual`, and `speed` are defined by items 1 to 9. We can then use the `cat()` function on the resulting object (of type character) to read it in the traditional way 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
```

Should we want to use these latent variables in a full SEM model, we do not need to define the latent variables again, only the new components. In the example below, the dependent variables DV (`speed` and `textual`) are mediated by the mediator M (`visual`) and predicted by the independent variables IV (`ageyr` and `grade`). Similarly, we specify covariances between the DVs and IVs, and in this case our indirect effects can be determined automatically.

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

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

model.sem <- write_lavaan(mediation, regression, covariance,
                          indirect, latent, label = TRUE)
cat(model.sem)
```

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

```
## speed ~ visual_speed*visual
## textual ~ visual_textual*visual
## visual ~ ageyr_visual*ageyr + grade_visual*grade
##
## ################################################
## # [---------Regressions (Direct effects)---------]
##
## speed ~ ageyr + grade
## textual ~ ageyr + grade
##
## ################################################
## # [-----------------Covariances-----------------]
##
## speed ~~ textual
## ageyr ~~ grade
##
## ################################################
## # [--------Mediations (indirect effects)---------]
##
## ageyr_visual_speed := ageyr_visual * visual_speed
## ageyr_visual_textual := ageyr_visual * visual_textual
## grade_visual_speed := grade_visual * visual_speed
## grade_visual_textual := grade_visual * visual_textual
```

Tables

The most popular {lavaanExtra} function for tables is `nice_fit()`, which extracts only some of the most popular fit indices, compares them among models automatically, and formats the output as an APA-style {flextable} (Gohel & Skintzos, 2023), through the {rempsyc} package (Thériault, 2022). 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)

fit_table <- nice_fit(dplyr::lst(fit.cfa, fit.sem), nice_table = TRUE)

fit_table
```

Model	χ^2	*df*	χ^2/df	*p*	CFI	TLI	RMSEA	SRMR	AIC	BIC
fit.cfa	85.31	24	3.55	< .001***	0.93	0.90	0.09	0.06	7,517.49	7,595.34
fit.sem	116.26	36	3.23	< .001***	0.93	0.89	0.09	0.06	8,638.13	8,749.25
Ideal Value[a]	—	—	**< 2 or 3**	**> .05**	**≥ .95**	**≥ .95**	**< .06-.08**	**≤ .08**	**Smaller is better**	**Smaller is better**

[a]As proposed by Schreiber et al. (2006).

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()`), or indirect effects (`lavaan_ind()`). For example, for indirect effects:

```
x <- lavaan_ind(fit.sem, nice_table = TRUE)
flextable::save_as_docx(x, path = "ind_table.docx")
```

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

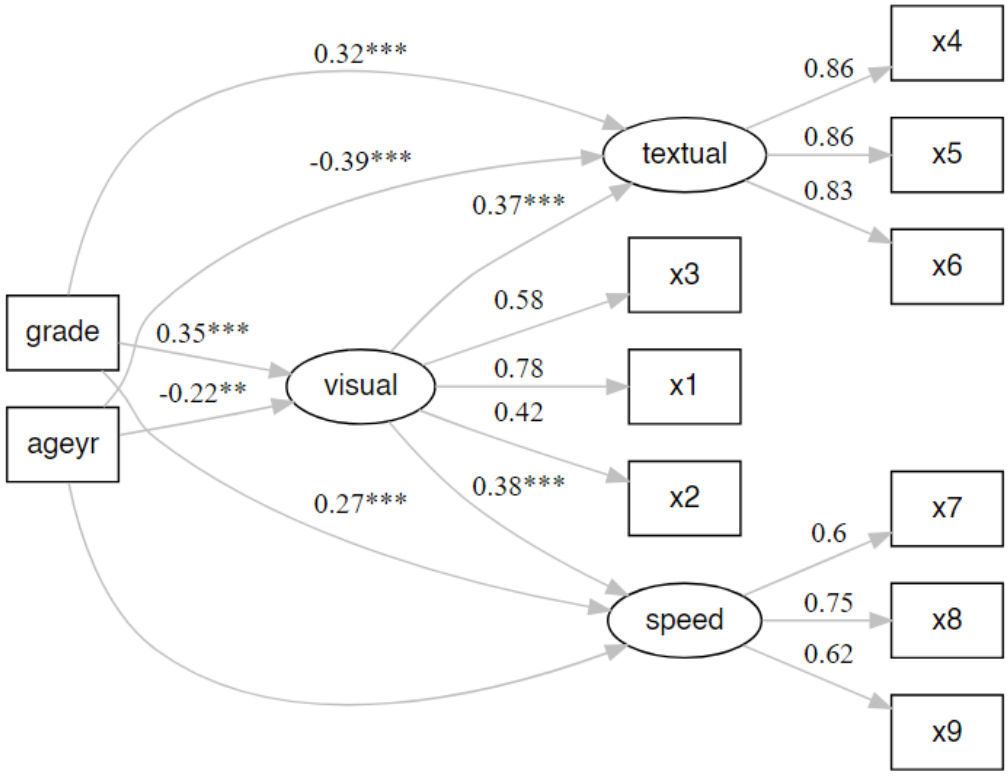
Indirect Effect	Paths	β	*p*
ageyr → visual → speed	ageyr_visual*visual_speed	-0.08	.020
ageyr → visual → textual	ageyr_visual*visual_textual	-0.08	.015
grade → visual → speed	grade_visual*visual_speed	0.13	.002
grade → visual → textual	grade_visual*visual_textual	0.13	.001

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, 2023). Yet, even for those excellent packages, most people do not view them as publication-ready or at least optimized in the best possible way.

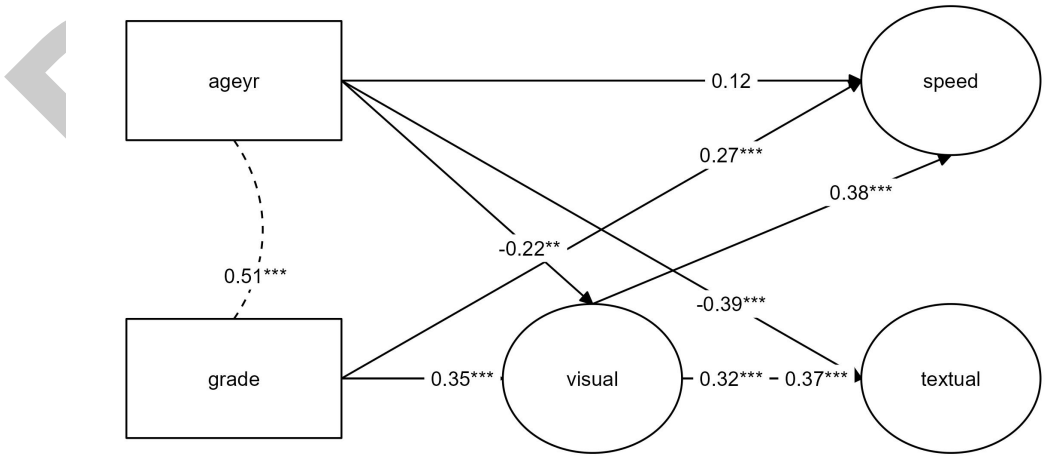
This is what `nice_lavaanPlot` and `nice_tidySEM` aim to correct. `nice_lavaanPlot` is not optimal for publications but will yield excellent results for a quick and easy check.

```
nice_lavaanPlot(fit.sem)
```



The best option for publication is `nice_tidySEM`. When our model is simply made of three "levels": independent variables, mediators, and dependent variables, or that we do not want to draw the items, we can specify the layout by simply feeding it the `indirect` object that we created earlier.

```
nice_tidySEM(fit.sem, layout = indirect, label_location = 0.75)
```



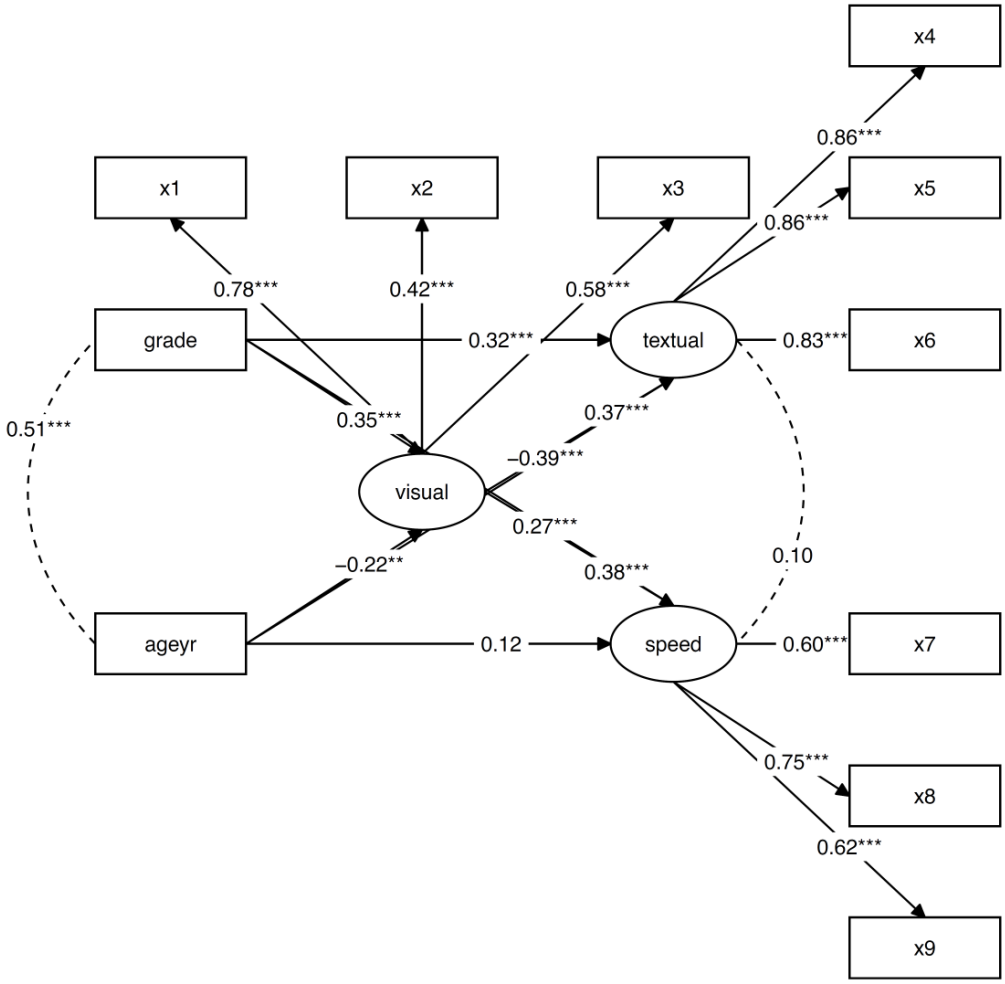
However, when the model is more complex (or that we want to include items), it is necessary to specify the layout manually using a matrix or data frame, which allows a fine-grained control over the generated figure.

```
mylayout <- data.frame(
  IV = c("", "x1", "grade", "", "ageyr", "", ""),
  M = c("", "x2", "", "visual", "", "", ""),
```

```
  DV = c("", "x3", "textual", "", "speed", "", ""),
  DV.items = c(paste0("x", 4:6), "", paste0("x", 7:9)))
as.matrix(mylayout)
```

```
##      IV      M        DV        DV.items
## [1,] ""      ""       ""        "x4"
## [2,] "x1"    "x2"     "x3"      "x5"
## [3,] "grade" ""       "textual" "x6"
## [4,] ""      "visual" ""        ""
## [5,] "ageyr" ""       "speed"   "x7"
## [6,] ""      ""       ""        "x8"
## [7,] ""      ""       ""        "x9"
```

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



This figure can be saved using `ggplot2::ggsave()` (Wickham, 2016).

```
ggplot2::ggsave("my_semPlot.pdf", width = 7, height = 4)
```

Availability

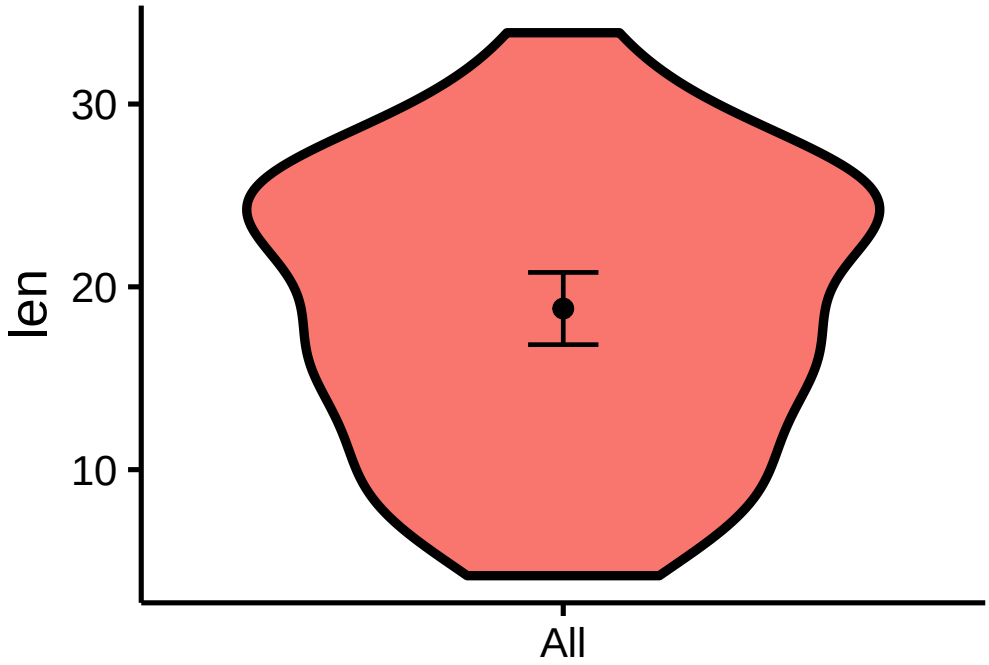
The {lavaanExtra} package is licensed under the MIT License. It is available on CRAN, and can be installed using `install.packages("lavaanExtra")`. The full tutorial website can be

accessed at: https://lavaanExtra.remi-theriault.com/. All code is open-source and hosted on GitHub, and bugs can be reported at https://github.com/rempsyc/lavaanExtra/issues/.

Acknowledgements

I would like to thank Hugues Leduc, Charles-Étienne Lavoie, Jany St-Cyr, and Andreea Gavrila 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.

Testing EPS vector graphics



References

Gohel, D., & Skintzos, P. (2023). *flextable: Functions for tabular reporting*. https://CRAN.R-project.org/package=flextable

Lishinski, A. (2021). *lavaanPlot: Path diagrams for 'lavaan' models via 'DiagrammeR'*. https://CRAN.R-project.org/package=lavaanPlot

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

Thériault, R. (2022). *rempsyc: Convenience functions for psychology*. https://rempsyc.remi-theriault.com

van Lissa, C. J. (2023). *tidySEM: Tidy structural equation modeling*. https://CRAN.R-project.org/package=tidySEM

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org