

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Published: unpublished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, vector-based syntax to 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). `{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 (following the style of the American Psychological Association, APA).

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, 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 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 many people do not use the same

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

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

45 I provide a simple Confirmatory Factor Analysis (CFA) example below, where the latent variables
46 visual, textual, and speed are defined by items 1 to 9. We can then use the `cat()` function
47 on the resulting object (of type character) to read it in the traditional way and make sure we
48 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)

49 ## #####
50 ## # [-----Latent variables (measurement model)-----]
51 ##
52 ## visual =~ x1 + x2 + x3
53 ## textual =~ x4 + x5 + x6
54 ## speed =~ x7 + x8 + x9

55 Should we want to use these latent variables in a full SEM model, we do not need to define
56 the latent variables again, only the new components. With the lavaanExtra syntax, when
57 defining our lists of components, we can think of the = sign as “predicted by”, a bit like ~ for
58 regression. There is an exception to this for the indirect object, which also allows specifying
59 our variables directly instead. When such is the case, write_lvaan() will define all indirect
60 paths 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", x4 = c("x5", "x6"))
indirect <- list(IV = IV, M = M, DV = DV)

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

61 ## #####
62 ## # [-----Latent variables (measurement model)-----]
63 ##
64 ## visual =~ x1 + x2 + x3
65 ## textual =~ x4 + x5 + x6
66 ## speed =~ x7 + x8 + x9
67 ##
68 ## #####
```

```

69 ## # [-----Mediations (named paths)-----]
70 ##
71 ## speed ~ visual_speed*visual
72 ## textual ~ visual_textual*visual
73 ## visual ~ ageyr_visual*ageyr + grade_visual*grade
74 ##
75 ## #####
76 ## # [-----Regressions (Direct effects)-----]
77 ##
78 ## speed ~ ageyr + grade
79 ## textual ~ ageyr + grade
80 ##
81 ## #####
82 ## # [-----Covariances-----]
83 ##
84 ## speed ~~ textual
85 ## ageyr ~~ grade
86 ## x4 ~~ x5 + x6
87 ##
88 ## #####
89 ## # [-----Mediations (indirect effects)-----]
90 ##
91 ## ageyr_visual_speed := ageyr_visual * visual_speed
92 ## ageyr_visual_textual := ageyr_visual * visual_textual
93 ## grade_visual_speed := grade_visual * visual_speed
94 ## 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 and organize them such that it is easy to compare models. There is an option to format the table as an APA {flextable} (Gohel & Skintzos, 2023), through the {rempsyc} package (Thériault, 2022), using option `nice_table = TRUE`. This flextable object can then be easily exported to Microsoft 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)

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	90% CI (RMSEA)	SRMR	AIC	BIC
fit.cfa	85.31	24	3.55	<.001***	.93	.90	.09	[0.07, 0.11]	.06	7,517.49	7,595.34
fit.sem	114.20	34	3.36	<.001***	.93	.88	.09	[0.07, 0.11]	.06	8,640.07	8,758.59
Ideal Value^a	—	—	< 2 or 3	> .05	≥ .95	≥ .95	< .06-.08	[.00, .10]	≤ .08	Smaller is better	Smaller is better

^aAs 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()), correlations (lavaan_cor()), or indirect effects
(lavaan_ind()). For example, for indirect effects:
lavaan_ind(fit.sem, nice_table = TRUE)
```

Table with 5 columns: Indirect Effect, Paths, p, β, 95% CI. It lists indirect effects such as ageyr → visual → speed and grade → visual → speed with their corresponding path coefficients and confidence intervals.

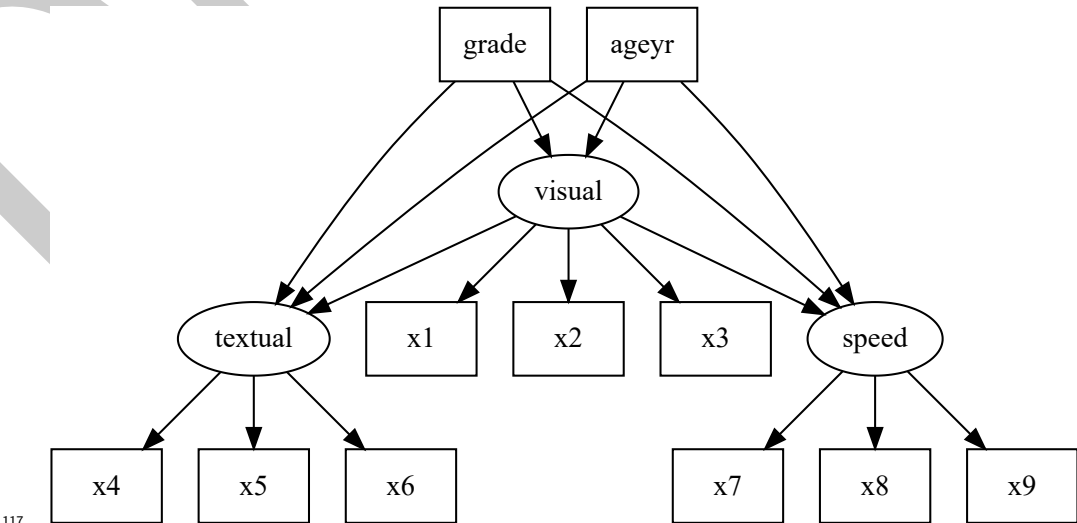
108

109 Figures

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

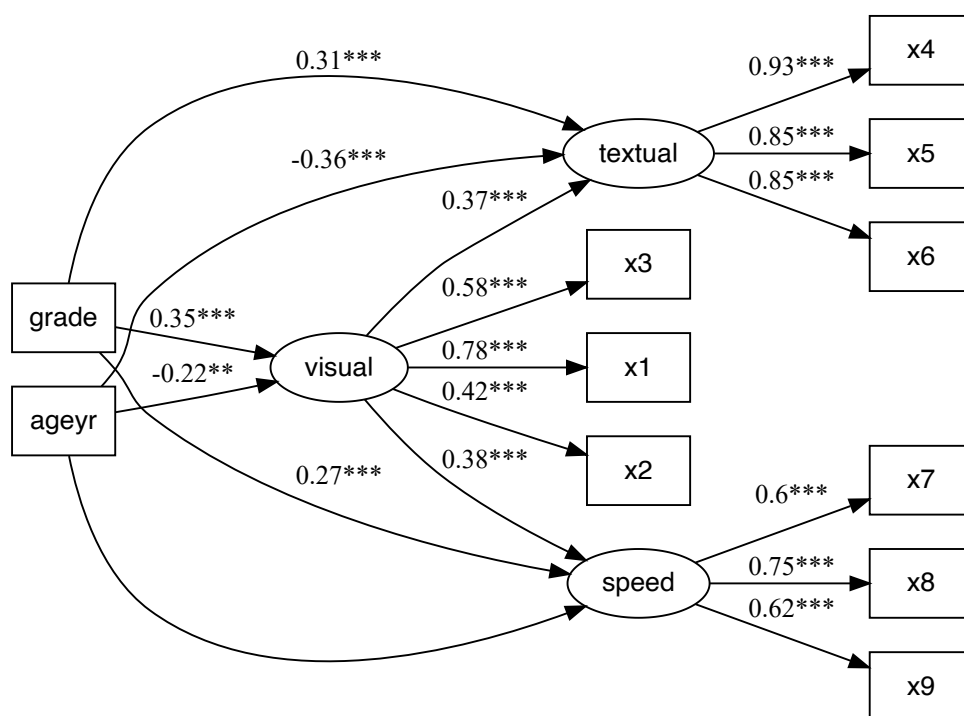
115 This is what nice_lavaanPlot and nice_tidySEM aim to correct. Let's compare the default
116 lavaanPlot() and nice_lavaanPlot() outputs side-by-side for demonstration purposes.

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



117

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



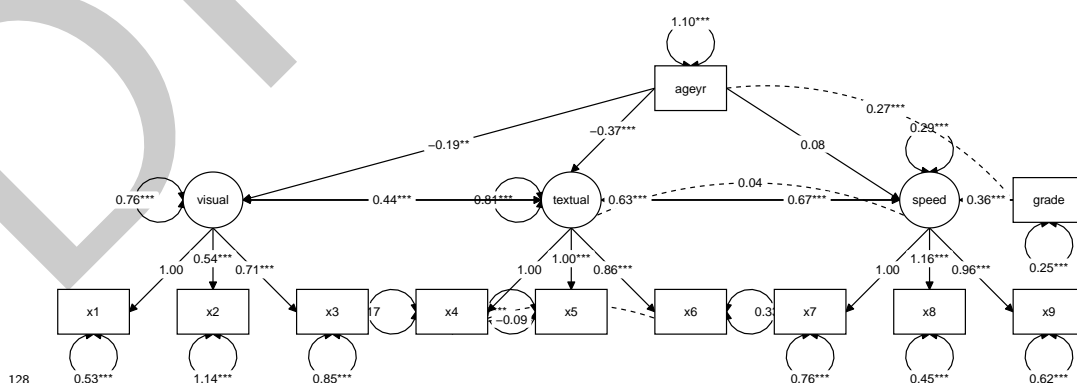
118

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

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

126 In turn, the best option for publication is `nice_tidySEM`. Let's first look at the default output
 127 of the base `tidySEM::graph_sem()` for reference.

`tidySEM::graph_sem(fit.sem)`



128

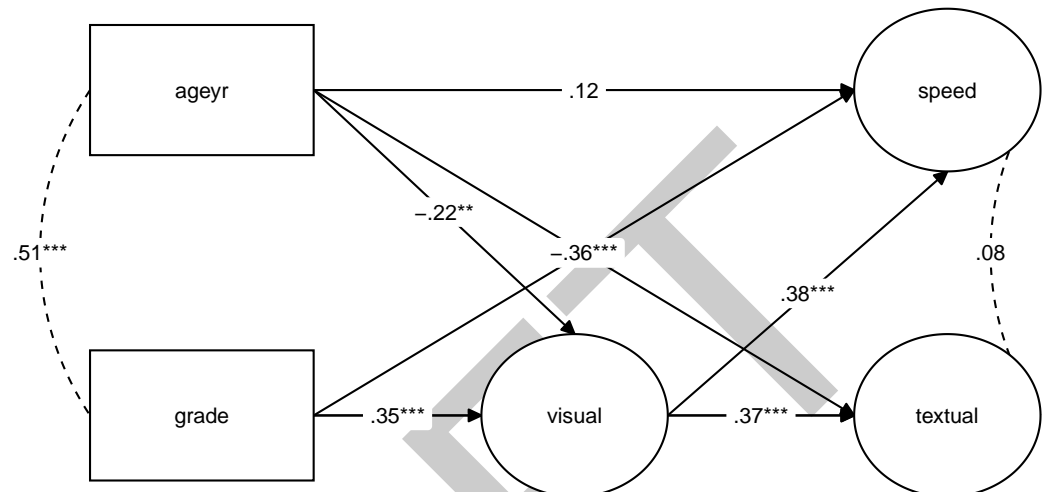
129 The author of the `{tidySEM}` package notes that

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

133 In this sense, most of the time, both `tidySEM` and `nice_tidySEM` will need a layout in order to
 134 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)
```



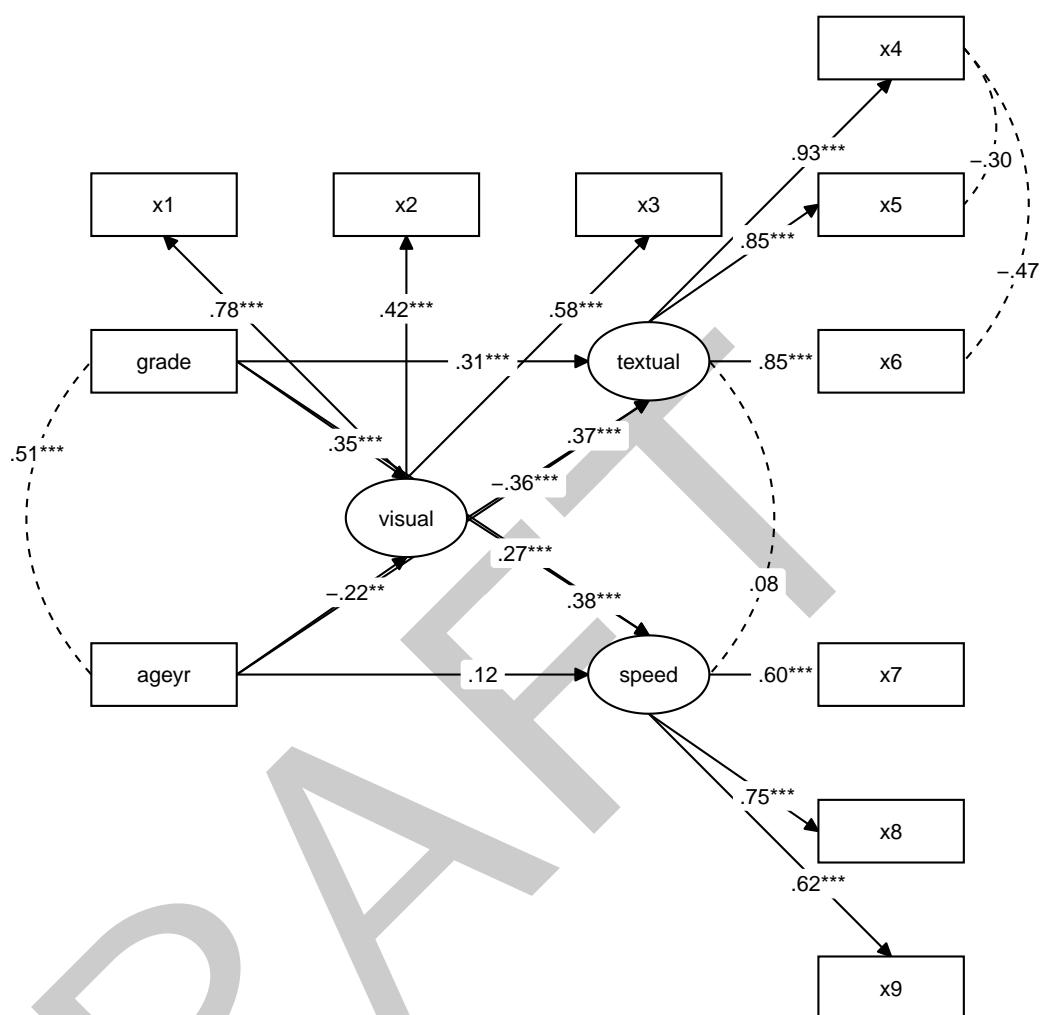
139

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 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)))
# mylayout <- data.frame(
#   IV = c("x1", "grade", "", "ageyr", ""),
#   M = c("x2", "", "visual", "", "x7"),
#   DV = c("x3", "textual", "", "speed", "x8"),
#   DV.items = c(paste0("x", 4:6), "", "x9"))
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"
```

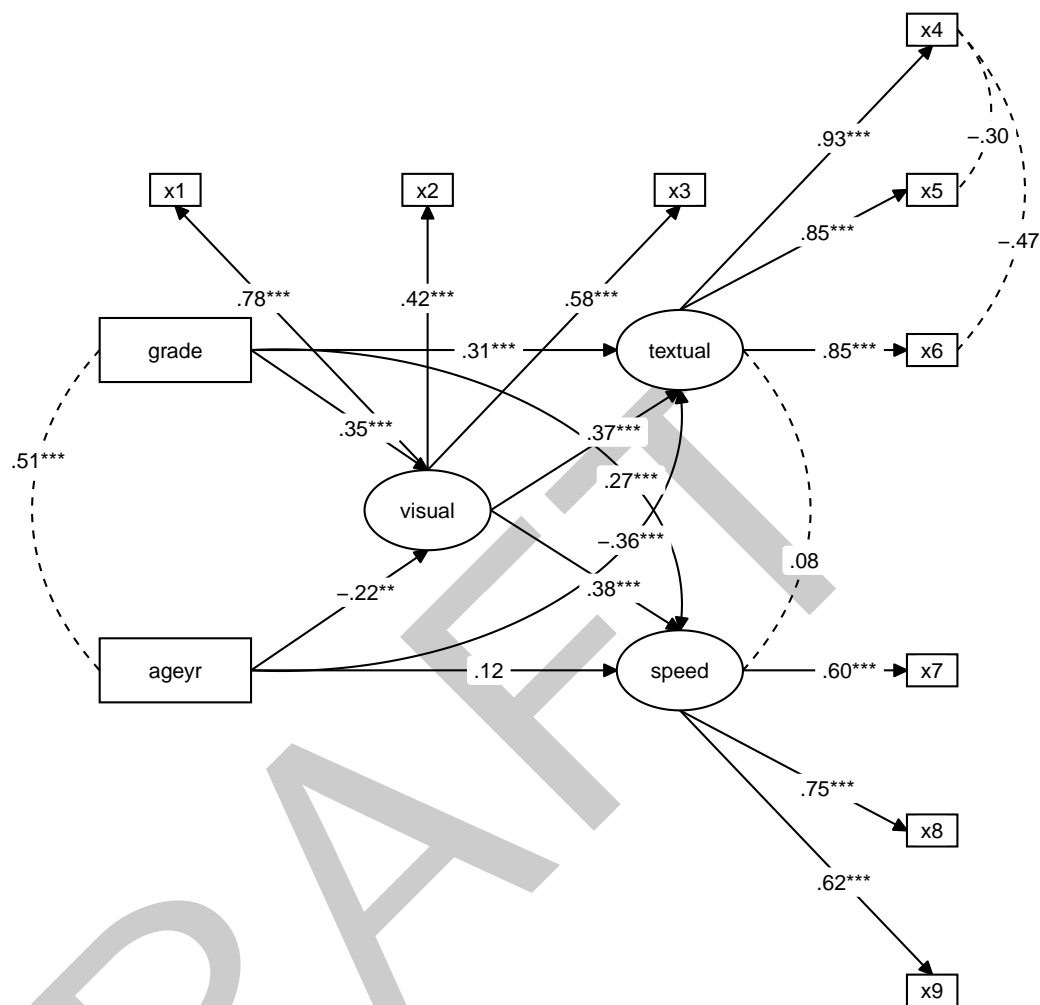
```
nice_tidySEM(fit.sem, layout = mylayout, label_location = 0.70)
```



152

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

```
x <- nice_tidySEM(fit.sem, layout = mylayout, label_location = 0.65,
  plot = FALSE)
x$edges[x$edges$from == "grade" & x$edges$to == "speed",
  "curvature"] <- 40
x$edges[x$edges$from == "ageyr" & x$edges$to == "textual",
  "curvature"] <- -40
x$edges$linetype <- ifelse(x$edges$arrow == "none", 2, 1)
x$nodes[6:14,]$node_xmin <- x$nodes[6:14,]$node_xmin + 0.4
x$nodes[6:14,]$node_xmax <- x$nodes[6:14,]$node_xmax - 0.4
x$nodes[6:14,]$node_ymin <- x$nodes[6:14,]$node_ymin + 0.2
x$nodes[6:14,]$node_ymax <- x$nodes[6:14,]$node_ymax - 0.2
plot(x)
```



156

157 In any case, the resulting figure can be saved using `ggplot2::ggsave()`.

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

158 Other differences between `{tidySEM}` and `nice_tidySEM()` are that: (a) the latter displays
159 standardized coefficients by default (but unstandardized coefficients can be specified with
160 `est_std = FALSE`), (b) if using standardized coefficients, the leading zero is omitted (as per
161 APA requirements); (c) does not plot the variances per default, and (d) allows defining an
162 automatic layout in specific cases (as described earlier).

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

171 Availability

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

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

176 Acknowledgements

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

181 References

- 182 Gohel, D., & Skintzos, P. (2023). *flextable: Functions for tabular reporting*. <https://CRAN.R-project.org/package=flextable>
- 183
- 184 Lishinski, A. (2021). *lavaanPlot: Path diagrams for 'lavaan' models via 'DiagrammeR'*.
185 <https://CRAN.R-project.org/package=lavaanPlot>
- 186 Rosseel, Y. (2012). lavaan: An R package for structural equation modeling. *Journal of*
187 *Statistical Software*, 48(2), 1–36. <https://doi.org/10.18637/jss.v048.i02>
- 188 Thériault, R. (2022). *rempsyc: Convenience functions for psychology*. <https://rempsyc.remi-theriault.com>
- 189
- 190 van Lissa, C. J. (2023a). *Plotting graphs for structural equation models*. https://cjvanlissa.github.io/tidySEM/articles/Plotting_graphs.html
- 191
- 192 van Lissa, C. J. (2023b). *tidySEM: Tidy structural equation modeling*. <https://CRAN.R-project.org/package=tidySEM>
- 193
- 194 Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York.
195 <https://ggplot2.tidyverse.org>