

Latent Class Analysis

With Open-Source Software in R

Caspar J. Van Lissa

2026-08-29

Table of contents

General Information	5
Relevance	5
Requirements	5
Content	6
Thursday September 17th, 14:00 to 18:00	6
Friday September 18th, 9:00 to 13:00	6
Practicalities	6
Course materials	6
Staff	6
Software	7
1 Introduction to Latent Class Analysis	8
1.1 Pre-Analysis	8
1.1.1 Using Your Own Data	9
1.2 Examining Observed Data	9
1.3 Data Preprocessing	12
1.4 Missing Data	16
1.5 Model Specification	16
1.6 Latent Profile Analysis (LPA) with Continuous Indicators	17
1.6.1 Model Specification	18
1.6.2 Maximum Number of Classes	19
1.6.3 Class Enumeration	20
1.6.4 Eliminating Models from Consideration	21
1.6.5 Reporting Results	22
1.7 Latent Class Analysis with Ordinal indicators	25
1.8 Dichotomous indicators: <code>mx_lca()</code>	31
1.8.1 Model Fitting	32
1.9 Mixed indicator types: <code>mx_mixed_lca()</code>	34
1.10 Classification Accuracy Deep Dive	37
1.11 Reporting Results	40
2 Confirmatory Latent Class Analysis	42
2.1 Simulate example data	42
2.2 Test the number of classes	43
2.3 Translate theoretical assumptions into parameter constraints	45

3	Caregiver Compass	49
3.1	Loading the Data	49
3.2	Descriptive statistics	50
3.3	Conducting Latent Profile Analysis	52
3.4	Class Enumeration	53
3.4.1	Using ICs	54
3.4.2	Using LMR tests	54
3.4.3	Using BLRT tests	55
3.5	Comparing Competing Theoretical Model Specifications	56
3.6	Interpreting the Final Class Solution	57
4	Auxiliary Variable Analyses	60
4.1	Why auxiliary-variable analyses require special care	60
4.2	Example Data and Model	61
4.3	Inspect Classification Uncertainty	63
4.4	The BCH method	64
4.4.1	Comparing class means on a continuous outcome	64
4.4.2	Comparing Proportions Across Classes	66
4.4.3	Comparing Auxiliary Models Across Classes	67
4.5	Pseudo-class analysis	69
4.5.1	Using OpenMx Models	70
4.6	Generating pseudo-class draws explicitly	71
4.7	BCH or Pseudo-class?	72
4.8	Predicting class membership	72
4.9	Forecasting Class for New Cases	75
5	Latent Class Growth Analysis	76
5.1	Data preprocessing	76
6	Latent Class Growth Analysis	80
6.1	Class enumeration	81
6.1.1	Scree plot	82
6.2	Wald tests	84
6.3	Trajectory plot	85
7	Specifying Custom Models	87
7.1	Three ways to specify a mixture model	88
7.2	Using <code>lavaan</code> Syntax with <code>{C}</code>	88
7.3	Example 1: A latent class regression model	89
7.4	Specifying Class-specific Models	91
7.5	Modeling Zero Inflation	92
7.5.1	Simulating the data	92
7.5.2	Specifying the Models	93

7.5.3	Why not simply use <code>mx_profiles()</code> ?	95
7.6	Using OpenMx Models	96
7.7	Building models without immediately estimating them	97
7.8	Conclusion	99
References		100
Appendices		101
A Setup your Computer		101

General Information

Course name	Latent Class Analysis with Open-Source Software in R
Instruction language	English
Type of instruction	Lectures and lab sessions
Level	Graduate School of Psychology, University of Basel
Course load	8 hrs
Lectures	Dr. C. J. Van Lissa

Are there meaningful subgroups in your data that conventional analyses miss? Latent class analysis (LCA) can identify subgroups in your data that differ qualitatively in their response patterns. You may know it by other names, such as mixture modeling, latent class growth analysis, or hidden Markov modeling.

Relevance

This workshop is designed for PhD students who want to use person-centered methods to answer substantive research questions about unobserved groups. For example, you can explore whether your population is homogeneous or heterogeneous; test theories about categorical latent variables, like identity status theory or developmental stage theory; estimate people's unknown group membership - like whether they are at risk for a clinical disorder, are in a filter bubble, or have become radicalized. Group membership can be based on simple descriptive statistics (e.g., means on questionnaire items), or on more complex models (e.g., developmental trajectory over time).

Requirements

The workshop is suitable for a wide range of experience levels: complete beginners in R, researchers who already use R but are new to LCA, and advanced users who want to transition to open-source software or discuss current best practices.

Content

Thursday September 17th, 14:00 to 18:00

You will learn the core ideas behind LCA and get hands-on experience with user-friendly R functions for estimating latent class models with continuous, ordinal, and dichotomous indicators, and combinations of these. We will focus on the practical decisions researchers face: evaluating model fit, assessing classification accuracy, interpreting model parameters, and reporting results clearly with publication-ready tables and figures.

Friday September 18th, 9:00 to 13:00

Several advanced topics are introduced, and students choose two to focus on:

- Confirmatory LCA: testing hypotheses about the number of classes
- Auxiliary variables: examining differences between classes on external variables
- Latent class growth analysis
- Custom model specification for LCA

Practicalities

You will need to complete the setup tutorial before the workshop, so we spend our time together productively: Appendix [A](#). All exercises are supported by tutorial vignettes and fully reproducible R code that you can adapt to your own research. You are encouraged to bring your own data, so you can directly apply what you learn to your dissertation or ongoing projects. Exercises are completed in pairs, creating space for discussion, feedback, and shared problem-solving.

Course materials

You do not need a book for this course. This GitBook provides all relevant course materials, along with primary literature linked in the GitBook.

Staff

[dr. Caspar J. Van Lissa](#)

Software

The official software for this course is R, which you should install on your laptop as explained in Appendix [A](#).

1 Introduction to Latent Class Analysis

This vignette covers the core ideas behind LCA, and takes you through the estimation of models with continuous, ordinal, and dichotomous indicators, and combinations of these. We focus on the practical decisions researchers face: evaluating model fit, assessing classification accuracy, interpreting model parameters, and reporting results clearly with publication-ready tables and figures.

To follow along and run the code yourself, open the file `lca_introduction.qmd` in the `lca_course` project, as explained in Appendix [A](#).

After this vignette, you should be able to:

- Prepare a dataset for LCA
- fit latent profile models with continuous indicators using `mx_profiles()`;
- fit latent class models with ordinal or dichotomous indicators using `mx_lca()`;
- fit models that combine indicator types using `mx_mixed_lca()`;
- compare candidate class solutions using `table_fit()`;
- assess classification uncertainty using `class_prob()`;
- interpret class-specific parameters using `table_results()`;
- produce reporting-quality tables and figures

Note that this vignette roughly follows the steps outlines in the SMART-LCA Checklist (Standards for More Accuracy in Reporting of different Types of Latent Class Analysis), but not in detail. You can examine the checklist by running `vignette("SMART_LCA_checklist", package = "tidySEM")`.

1.1 Pre-Analysis

In this case, we're LCAs as toy examples to become familiar with the technique. This application of LCA is purely **exploratory** (there is no theory about the number and kind of classes). We are unsure if the indicators are valid and reliable for LCA, and our sample size is predetermined by the data available for secondary analysis. We will use data from the following paper:

Zhao K, Zhao Y, Xu W. Relationship between middle school students' academic stress and physical exercise behavior from the perspective of Self-Determination Theory: The chained mediation of motivation and intention. PLoS One. 2025 Jan 3;20(1):e0316599. doi: 10.1371/journal.pone.0316599. PMID: 39752352; PMCID: PMC11698333.

These data are available in the file [pone.0316599.s002.xlsx](#), and we can load them as follows:

```
library(tidySEM)
library(OpenMx)
library(ggplot2)
library(readxl)
set.seed(2026)
df <- readxl::read_xlsx("pone.0316599.s002.xlsx", 1)
```

Run `View(df)` to inspect the data.

1.1.1 Using Your Own Data

If you are using your own data, you might be able to load them by using or modifying one of the following functions:

```
# For CSV data:
df <- read.csv("your_file.csv")
# For Excel spreadsheets:
df <- readxl::read_excel("your_file.xlsx", sheet = 1)
# For SPSS data:
df <- foreign::read.spss("your_file.sav", to.data.frame = TRUE)
# For Stata data:
df <- foreign::read.dta("your_file.dta")
```

1.2 Examining Observed Data

Scroll through the dataset; note that the bottom rows are all empty. We can drop them as follows:

```
# Drop empty rows (they have missing ID numbers).
df <- df[!is.na(df[[1]]), ]
# Drop ID number
df <- df[, -1]
```

Now, inspect descriptive statistics for the variables in the dataset. This is an important initial data-screening step because it helps us understand the structure of the data and identify features such as variable ranges, means, standard deviations, missing values, and the number of unique observed values.

```
table_desc <- descriptives(df)
table_desc
```

	name	type	n	missing	unique	mean	median	mode	mode_value	sd
1	Grade	numeric	290	0	3	2.044828	2	2	NA	0.9044627
2	Age	numeric	290	0	4	13.758621	14	14	NA	0.9577646
3	Gender	numeric	290	0	2	1.600000	2	2	NA	0.4907448
4	TS1	numeric	290	0	7	2.944828	3	3	NA	1.5464182
5	TS2	numeric	290	0	7	3.110345	3	3	NA	1.5479297
6	TS3	numeric	290	0	7	2.541379	2	2	NA	1.6130989
7	TS4	numeric	290	0	7	2.962069	3	3	NA	1.4535278
8	SOS1	numeric	290	0	7	4.624138	5	5	NA	1.6762446
9	SOS2	numeric	290	0	7	3.468966	3	3	NA	1.7805375
10	SOS3	numeric	290	0	7	2.724138	2	2	NA	1.8168102
11	SOS4	numeric	290	0	7	2.920690	3	3	NA	1.4754928
12	PS1	numeric	290	0	7	4.603448	5	5	NA	1.6568998
13	PS2	numeric	290	0	7	2.627586	2	2	NA	1.5934330
14	PS3	numeric	290	0	7	3.189655	3	3	NA	1.6993475
15	PS4	numeric	290	0	7	3.351724	3	3	NA	1.8546902
16	PS5	numeric	290	0	7	3.048276	3	3	NA	1.7522345
17	AM1	numeric	290	0	7	4.431034	4	4	NA	1.5261512
18	AM2	numeric	290	0	7	4.806897	5	5	NA	1.4637325
19	AM3	numeric	290	0	7	4.603448	4	4	NA	1.4757516
20	AM4	numeric	290	0	7	4.500000	4	4	NA	1.4676441
21	AM5	numeric	290	0	7	4.713793	5	5	NA	1.4918983
22	CM1	numeric	290	0	7	2.889655	3	3	NA	1.5002525
23	CM2	numeric	290	0	7	2.865517	3	3	NA	1.4381784
24	CM3	numeric	290	0	7	3.306897	4	4	NA	1.5559754
25	CM4	numeric	290	0	7	3.427586	4	4	NA	1.6290818
26	BI1	numeric	290	0	7	4.782759	5	5	NA	1.4376473
27	BI2	numeric	290	0	7	4.913793	5	5	NA	1.3680042
28	BI3	numeric	290	0	7	4.982759	5	5	NA	1.4128841
29	BH1	numeric	290	0	7	4.986207	5	5	NA	1.7068403
30	BH2	numeric	290	0	7	3.465517	3	3	NA	1.5341050
	v	min	max	range	skew	skew_2se	kurt	kurt_2se		
1	NA	1	3	2	-0.0886256	-0.3096579	-1.77749840	-3.11572251		
2	NA	12	15	3	-0.1663658	-0.5812821	-1.00005803	-1.75297110		

3	NA	1	2	1	-0.4103740	-1.4338470	-1.84436097	-3.23292384
4	NA	1	7	6	0.3584376	1.2523814	-0.65471422	-1.14762850
5	NA	1	7	6	0.3272700	1.1434818	-0.60967008	-1.06867201
6	NA	1	7	6	0.8463815	2.9572575	-0.11074752	-0.19412594
7	NA	1	7	6	0.2025191	0.7076019	-0.76403094	-1.33924644
8	NA	1	7	6	-0.3195548	-1.1165248	-0.71102556	-1.24633492
9	NA	1	7	6	0.3438411	1.2013811	-0.78467060	-1.37542505
10	NA	1	7	6	0.9002934	3.1456257	-0.25172557	-0.44124204
11	NA	1	7	6	0.4304447	1.5039743	-0.59766759	-1.04763322
12	NA	1	7	6	-0.3071955	-1.0733413	-0.51186072	-0.89722498
13	NA	1	7	6	0.7964309	2.7827304	-0.11852521	-0.20775920
14	NA	1	7	6	0.2628300	0.9183282	-1.05909207	-1.85645004
15	NA	1	7	6	0.2754714	0.9624973	-1.03640628	-1.81668482
16	NA	1	7	6	0.4504039	1.5737115	-0.83333422	-1.46072603
17	NA	1	7	6	-0.3711418	-1.2967696	-0.07259923	-0.12725697
18	NA	1	7	6	-0.5346357	-1.8680175	0.15431420	0.27049264
19	NA	1	7	6	-0.3145517	-1.0990438	-0.03880199	-0.06801482
20	NA	1	7	6	-0.2149216	-0.7509362	-0.05178496	-0.09077228
21	NA	1	7	6	-0.4510817	-1.5760798	0.02413972	0.04231378
22	NA	1	7	6	0.6662803	2.3279840	0.10915634	0.19133680
23	NA	1	7	6	0.3500659	1.2231304	-0.49225767	-0.86286340
24	NA	1	7	6	0.2778648	0.9708599	-0.35040617	-0.61421625
25	NA	1	7	6	0.3380570	1.1811716	-0.36721528	-0.64368042
26	NA	1	7	6	-0.4300435	-1.5025725	0.12655711	0.22183808
27	NA	1	7	6	-0.6191560	-2.1633314	0.51726084	0.90669069
28	NA	1	7	6	-0.6660460	-2.3271653	0.53137380	0.93142885
29	NA	1	7	6	-0.4282368	-1.4962598	-0.68240157	-1.19616081
30	NA	1	7	6	0.6271490	2.1912593	0.07026806	0.12317074

The `type` column tells you the measurement level of each variable. The `tidySEM` package supports latent class analysis with `numeric` (or `integer`) and `ordered` variables.

Numeric and integer variables should be truly continuous (e.g., centimeters), or you must be willing to assume that they behave as continuous (scale mean scores in a questionnaire).

Ordinal indicators consist of categories that have a meaningful order, such as Likert responses ranging from *strongly disagree* to *strongly agree*.

Note that integer variables containing only a small number of values could be numeric score, but could also be ordered. The correct treatment depends on what the numbers mean, not merely on how R stores them. Check the `unique` column of `descriptives()` to see how many unique values there are; this may alert you to such issues.

If you have variables with other data types than numeric/integer/ordered - you will need to convert them.

One key issue is that variables are not always correctly stored. The `type` column may not correspond to the variable's substantive measurement level. For example, you might have a variable of type `integer` with values 1, 2, but which actually represents employment, with values `Unemployed` and `Employed`. Moreover, an integer variable may represent a truly continuous quantity (e.g., number of children), or an ordinal Likert scale, in which case it should be recoded to `ordered`. Similarly, a factor may represent an unordered categorical variable with several categories. For LCA, each category should be coded as a binary `ordered` variable.

You should therefore inspect both the variable type and the meaning of the variable before deciding how it should enter the model.

Quiz

Q1: An integer variable with values (1,2), which represent `Employed` and `Unemployed`, should be recoded to factor.

- ☐ True
- ☐ False

Q2: An ordinal Likert scale should be coded as a numeric variable.

- ☐ True
- ☐ False

Q3: The factor variable 'major in university', with four different values, should be coded as four binary variables of type `ordered` for LCA.

- ☐ True
- ☐ False

Answers

Q1: TRUE
Q2: FALSE
Q3: TRUE

1.3 Data Preprocessing

If a variable is incorrectly coded, you may be able to restore its correct type. Here are a few examples of type conversions:

```
# This is a character variable, because someone mistyped a t:
v1 <- c("5.1", "4.9", "4.7", "4.6t", "5", "5.4")
class(v1)
```

```
[1] "character"
```

```
# You can turn it into numeric; the mistyped value is lost:  
as.numeric(v1)
```

```
[1] 5.1 4.9 4.7 NA 5.0 5.4
```

```
# This is an integer variable representing a Likert scale:  
v2 <- c(2L, 3L, 2L, 5L, 3L, 4L)  
class(v2)
```

```
[1] "integer"
```

```
# You can turn it into ordered:  
ordered(v2)
```

```
[1] 2 3 2 5 3 4  
Levels: 2 < 3 < 4 < 5
```

```
# This is a factor variable:  
v3 <- factor(c("Medicine", "Humanities", "Humanities", "Medicine", "Social science",  
"Medicine"))  
class(v3)
```

```
[1] "factor"
```

```
# You can encode it as multiple binary dummies:  
tidySEM::mx_dummies(v3)
```

	Humanities	Medicine	Social.science
1	0	1	0
2	1	0	0
3	1	0	0
4	0	1	0
5	0	0	1
6	0	1	0

The final command takes an unordered factor and represents it as a set of binary dummy variables; one for each category. These dummy variables can then be added to the analysis dataset.

Note that, in our example dataset, all variable `types` are numeric, even though some must be integer (age), and others must be factors (Gender) or ordered (all Likert scales). Note that all variables have just few low number of unique values; this is another hint that some recoding is in order.

Here we recode ‘gender’ from its original numeric codes into a factor variable. We also attach meaningful text labels to the numeric categories so that subsequent tables and output are easier to understand. The ‘levels’ argument specifies the original codes found in the dataset, while ‘labels’ specifies the corresponding category names that R should display instead.

```
df$Gender <- factor(df$Gender,
                    levels = c(1,2),
                    labels = c("M", "F"))
```

Several variables in the dataset are individual questionnaire items that belong to broader multi-item scales.

We can use these to practice two things:

- 1) Computing numeric scale mean scores
- 2) Encoding them as ordered variables

We will compute the scale scores first, because this still requires these variables to be **numeric**.

For example, the scale named TS consists of TS1 through TS4, while SOS consists of SOS1 through SOS4. I like to store this information in a list, as this would allow you to, for example, easily perform psychometric analyses for every scale automatically (which is out of scope for the current tutorial).

```
# To prepare the scales list, I used:
scales_list <- grep("\\w{2}\\d", names(df), value = TRUE)
scalenames <- unique(gsub("\\d", "", scales_list))
scales_list <- lapply(scalenames, grep, x = names(df), value = TRUE)
names(scales_list) <- scalenames

# This gives the list below; you can also construct it for your dataset by hand:
scales_list <- list(
  TS = c("TS1", "TS2", "TS3", "TS4"),
  SOS = c("SOS1", "SOS2", "SOS3", "SOS4"),
  PS = c("PS1", "PS2", "PS3", "PS4", "PS5"),
```

```

AM = c("AM1", "AM2", "AM3", "AM4", "AM5"),
CM = c("CM1", "CM2", "CM3", "CM4"),
BI = c("BI1", "BI2", "BI3"),
BH = c("BH1", "BH2")
)

```

To compute mean scale scores, we can use:

```

df_scales <- sapply(scales_list, function(items){
  rowMeans(df[, items, drop = FALSE], na.rm = TRUE)
})
# Add them to df:
df <- data.frame(df, df_scales)

```

Second, to encode these items as ordered variables, we could grab all relevant variables based on the naming pattern (these variable names end with a number), or we can take all items in the `scales_list` using `unlist()`:

```

ord_vars <- grep("\\d$", names(df), value = TRUE)
ord_vars

```

```

[1] "TS1" "TS2" "TS3" "TS4" "SOS1" "SOS2" "SOS3" "SOS4" "PS1" "PS2"
[11] "PS3" "PS4" "PS5" "AM1" "AM2" "AM3" "AM4" "AM5" "CM1" "CM2"
[21] "CM3" "CM4" "BI1" "BI2" "BI3" "BH1" "BH2"

```

```

# Or alternatively:
ord_vars <- unlist(scales_list)

```

Then, we can recode these variables as ordered:

```

df[ord_vars] <- lapply(df[ord_vars], ordered)

```

Inspect the cleaned dataset again before fitting the model. Check that numeric indicators really represent continuous quantities, ordered indicators have the intended category order, and binary indicators contain exactly two possible response categories. Also check for unexpected values, unused factor levels, miscoded missing values, and variables with little or no variation.

1.4 Missing Data

This dataset has no missing data. However, you can increase the challenge for yourself by adding some missingness. If you want to do this - do it immediately after loading the data and removing the empty rows, with code like this:

```
library(mice)
df <- ampute(df)$amp
```

Regardless of the presence of missingness or not, **tidySEM** uses FIML which includes cases with partially missing data.

1.5 Model Specification

The type of model you can specify depends partially on the measurement level of your indicators:

Indicator type	Typical examples	tidySEM function	Class-specific quantities
Continuous	test scores, reaction time, symptom sum scores	<code>mx_profiles()</code>	means, variances, possibly covariances
Ordinal	Likert items, ordered severity categories	<code>mx_lca()</code>	category probabilities / thresholds
Dichotomous	yes/no, present/absent, correct/incorrect	<code>mx_lca()</code>	response probabilities
Mixed	a combination of the above	<code>mx_mixed_lca()</code>	parameters appropriate to each indicator type

Quiz

Q1: A dataset containing three mean scale score variables would be analyzed with:

- A. `mx_profiles()`
- B. `mxModel()`
- C. `mx_mixed_lca()`

D. `mx_lca()`

Q2: A dataset containing ten ordinal questionnaire items would be analyzed with:

- A. `mx_lca()`
- B. `mx_mixed_lca()`
- C. `mxModel()`
- D. `mx_profiles()`

Q3: A dataset containing both continuous scale scores and binary indicators would require:

- A. `mx_profiles()`
- B. `mx_mixed_lca()`
- C. `mx_lca()`
- D. `mxModel()`

Answers

Q1: `mx_profiles()`

Q2: `mx_lca()`

Q3: `mx_mixed_lca()`

1.6 Latent Profile Analysis (LPA) with Continuous Indicators

We will use two scale scores from the Zhao dataset as a teaching example; specifically, `c("AM", "CM")`, which represent autonomous and controlled motivation. You can increase the challenge level of this tutorial by using your own dataset, or by selecting different scales. For example, `c("TS", "SOS", "PS")`, which represent three sources of stress/pressure: teacher stress, social stress, and parent stress.

```
df_lpa <- df[, c("AM", "CM")]
```

Note that standardizing indicators is not required and should not affect results, with two caveats: if continuous indicators are on very different scales (e.g., the variance differs by a factor 10+), this can cause problems for model convergence. Second, this can make plots harder to read because the plotting range of one variable could be much smaller than another.

1.6.1 Model Specification

The arguments `variances` and `covariances` control the model specification for LPA. For example, a model with equal variances and zero covariances assumes that every indicator has the same residual variance across classes, and that indicators are “locally independent” (=uncorrelated) within each class, even if they should be correlated at the sample level. Note that you can also specify different (co)variance structures, either for thorough exploration of because you have some theory about e.g., smaller variance in some classes, or different correlations between classes. See the help file by running `?mx_profiles`. For example, you could specify the code below.

Note that this analysis will take a very long time because these are many models and some of them are hard to estimate. For any computationally intensive analyses like this, we can save the results to a file, so you can load the results from that file directly without having to wait for the code to run. We do recommend that you experiment with running *some* analyses yourself, of course.

```
res_lpa <- mx_profiles(
  df_lpa,
  classes = 1:5,
  variances = c("equal", "varying"),
  covariances = c("zero", "equal")
)
saveRDS(res_lpa, "res_lpa_different_spect.rds")
```

```
res_lpa <- readRDS("res_lpa_different_spect.rds")
format_numeric(table_fit(res_lpa))
```

		Name	Classes	LL	n	Parameters	AIC	BIC	
1		equal var 1	1.00	-938.90	290.00	4	1885.79	1900.47	
2		equal var 2	2.00	-915.49	290.00	7	1844.99	1870.68	
3		equal var 3	3.00	-894.71	290.00	10	1809.41	1846.11	
4		equal var 4	4.00	-888.40	290.00	13	1802.80	1850.51	
5		equal var 5	5.00	-885.87	290.00	16	1803.74	1862.46	
6	free var, equal cov 1	1.00	-913.54	290.00	5	1837.07	1855.42		
7	free var, equal cov 2	2.00	-904.62	290.00	10	1829.23	1865.93		
8	free var, equal cov 3	3.00	-794.43	290.00	15	1618.85	1673.90		
9	free var, equal cov 4	4.00	-754.85	290.00	20	1549.70	1623.10		
10	free var, equal cov 5	5.00	-828.60	290.00	25	1707.19	1798.94		
	saBIC	Entropy	prob_min	prob_max	n_min	n_max	warning	np_ratio	np_local
1	1887.79	1.00	1.00	1.00	1.00	1.00	NA	72.50	72.50
2	1848.48	0.70	0.69	0.97	0.15	0.85	NA	41.43	14.67

3	1814.40	0.82	0.87	0.95	0.10	0.61	NA	29.00	10.50
4	1809.28	0.85	0.80	0.95	0.05	0.58	NA	22.31	6.00
5	1811.72	0.82	0.48	0.96	0.03	0.58	NA	18.12	3.75
6	1839.57	1.00	1.00	1.00	1.00	1.00	NA	58.00	58.00
7	1834.22	0.29	0.51	0.91	0.24	0.76	NA	29.00	15.56
8	1626.33	0.79	0.77	1.00	0.16	0.60	TRUE	19.33	10.85
9	1559.67	0.83	0.84	1.00	0.08	0.38	TRUE	14.50	5.41
10	1719.66	0.86	0.80	0.97	0.02	0.44	TRUE	11.60	1.43

More complex models are not automatically better: extra parameters can create unstable solutions, tiny classes, or local maxima. Think about which models make sense to estimate, and how many classes your data could reasonably support.

1.6.2 Maximum Number of Classes

We can set an upper limit on the number of classes to estimate, for example, by saying we want to aim for 20 observations per parameter, if cases would be evenly split between classes. We could then estimate $290/20 = 14.5$ parameters. With 2 indicators, zero covariances, and equal variances, the number of parameters for our model is:

- k-1 mixture weights
- k*2 means
- 2 variances

Which would give this many cases per parameter:

```
k <- 1:10
nrow(df) / (1+3*k)
```

```
[1] 72.500000 41.428571 29.000000 22.307692 18.125000 15.263158 13.181818
[8] 11.600000 10.357143  9.354839
```

So we could estimate at least 4 classes. Let's round up and estimate 1:5 class solutions:

```
res_lpa <- mx_profiles(
  df_lpa,
  classes = 1:5,
  variances = "equal",
  covariances = "zero"
)
saveRDS(res_lpa, "res_lpa_equal.rds")
```

```
res_lpa <- readRDS("res_lpa_equal.rds")
```

1.6.3 Class Enumeration

First, extract a model fit table:

```
tab_lpa_fit <- table_fit(res_lpa)
tab_lpa_fit
```

	Name	Classes	LL	n	Parameters	AIC	BIC	saBIC	
1	equal	var 1	1	-938.8958	290	4	1885.792	1900.471	1887.786
2	equal	var 2	2	-915.4934	290	7	1844.987	1870.676	1848.478
3	equal	var 3	3	-894.7056	290	10	1809.411	1846.110	1814.398
4	equal	var 4	4	-892.8343	290	13	1811.669	1859.377	1818.152
5	equal	var 5	5	-885.8717	290	16	1803.743	1862.462	1811.723
	Entropy	prob_min	prob_max	n_min	n_max	np_ratio	np_local		
1	1.0000000	1.0000000	1.0000000	1.00000000	1.00000000	72.50000	72.50000		
2	0.6995815	0.6881145	0.9677946	0.15172414	0.8482759	41.42857	14.66667		
3	0.8159663	0.8741811	0.9504706	0.09655172	0.6137931	29.00000	10.50000		
4	0.7966132	0.3619379	0.9541764	0.02068966	0.6137931	22.30769	2.40000		
5	0.8231484	0.4849168	0.9562938	0.03103448	0.5793103	18.12500	3.75000		

1.6.3.1 BIC

One well-supported criterion for class enumeration is the BIC: a relative fit index that rewards good fit and penalizes model complexity. The absolute BIC value has no substantive meaning; you can compare BIC values of models fitted to the same data, and lower values are preferred. We can see that the three-class model has the lowest BIC:

```
which.min(tab_lpa_fit$BIC)
```

```
[1] 3
```

1.6.3.2 BLRT

We can also conduct a bootstrapped likelihood ratio test, which asks whether a model with k classes improves fit over a model with $k - 1$ classes. However, this test is extremely computationally expensive. Before we do so, it is useful to check if we can eliminate some models from consideration (also see next section). For example, note that `np_local`, the minimum number

of cases assigned to a class divided by the number of parameters in that class, is as low as 2.4 for models with 4-5 classes. You want a healthy number of cases per parameter, so we can exclude these two models, then perform our BLRT:

```
res_lpa_blrt <- res_lpa
res_lpa_blrt[c(4, 5)] <- NULL
res_blrt <- BLRT(res_lpa_blrt)
saveRDS(res_blrt, "res_blrt_intro.rds")
```

```
res_blrt <- readRDS("res_blrt_intro.rds")
res_blrt
```

Bootstrapped Likelihood Ratio Test:

	null	alt	lr	df	blrt_p	samples
mix1	mix2	46.8	3	0	100	
mix2	mix3	41.6	3	0	98	

We see that adding classes leads to a significant improvement in model fit.

1.6.4 Eliminating Models from Consideration

As mentioned before, we will eliminate the 4-5 class models because they have a very low number of observations per parameter.

We can also check for any model convergence problems; a value other than 0 for the convergence status can indicate a problem:

```
sapply(res_lpa, function(x){x$output$status$status})
```

equal	var 1	equal	var 2	equal	var 3	equal	var 4	equal	var 5
	0		0		0		0		0

Another consideration is that classification diagnostics are poor for the 4-5 class models:

```
tab_lpa_fit[, c("Entropy", "prob_min", "prob_max")]
```

	Entropy	prob_min	prob_max
1	1.0000000	1.0000000	1.0000000
2	0.6995815	0.6881145	0.9677946
3	0.8159663	0.8741811	0.9504706
4	0.7966132	0.3619379	0.9541764
5	0.8231484	0.4849168	0.9562938

The entropy statistic is an overall summary of class separation, where values approaching 1 indicate clear delineation of classes (Celeux & Soromenho, 1996).

The statistics `prob_min` and `prob_max` give us the minimum and maximum of the diagonal of the `mostlikely.class` classification table (see `?class_prob`). If C is the true class of an observation, and N is the most likely class based on the model, then this table shows the probability $P(N=i|C=j)$. The diagonal represents the probability that observations in each class will be correctly classified. We see that `prob_min` is very low for the 4-5 class models; thus, for some classes, there is a low probability that its members will be correctly classified. This is also reflected in the entropy.

1.6.5 Reporting Results

We already obtained the model fit table. There are many ways to prepare it for a scientific paper. The regular formatting function in R has some peculiar behavior; it does not guarantee that a certain number of decimals are displayed. This is a design choice in R, and has to do with the principle of “significant figures”. However, APA style simply requires reporting to double digits. So let’s consider two options:

```
# Standard R formatting, some columns get 3 digits, some get 1, but they remain
# numeric type:
tab_lpa_fit_formatted <- format(tab_lpa_fit, digits = 2)
tab_lpa_fit_formatted
```

	Name	Classes	LL	n	Parameters	AIC	BIC	saBIC	Entropy	prob_min
1	equal var	1	-939	290	4	1886	1900	1888	1.00	1.00
2	equal var	2	-915	290	7	1845	1871	1848	0.70	0.69
3	equal var	3	-895	290	10	1809	1846	1814	0.82	0.87
4	equal var	4	-893	290	13	1812	1859	1818	0.80	0.36
5	equal var	5	-886	290	16	1804	1862	1812	0.82	0.48

	prob_max	n_min	n_max	np_ratio	np_local
1	1.00	1.000	1.00	72	72.5
2	0.97	0.152	0.85	41	14.7
3	0.95	0.097	0.61	29	10.5
4	0.95	0.021	0.61	22	2.4

```
5      0.96 0.031  0.58      18      3.8
```

```
# Using format_numeric; reports 2 digits, but all columns become
# character type:
tab_lpa_fit_formatted <- format_numeric(tab_lpa_fit, digits = 2)
write.csv(tab_lpa_fit_formatted, "tab_lpa_fit.csv", row.names = FALSE)
```

Based on the BIC, the elimination of the 4-5 class models, and significant BLRT tests, we would retain a 3-class model.

The class proportions of this model are:

```
class_prob(res_lpa[[3]], "sum.posterior")
```

```
$sum.posterior
      class      count proportion
1 class1 171.83359  0.5925296
2 class2  88.42000  0.3048965
3 class3  29.74641  0.1025738
```

We can examine the parameters of this model as follows:

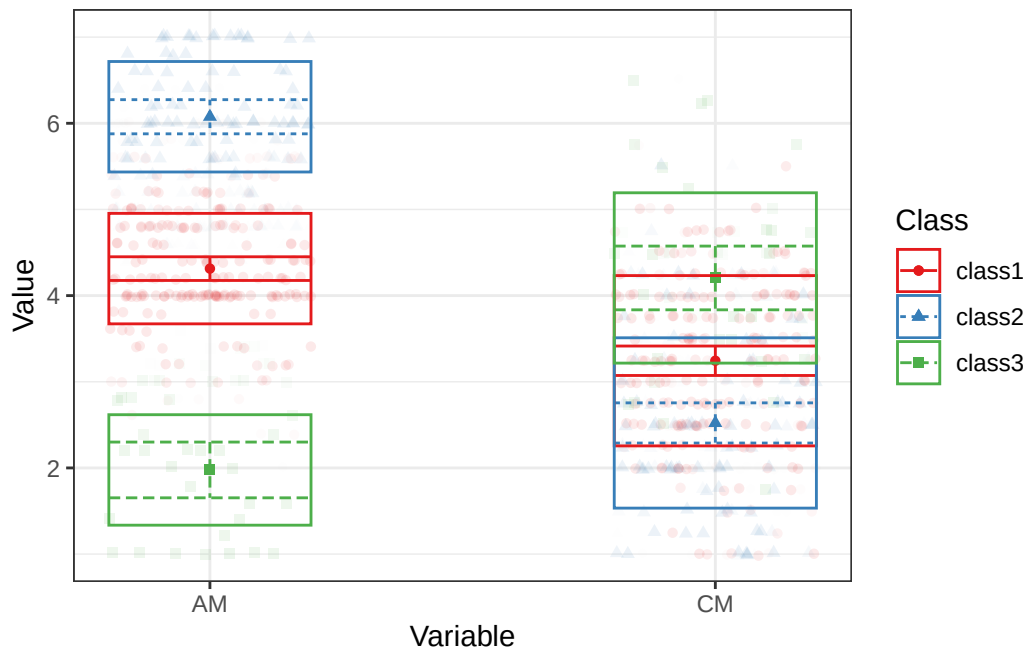
```
tab_lpa_res <- table_results(res_lpa[[3]])
tab_lpa_res
```

	label	est_sig	se	pval	confint	class
1	Means.AM.class1	4.31***	0.07	0.00	[4.18, 4.45]	class1
2	Means.CM.class1	3.24***	0.09	0.00	[3.07, 3.41]	class1
3	Variances.AM.class1	0.41***	0.05	0.00	[0.32, 0.51]	class1
4	Variances.CM.class1	0.98***	0.09	0.00	[0.81, 1.14]	class1
5	Means.AM.class2	6.08***	0.10	0.00	[5.88, 6.27]	class2
6	Means.CM.class2	2.52***	0.12	0.00	[2.29, 2.76]	class2
7	Means.AM.class3	1.98***	0.17	0.00	[1.65, 2.30]	class3
8	Means.CM.class3	4.20***	0.19	0.00	[3.83, 4.57]	class3
9	mix3.weights[1,1].NA	1.00	<NA>	<NA>	<NA>	<NA>
10	mix3.weights[1,2].NA	0.51***	0.09	0.00	[0.34, 0.69]	<NA>
11	mix3.weights[1,3].NA	0.17***	0.04	0.00	[0.09, 0.25]	<NA>

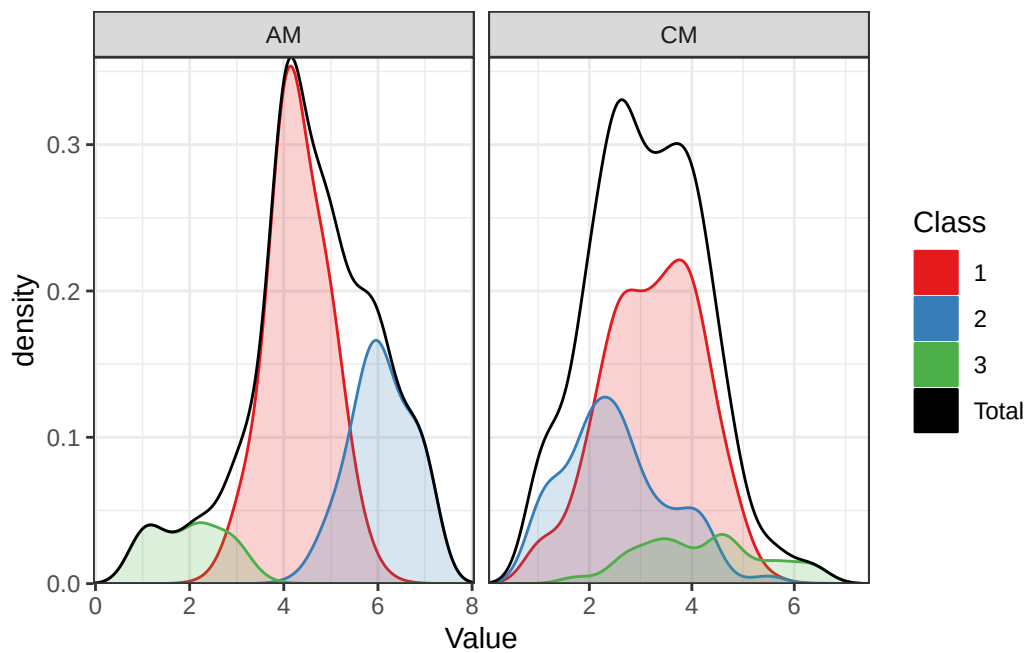
Notice that we get one estimate for the variances (because these are equal across classes), no covariances, and unique estimates for the means across classes.

The pattern of results is easier to see when plotting it:

```
plot_profiles(res_lpa[[3]])
```



```
plot_density(res_lpa[[3]])
```



Quiz

Q1: Which class has moderate values on both indicators?

- A. Class 3
- B. Class 1
- C. Class 2

Q2: Class 2 has a high mean for both AM and CM

- ☐ True
- ☐ False

Q3: The marginal (whole-sample) distribution of both variables is pretty close to normally distributed.

- ☐ True
- ☐ False

Q4: Based on these results (relatively low class separability and approximately normal distribution of indicators, we should seriously consider the possibility that there are no distinct classes.

- ☐ True
- ☐ False

Answers

Q1: Class 1
Q2: FALSE
Q3: TRUE
Q4: TRUE

1.7 Latent Class Analysis with Ordinal indicators

We can conduct ordinal LCA by using some of the raw items underlying the scale scores, for example:

```
df_lca <- df[, c("AM1", "AM2", "AM3")]
```

For a model with ordinal indicators, there is (currently) only one parametrization available. Specifically, we represent each ordinal variable as a latent normal variable with thresholds representing the quantiles of people who score a particular score or lower. In these models,

each variable will have a number of thresholds equal to the number of response options minus one. It is easy to see that this model will have more parameters than the LPA model. If L is the number of levels of our ordinal indicators, and M is the number of indicators:

- $k-1$ mixture weights
- $k((L-1)M)$ thresholds

Since our indicators have 7 categories, this gives:

```
k <- 1:10
L <- length(unique(df_lca$AM1))
M <- ncol(df_lca)
nrow(df) / ((k-1)+k*((L-1)*M))
```

```
[1] 16.111111  7.837838  5.178571  3.866667  3.085106  2.566372  2.196970
[8]  1.920530  1.705882  1.534392
```

Note that even for a 1-class model, we have fewer than 20 observations per parameter. Thus, we might doubt whether we have enough data here to estimate more than one class. For educational purposes - we will proceed, but to save time estimating overly complex models, we omit the 4-5 class models.

```
res_lca <- mx_lca(
  df_lca,
  classes = 1:3
)
saveRDS(res_lca, "res_lca_intro.rds")
```

```
res_lca <- readRDS("res_lca_intro.rds")
tab_lca_fit <- table_fit(res_lca)
tab_lca_fit
```

	Name	Classes	LL	n	Parameters	AIC	BIC	saBIC	Entropy
1	1	1	-1485.423	290	18	3006.847	3072.905	3015.824	1.0000000
2	2	2	-1313.780	290	37	2701.559	2837.345	2720.011	0.8707661
3	3	3	-1230.037	290	56	2572.075	2777.588	2600.002	0.8956807
	prob_min	prob_max	n_min	n_max	np_ratio	np_local			
1	1.0000000	1.0000000	1.0000000	1.0000000	16.111111	16.111111			
2	0.9542104	0.9690095	0.4931034	0.5068966	7.837838	7.944444			
3	0.9513416	0.9666826	0.1586207	0.5103448	5.178571	2.555556			

The BIC indicates that adding classes improves model fit. However, even for the 2-class model, the number of cases/parameters in the smallest class (`np_local`) is smaller than 10. Thus, we might be skeptical about selecting more than 1 class for these data. For educational purposes, however, let's examine the 2-class solution.

```
tab_lca_res <- table_results(res_lca[[2]])
tab_lca_res
```

	label	est_sig	se	pval	confint	class
1	Variances.AM1.class1	1.00	<NA>	<NA>	<NA>	class1
2	Variances.AM2.class1	1.00	<NA>	<NA>	<NA>	class1
3	Variances.AM3.class1	1.00	<NA>	<NA>	<NA>	class1
4	class1.Thresholds[1,1].class1	-1.14***	0.13	0.00	[-1.40, -0.88]	class1
5	class1.Thresholds[1,2].class1	-1.45***	0.15	0.00	[-1.75, -1.15]	class1
6	class1.Thresholds[1,3].class1	-1.40***	0.15	0.00	[-1.70, -1.11]	class1
7	class1.Thresholds[2,1].class1	-0.84***	0.12	0.00	[-1.07, -0.60]	class1
8	class1.Thresholds[2,2].class1	-1.08***	0.13	0.00	[-1.33, -0.82]	class1
9	class1.Thresholds[2,3].class1	-0.99***	0.13	0.00	[-1.24, -0.75]	class1
10	class1.Thresholds[3,1].class1	-0.34**	0.11	0.00	[-0.56, -0.13]	class1
11	class1.Thresholds[3,2].class1	-0.64***	0.11	0.00	[-0.87, -0.42]	class1
12	class1.Thresholds[3,3].class1	-0.52***	0.11	0.00	[-0.74, -0.30]	class1
13	class1.Thresholds[4,1].class1	1.53***	0.21	0.00	[1.13, 1.94]	class1
14	class1.Thresholds[4,2].class1	0.78***	0.14	0.00	[0.49, 1.06]	class1
15	class1.Thresholds[4,3].class1	1.29***	0.18	0.00	[0.94, 1.65]	class1
16	class1.Thresholds[5,1].class1	2.22***	0.36	0.00	[1.53, 2.92]	class1
17	class1.Thresholds[5,2].class1	1.58***	0.19	0.00	[1.21, 1.95]	class1
18	class1.Thresholds[5,3].class1	2.61***	0.58	0.00	[1.47, 3.75]	class1
19	class1.Thresholds[6,1].class1	2.47***	0.36	0.00	[1.77, 3.17]	class1
20	class1.Thresholds[6,2].class1	1.79***	0.20	0.00	[1.40, 2.18]	class1
21	class1.Thresholds[6,3].class1	2.61***	0.57	0.00	[1.50, 3.73]	class1
22	Variances.AM1.class2	1.00	<NA>	<NA>	<NA>	class2
23	Variances.AM2.class2	1.00	<NA>	<NA>	<NA>	class2
24	Variances.AM3.class2	1.00	<NA>	<NA>	<NA>	class2
25	class2.Thresholds[1,1].class2	-165.71	<NA>	<NA>	<NA>	class2
26	class2.Thresholds[1,2].class2	-12.90	<NA>	<NA>	<NA>	class2
27	class2.Thresholds[1,3].class2	-300.65	<NA>	<NA>	<NA>	class2
28	class2.Thresholds[2,1].class2	-162.99	<NA>	<NA>	<NA>	class2
29	class2.Thresholds[2,2].class2	-10.43	<NA>	<NA>	<NA>	class2
30	class2.Thresholds[2,3].class2	-13.60	<NA>	<NA>	<NA>	class2
31	class2.Thresholds[3,1].class2	-2.14***	0.29	0.00	[-2.71, -1.57]	class2
32	class2.Thresholds[3,2].class2	-8.33	<NA>	<NA>	<NA>	class2
33	class2.Thresholds[3,3].class2	-2.45***	0.36	0.00	[-3.17, -1.74]	class2

```

34 class2.Thresholds[4,1].class2 -1.04*** 0.17 0.00 [-1.37, -0.72] class2
35 class2.Thresholds[4,2].class2 -1.88* 0.87 0.03 [-3.58, -0.18] class2
36 class2.Thresholds[4,3].class2 -1.24*** 0.18 0.00 [-1.60, -0.88] class2
37 class2.Thresholds[5,1].class2 -0.09 0.12 0.46 [-0.31, 0.14] class2
38 class2.Thresholds[5,2].class2 -0.37 0.84 0.66 [-2.01, 1.27] class2
39 class2.Thresholds[5,3].class2 -0.24 0.12 0.05 [-0.48, 0.00] class2
40 class2.Thresholds[6,1].class2 0.95*** 0.13 0.00 [0.70, 1.20] class2
41 class2.Thresholds[6,2].class2 0.76 0.84 0.36 [-0.88, 2.40] class2
42 class2.Thresholds[6,3].class2 0.74*** 0.12 0.00 [0.50, 0.98] class2
43          mix2.weights[1,1].NA      1.00 <NA> <NA>          <NA> <NA>
44          mix2.weights[1,2].NA      0.94*** 0.13 0.00 [0.68, 1.20] <NA>

```

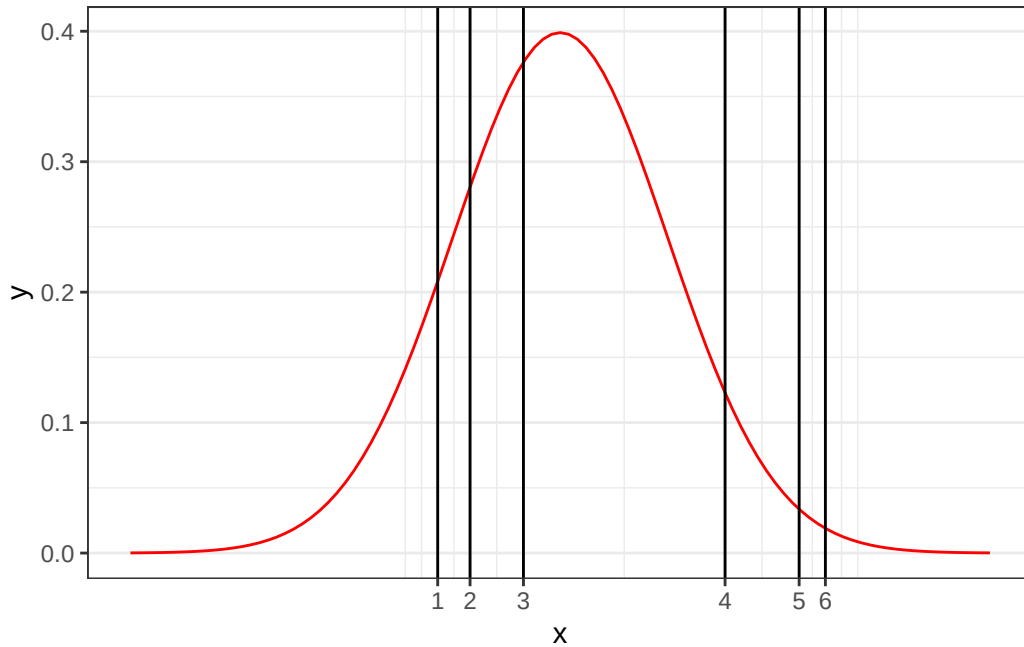
We see that all the parameters are thresholds (and a mixture weight).

These thresholds are z-values at which a standard normal distribution would be intersected so that the proportion of participants falling below the threshold corresponds to the proportion of participants who selected that answer category or lower. We can plot this to give you an intuitive understanding, for example, for the thresholds for variable AM1 in class 1:

```

library(ggplot2)
thresholds <- res_lca[[2]]$class1$Thresholds$result[, "AM1"]
ggplot(data.frame(x = c(-4, 4)), aes(x = x)) +
  stat_function(fun = dnorm, color = "red") +
  geom_vline(xintercept = thresholds) +
  scale_x_continuous(breaks = thresholds, labels = 1:6) +
  theme_bw()

```



This figure implies that the most common response to AM1 in class 1 is 4, quite a few people score 1, and very few people score 6 or 7. We can do a sanity check to confirm this interpretation by assigning people to their most likely class (note that this ignores classification uncertainty), and then tabulating the observed scores of AM1 for the people who are assigned class 1:

```
individual_class <- predict_class(res_lca[[2]])
table(df$AM1[individual_class == 1])
```

```
 1  2  3  4  5  6  7
19 11 25 85  5  1  1
```

Another way to interpret the threshold parameters is to convert them to the estimated probability of choosing each response category:

```
tab_lca_prob <- table_prob(res_lca[[2]])
# Let's format it for easier visual interpretation:
format_numeric(tab_lca_prob, digits = 2)
```

	Variable	Category	Probability	group
1	AM1	1	0.13	class1
2	AM1	2	0.07	class1

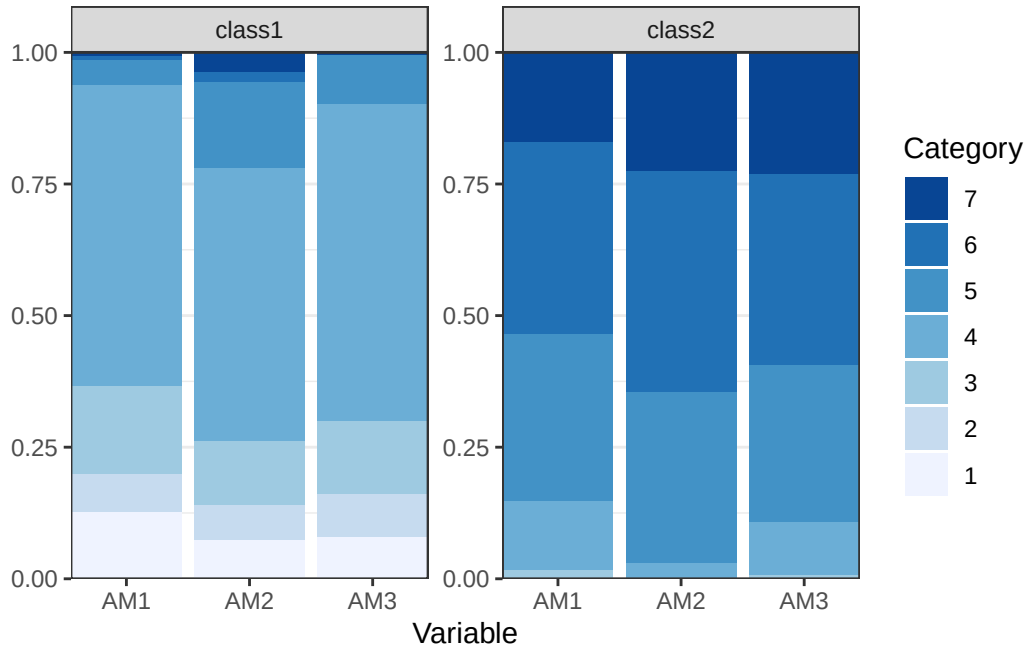
3	AM1	3	0.17 class1
4	AM1	4	0.57 class1
5	AM1	5	0.05 class1
6	AM1	6	0.01 class1
7	AM1	7	0.01 class1
8	AM2	1	0.07 class1
9	AM2	2	0.07 class1
10	AM2	3	0.12 class1
11	AM2	4	0.52 class1
12	AM2	5	0.16 class1
13	AM2	6	0.02 class1
14	AM2	7	0.04 class1
15	AM3	1	0.08 class1
16	AM3	2	0.08 class1
17	AM3	3	0.14 class1
18	AM3	4	0.60 class1
19	AM3	5	0.09 class1
20	AM3	6	0.00 class1
21	AM3	7	0.00 class1
22	AM1	1	0.00 class2
23	AM1	2	0.00 class2
24	AM1	3	0.02 class2
25	AM1	4	0.13 class2
26	AM1	5	0.32 class2
27	AM1	6	0.36 class2
28	AM1	7	0.17 class2
29	AM2	1	0.00 class2
30	AM2	2	0.00 class2
31	AM2	3	0.00 class2
32	AM2	4	0.03 class2
33	AM2	5	0.33 class2
34	AM2	6	0.42 class2
35	AM2	7	0.22 class2
36	AM3	1	0.00 class2
37	AM3	2	0.00 class2
38	AM3	3	0.01 class2
39	AM3	4	0.10 class2
40	AM3	5	0.30 class2
41	AM3	6	0.36 class2
42	AM3	7	0.23 class2

We see that the majority of people in class 1 scored a 4 on all indicators, and the majority

of people in class 2 scored a 5 or 6 on all indicators. Thus, we could call these classes the “moderate AM” class, and the “high AM” class, respectively.

We can also plot this information, leading to the same conclusion:

```
plot_prob(res_lca[[2]])
```



1.8 Dichotomous indicators: `mx_lca()`

The Zhao dataset does not contain any binary indicators. To practice LCA with binary indicators, we will dichotomize the indicators from the previous example to practice.

Note that in real-world analyses, one should (almost) never artificially dichotomize variables, because this loses information. Because we saw that the main differences between the two classes in the previous example was between people scoring mostly 4 (class 1) versus 5 and 6 (class 2), it makes sense to cut the variables at this value:

```
df_bin <- df_lca
df_bin[] <- lapply(df_bin, function(x) ordered(as.integer(x <= 4)))
```

1.8.1 Model Fitting

The number of parameters is calculated the same as for the other ordinal LCA example, but now, the number of levels L is 2. If we divide the number of cases by the number of parameters, we see that we could estimate up to 4 classes while still having, on average, about 20 observations per parameter:

```
k <- 1:10
L <- 2
M <- ncol(df_lca)
nrow(df) / ((k-1)+k*((L-1)*M))
```

```
[1] 96.666667 41.428571 26.363636 19.333333 15.263158 12.608696 10.740741
[8]  9.354839  8.285714  7.435897
```

We can fit the model:

```
res_bin <- mx_lca(
  df_bin,
  classes = 1:4
)
saveRDS(res_bin, "res_bin_intro.rds")
```

```
res_bin <- readRDS("res_bin_intro.rds")
tab_bin_fit <- table_fit(res_bin)
tab_bin_fit
```

	Name	Classes	LL	n	Parameters	AIC	BIC	saBIC	Entropy
1	1	1	-597.1057	290	3	1200.2115	1211.2211	1201.7076	1.0000000
2	2	2	-459.3672	290	7	932.7345	958.4236	936.2254	0.9147806
3	3	3	-451.2779	290	11	924.5559	964.9246	930.0416	0.7583881
4	4	4	-451.2779	290	15	932.5559	987.6041	940.0364	0.5838172
	prob_min	prob_max	n_min	n_max	np_ratio	np_local			
1	1.0000000	1.0000000	1.0000000	1.0000000	96.66667	96.66667			
2	0.9733905	0.9866516	0.4379310	0.5620690	41.42857	42.33333			
3	0.8064599	0.9917367	0.2896552	0.3620690	26.36364	28.00000			
4	0.4351570	1.0000000	0.1310345	0.3724138	19.33333	12.66667			

Here, we see that the 2-class solution is best according to the BIC, and that `np_local` is near-zero for the 3- and 4-class solution. This happens because (almost) no observations are allocated to those classes, see `n_min`. Thus, these classes are superfluous.

We can examine the results for the 2-class solution. Note that, because there are only two response categories, we can ignore half the table because the response proportions sum to 1, so half the rows are just one minus the other half of the rows. The table now only shows what percentage of participants scored 1 on each indicator:

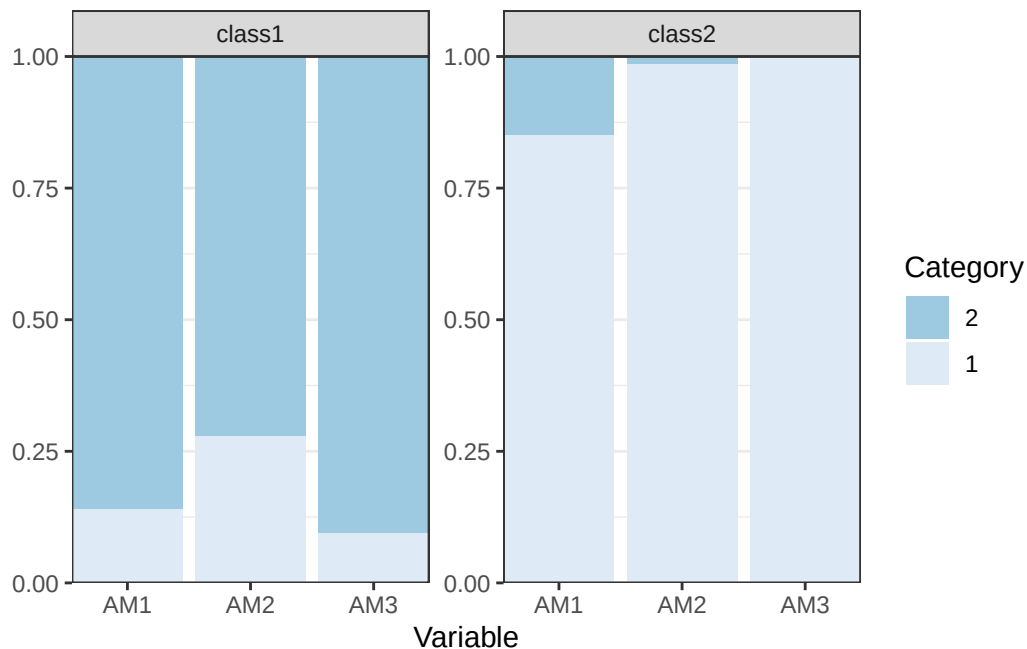
```
tab_bin_prob <- table_prob(res_bin[[2]])
tab_bin_prob <- tab_bin_prob[!tab_bin_prob$Category == 1, -2]
# Let's format it for easier visual interpretation:
format_numeric(tab_bin_prob, digits = 2)
```

	Variable	Probability	group
2	AM1	0.86	class1
4	AM2	0.72	class1
6	AM3	0.90	class1
8	AM1	0.15	class2
10	AM2	0.01	class2
12	AM3	0.00	class2

We see that people in class 2 tend to score 1 more often than people in class 1. This tells us that AM1 is most discriminating (nearly all people in class 2 endorse it, but nearly nobody in class 1), whereas AM2 is less discriminating.

We can also plot this information:

```
plot_prob(res_bin[[2]])
```



1.9 Mixed indicator types: `mx_mixed_lca()`

Many applied datasets contain continuous outcomes alongside ordered or binary indicators. `mx_mixed_lca()` is designed for this setting. As an example, we can combine `df_bin` with the scale score of CM:

```
df_mixed <- data.frame(df[, "CM", drop = FALSE], df_bin)
```

The parameters are now given as:

- $k-1$ mixture weights
- $k((L-1)\text{ncol}(\text{df_lca}))$ thresholds
- k means for CM
- 1 variance for CM (if we keep it constant across classes)
- No covariances

```
k <- 1:10
L <- 2
M <- ncol(df_lca)
nrow(df) / ((k-1)+k*((L-1)*M) + 1 + k)
```

```
[1] 58.000000 29.000000 19.333333 14.500000 11.600000 9.666667 8.285714
[8] 7.250000 6.444444 5.800000
```

So following our rule of thumb, we could safely estimate up to 3 classes:

```
res_mixed <- mx_mixed_lca(
  df_mixed,
  classes = 1:3
)
saveRDS(res_mixed, "res_mixed_intro.rds")
```

```
res_mixed <- readRDS("res_mixed_intro.rds")
tab_mixed_fit <- table_fit(res_mixed)
tab_mixed_fit
```

	Name	Classes	LL	n	Parameters	AIC	BIC	saBIC	Entropy
1	equal	1	-1036.9096	290	5	2083.819	2102.169	2086.313	1.0000000
2	equal1	2	-872.4004	290	10	1764.801	1801.500	1769.788	0.8734285
3	equal2	3	-869.7815	290	15	1769.563	1824.611	1777.043	0.9273675
	prob_min	prob_max	n_min	n_max	np_ratio	np_local			
1	1.0000000	1.0000000	1.0000000	1.0000000	58.00000	58.000000			
2	0.9693051	0.9752108	0.4965517	0.5034483	29.00000	32.000000			
3	0.8649222	0.9984559	0.1068966	0.4862069	19.33333	7.153846			

Here, we see that the 2-class solution is best according to the BIC, and that `np_local` is low for the 3-class solution. We can examine the results for the 2-class solution. Note that these results will now be a combination of means/variances and probabilities (thresholds):

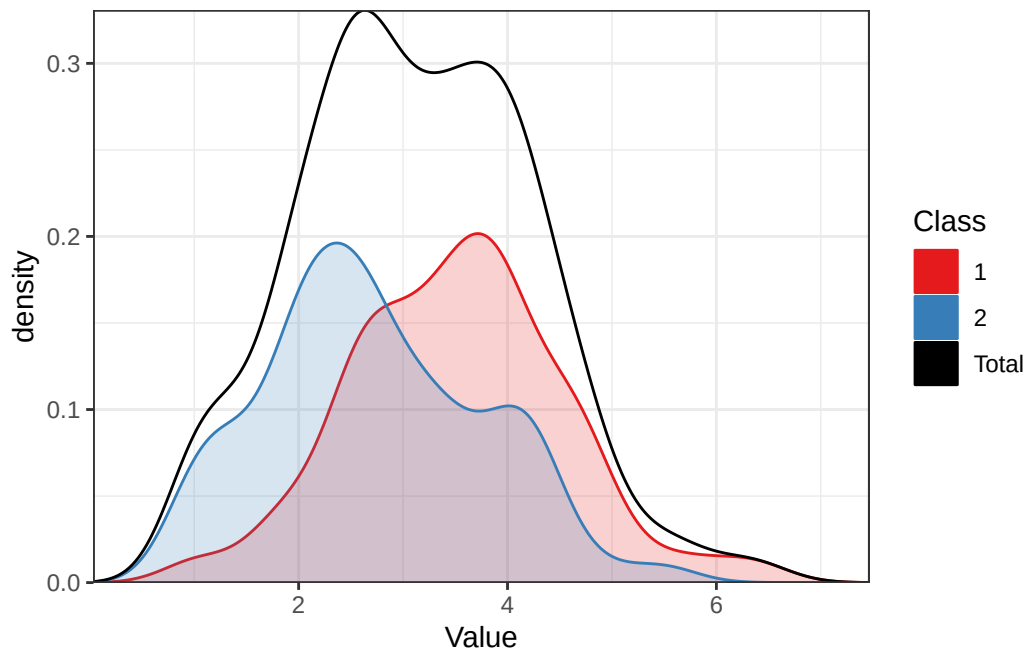
```
tab_mixed_res <- table_results(res_mixed[[2]])
tab_mixed_res
```

	label	est_sig	se	pval	confint	class
1	Means.CM.class1	3.52***	0.09	0.00	[3.35, 3.70]	class1
2	Variances.CM.class1	1.05***	0.09	0.00	[0.88, 1.23]	class1
3	Variances.AM1.class1	1.00	<NA>	<NA>	<NA>	class1
4	Variances.AM2.class1	1.00	<NA>	<NA>	<NA>	class1
5	Variances.AM3.class1	1.00	<NA>	<NA>	<NA>	class1
6	class1.Thresholds[1,1].class1	-1.60***	0.20	0.00	[-2.00, -1.21]	class1
7	class1.Thresholds[1,2].class1	-0.82***	0.13	0.00	[-1.08, -0.56]	class1
8	class1.Thresholds[1,3].class1	-1.38***	0.19	0.00	[-1.75, -1.02]	class1

9	Means.CM.class2	2.71***	0.09	0.00	[2.54, 2.89]	class2
10	Variances.AM1.class2	1.00	<NA>	<NA>	<NA>	class2
11	Variances.AM2.class2	1.00	<NA>	<NA>	<NA>	class2
12	Variances.AM3.class2	1.00	<NA>	<NA>	<NA>	class2
13	class2.Thresholds[1,1].class2	1.02***	0.15	0.00	[0.72, 1.32]	class2
14	class2.Thresholds[1,2].class2	1.86***	0.27	0.00	[1.33, 2.39]	class2
15	class2.Thresholds[1,3].class2	1.24***	0.16	0.00	[0.92, 1.56]	class2
16	equal var 2.weights[1,1].NA	1.00	<NA>	<NA>	<NA>	<NA>
17	equal var 2.weights[1,2].NA	0.97***	0.13	0.00	[0.72, 1.22]	<NA>

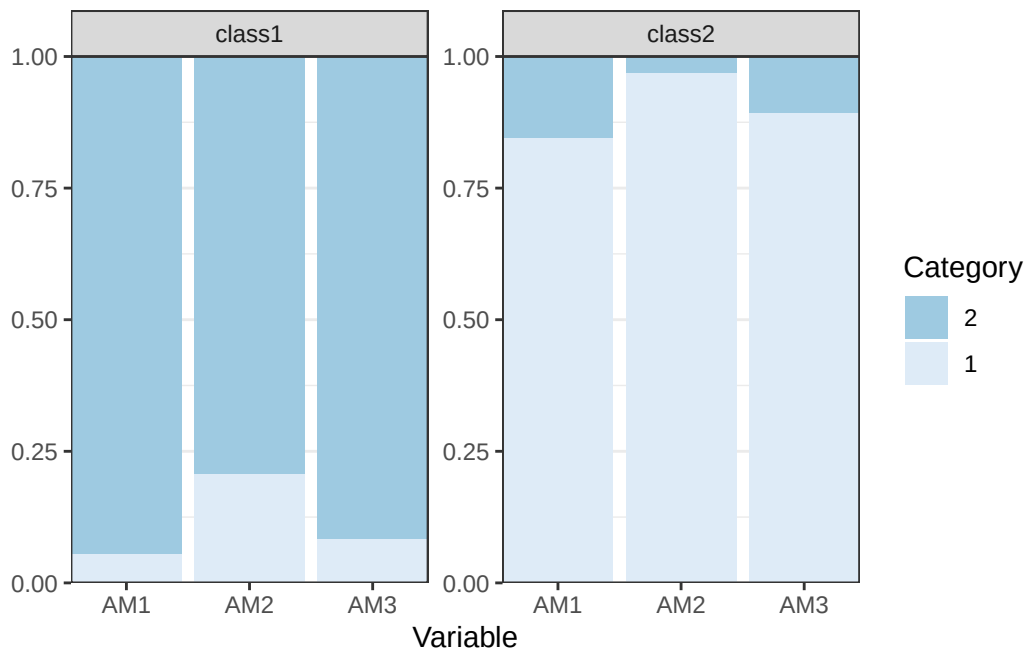
We can plot the continuous indicators:

```
plot_density(res_mixed[[2]], variables = "CM")
```



And the response probabilities for the binary indicators:

```
plot_prob(res_mixed[[2]])
```



The interpretation now depends on the indicator: describe class-specific means for continuous variables, response-category probabilities for ordinal variables, and endorsement probabilities for dichotomous variables.

1.10 Classification Accuracy Deep Dive

After estimating a mixture model, every participant has a vector of posterior class-membership probabilities (i.e., the probability that they belong to each class). The function `class_prob()` gives you this matrix, and several other diagnostic tables based on it:

```
cp <- class_prob(res_lpa[[3]])
format_numeric(head(cp$individual))
```

```
      class1 class2 class3 predicted
[1,] "0.00" "1.00" "0.00" "2.00"
[2,] "0.00" "0.00" "1.00" "3.00"
[3,] "0.99" "0.01" "0.00" "1.00"
[4,] "0.00" "1.00" "0.00" "2.00"
[5,] "0.76" "0.00" "0.24" "1.00"
[6,] "0.71" "0.29" "0.00" "1.00"
```

This table can be summarized in different ways to summarize the classifications, and to diagnose classification accuracy.

For example, `$sum.posterior` gives us the column sums of the individual classification probabilities; this indicates what proportion of your sample contributes to each class (and allows people to contribute fractionally to multiple classes):

```
format_numeric(head(cp$sum.posterior))
```

	class	count	proportion
1	class1	171.83	0.59
2	class2	88.42	0.30
3	class3	29.75	0.10

If you make a forced choice, assigning every person to the class for which they have the highest probability, you would get the following statistics. Note that these numbers now have measurement error baked into them:

```
format_numeric(head(cp$sum.mostlikely))
```

	class	count	proportion
1	class1	178	0.61
2	class2	84	0.29
3	class3	28	0.10

In Section 1.6.4, we already encountered the average classification probabilities for the most likely class membership, whose range is represented in the fit table as `prob_min` and `prob_max`. We can see the full matrix by calling:

```
format_numeric(cp$mostlikely.class)
```

	assigned.1	assigned.2	assigned.3
avgprob.1	"0.95"	"0.04"	"0.01"
avgprob.2	"0.13"	"0.87"	"0.00"
avgprob.3	"0.12"	"0.00"	"0.88"

On the diagonal, we find the average probability that observations belongs to the classes they are assigned to. If the classes are well-separated, you should see large diagonal values and small off-diagonal values. The separation here is OK; some diagonal values are not that high (.87-.88), and some off-diagonal values are non-negligible (.12-.13). For instance, you see that

the people who are assigned to class 1 have an average posterior probability of .95 for belonging to that class - which is very high. However, they also have non-zero probabilities of belonging to class 2 or 3.

Quiz

Q1: Person A has classification probabilities (.41, .34, .25). This person belongs to class:

...

Q2: Person B has classification probabilities (.92, .06, .02). Which person can be classified with greater certainty?

- A. Person A
- B. Person B

Q3: If you treat class membership as observed in follow-up analyses, your standard errors will be too small.

- ☐ True
- ☐ False

Q4: You can generally use most likely class membership as a predictor or covariate to perform follow-up analyses to compare classes, e.g., with ANOVA or regression.

- ☐ True
- ☐ False

Q5: If prob_min and prob_max are .99 and 1.00, it's fine to treat most likely class membership as an observed variable in follow-up analyses.

- ☐ True
- ☐ False

Answers

Q1: 1
Q2: Person B
Q3: TRUE
Q4: FALSE
Q5: TRUE

1.11 Reporting Results

Going back to the LPA example from Section 1.6, you might report the analysis as follows:

We conducted exploratory latent profile analysis using `tidySEM` version 0.2.12 and `OpenMx` version 2.22.11. We followed recommended practices by Van Lissa et al. (2024). The two continuous indicators were AM and CM. Model parameters were class-varying means and fixed variances across classes. We assumed conditional independence of these indicators within classes. Based on the available sample size, we considered a maximum of five classes. This model has 16 parameters, which allows for a maximum of about 20 observations per parameter. We selected the best-fitting model based on the BIC and significant BLRT test. We eliminated solutions with classes where the number of observations to parameters was smaller than 10. Based on these criteria, a 3-class solution was retained. This model had reasonable classification accuracy: entropy was .82, and the probability that observations in each class were correctly classified ranged from [.87, .95]. The smallest class contained 9.7% of participants. The model parameters are shown in Table 1, and are visually represented in Figure 1. The retained classes were characterized as [label 1], [label 2], and [label 3] based on their estimated [means / category-response probabilities / endorsement probabilities].

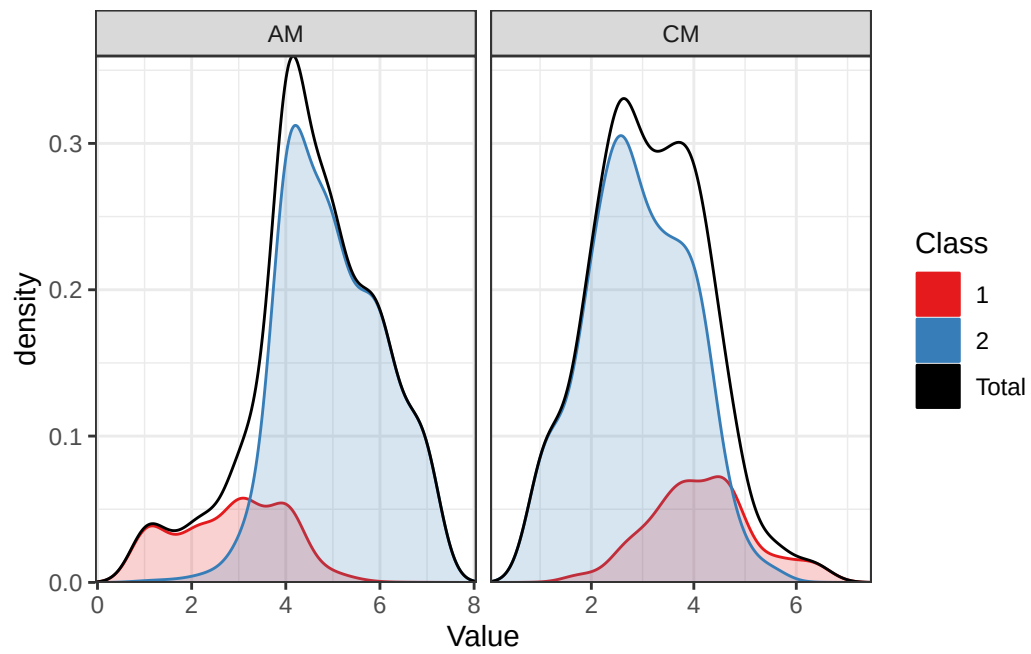
Here is the code to produce Table 1; we extract the means and paste the SE behind it (note that you can also use `est_sig` or another value column). We do not report the variances in this table, because they are fixed across classes. We also write it to a spreadsheet, so we can easily import it into other desktop publishing software:

```
tab <- table_results(res_lpa[[2]], columns = NULL)
tab <- tab[tab$Category == "Means", c("lhs", "class", "est", "se")]
tab$Mean <- glue::glue("{tab$est} ({tab$se})")
tab <- reshape(tab[, c("class", "lhs", "Mean")], direction = "wide", idvar = "class", timevar = "lhs")
write.csv(tab, "tab_res.csv", row.names = FALSE)
tab
```

	class	Mean AM	Mean CM
1	class1	2.83 (0.42)	4.14 (0.19)
5	class2	5.01 (0.11)	2.90 (0.10)

To produce Figure 1, use the following code. We're saving the file to SVG format, which is a (lossless) type of vector graphic. Other vector graphic types include PDF and EPS. Do not save to a raster graphic type (PNG, BMP, JPG), because these images will have low quality when reproduced in print or digital publication. The dimensions of the figure, in millimeter, correspond to A6 paper format:

```
p <- plot_density(res_lpa[[2]])
```



```
ggsave(filename = "figure1.svg", plot = p, width = 148, height = 105, units = "mm", dpi = 300)
```

2 Confirmatory Latent Class Analysis

Latent profile analysis (LPA) is often used exploratorily: researchers estimate several class solutions and then decide which one best describes the data. But LPA can also be used **confirmatorily**, when theory makes specific predictions about the latent class structure.

To follow along and run the code yourself, open the file `lca_confirmatory.qmd` in the `lca_course` project, as explained in Appendix [A](#).

In this first example, suppose that clinical psychological theory proposes the existence of two subpopulations:

- people **without depression**
- people **with depression**

If all we know is people's depression score, then we can try to identify two latent classes, corresponding to these subpopulations. Suppose that the theory further specifies that a depression score of about 10 or higher is required to count as depressed. We will use this theory to formulate two increasingly specific hypotheses:

1. a **two-class model** should fit better than a one-class model. Adding a third class should not significantly improve fit.
2. the class means should be close to **0** for the non-depressed class and **10** for the depressed class.

The second hypothesis is considerably stronger because it constrains the class-specific means to exact values.

2.1 Simulate example data

We first create a simple dataset containing a single continuous depression score.

```
library(tidySEM)
library(OpenMx)

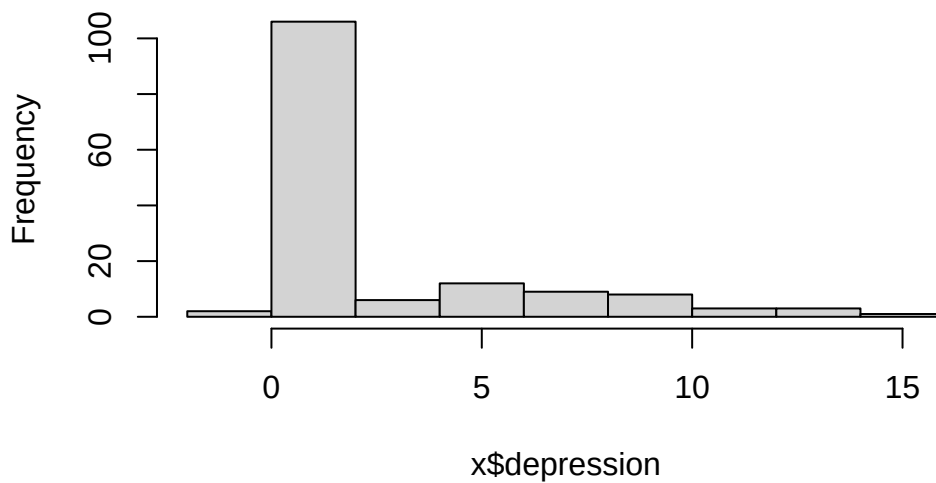
set.seed(123)
x <- data.frame(
```

```

depression = c(
  # The people with depression:
  rnorm(50, mean = 6, sd = 4),
  # The people without depression:
  rbeta(100, 2, 3)
)
hist(x$depression)

```

Histogram of x\$depression



2.2 Test the number of classes

The first confirmatory hypothesis concerns **class enumeration**. Theory predicts two classes, so we estimate one-, two-, and three-class solutions.

```

res <- mx_profiles(
  x,
  classes = 1:3
)
saveRDS(res, "res_depression.rds")

```

```
res <- readRDS("res_depression.rds")
table_fit(res)
```

	Name	Classes	LL	n	Parameters	AIC	BIC	saBIC	
1	equal	var 1	1	-398.3120	150	2	800.6240	806.6453	800.3156
2	equal	var 2	2	-346.5824	150	4	701.1647	713.2072	700.5480
3	equal	var 3	3	-314.9716	150	6	641.9432	660.0070	641.0182

	Entropy	prob_min	prob_max	n_min	n_max	np_ratio	np_local
1	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	75.0	75.00000
2	0.9495445	0.9432520	0.9953478	0.1733333	0.8266667	37.5	17.33333
3	0.9658533	0.9590919	0.9889837	0.1200000	0.7200000	25.0	13.50000

The fit table lets us inspect information criteria and other model-fit statistics across the competing class solutions.

If we want to test the theory, we can compare adjacent class solutions using the bootstrapped likelihood ratio test (BLRT).

```
res_blrt <- BLRT(res)
saveRDS(res_blrt, "blrt_depression.rds")
```

```
res_blrt <- readRDS("blrt_depression.rds")
res_blrt
```

Bootstrapped Likelihood Ratio Test:

	null	alt	lr	df	blrt_p	samples
	mix1	mix2	103.5	2	0	67
	mix2	mix3	63.2	2	0	91

Quiz

Q1: According to BIC, the best-fitting model is:

- A. 1 class
- B. 2 class
- C. 3 class

Q2: According to BLRT, the two-class model fits significantly better than the 1- and 3-class model.

☐ True

☐ False

Answers

Q1: 3 class
Q2: FALSE

2.3 Translate theoretical assumptions into parameter constraints

The models above still allow all parameters to be estimated freely from the data. That tests only the broad hypothesis that two groups exist. The groups themselves could diverge substantially from what was theorized.

Our theory posits that the non-depressed class should have a mean near 0, whereas the depressed class should have a mean near 10 (technically, the minimum value in that class should be 10, but that's a bit more difficult to enforce).

We can represent this stronger hypothesis by fixing the two class means to these values, or by e.g. setting a parameter constraint for the two classes.

First, create the two-class profile model without estimating it and examine its parameters:

```
res_theory <- mx_profiles(  
  x,  
  classes = 2,  
  run = FALSE  
)  
omxGetParameters(res_theory)
```

equal var 2.weights[1,2]	v1	m11
0.2605042	2.3038225	0.7251967
m21		
8.3585995		

The relevant mean parameters are m11 and m21. We can either fix these means to the exact values of 0 and 10:

```
res_theory <- omxSetParameters(  
  res_theory,  
  labels = c("m11", "m21"),  
  values = c(0, 10),
```

```

    free = FALSE
  )

```

Or alternatively, we could set a lower and upper bound for these parameters, thus allowing bounded free estimation:

```

res_theory_bounds <- omxSetParameters(
  res_theory,
  labels = c("m11", "m21"),
  values = c(0, 10),
  free = TRUE,
  lbound = c(NA, 10),
  ubound = c(1, NA)
)

```

Now, run your chosen model:

```

res_theory <- run_mx(res_theory)
res_theory_bounds <- run_mx(res_theory_bounds)

```

Next, compare the unconstrained two-class solution with the theory-constrained model.

```

compare_these <- list(
  res[[2]],
  res_theory,
  res_theory_bounds
)

table_fit(compare_these)

```

	Name	Classes	LL	n	Parameters	AIC	BIC	saBIC	Entropy
1	1	2	-346.5824	150	4	701.1647	713.2072	700.5480	0.9495445
2	2	2	-368.6431	150	2	741.2862	747.3075	740.9778	0.9488272
3	3	2	-350.6497	150	4	709.2994	721.3419	708.6827	0.9599385
	prob_min	prob_max	n_min	n_max	np_ratio	np_local			
1	0.9432520	0.9953478	0.1733333	0.8266667	37.5	17.33333			
2	0.9595115	0.9918444	0.1733333	0.8266667	75.0	52.00000			
3	0.9670867	0.9939682	0.1600000	0.8400000	37.5	16.00000			

We can also compare the models using the BLRT:

```
blrt_theory <- BLRT(list(res[[2]], res_theory))
saveRDS(blrt_theory, "blrt_theory.rds")
```

```
blrt_theory <- readRDS("blrt_theory.rds")
blrt_theory
```

Bootstrapped Likelihood Ratio Test:

```
      null      alt      lr df blrt_p samples
mix2 equal var 2 -44.1 -2   0.47      100
```

Note that fixing the class means to exact values is a very strong assumption. In many real applications, theory may imply that class means should be **near** certain values rather than exactly equal to them. You can also manually specify **start values**, and then test if the estimated **parameter values** deviate significantly from these:

```
res_theory_starts <- omxSetParameters(
  res_theory,
  labels = c("m11", "m21"),
  values = c(0, 10),
  free = TRUE
)
res_theory_starts <- run_mx(res_theory_starts)
omxGetParameters(res_theory_starts)
```

```
equal var 2.weights[1,2]          v1          m11
              0.2190895          2.4874123      0.8699903
              m21
              8.8424735
```

```
wald_test(res_theory_starts, "m11 = 0; m21 = 10")
```

NULL

Quiz

Q1: According to the BIC, which model is best?

- A. The free model
- B. The constrained model

C. The bounded model

Q2: According to the BLRT, we should prefer the constrained model to the free model.

- ☐ True
- ☐ False

Q3: According to the Wald test, the mean of class 2 is not significantly different from 10.

- ☐ True
- ☐ False

Answers

Q1: The free model

Q2: TRUE

Q3: FALSE

3 Caregiver Compass

For this next example, we will use simulated data based on work by Zegwaard and colleagues, who sought to establish a typology of caregivers who support a close other receiving outpatient psychological care. Qualitative research among experts resulted in a theory postulating the existence of four types of caregivers (translated from the original Dutch):

Balanced

The balanced caregiver experiences relative balance between the costs and benefits of caring for a close other.

Imbalanced

The imbalanced caregiver experiences a precarious balance between the costs and benefits of caring for a close other.

Lonely

The lonely caregiver experiences a strong sense of isolation.

Entrapped

The entrapped caregiver strongly feels a sense of being entangled in responsibilities which are difficult to fulfill.

The goal of this confirmatory study was to validate this hypothesized class solution in a sample of caregivers. A convenience sample was used, with no prior sample size justification. To view the data documentation, run the command `?tidySEM::zegwaard_carecompass` in the R console.

3.1 Loading the Data

To load the data, simply attach the `tidySEM` package. For convenience, we assign the variables used for analysis to an object called `df`. We first only use the four scales: `c("burdened", "trapped", "negaaffect", "loneliness")`.

```
# Load required packages
library(tidySEM)
library(ggplot2)
library(OpenMx)
library(scales)
library(mice)
# Load data
df <- zegwaard_carecompass[, c("burdened", "trapped", "negaaffect", "loneliness")]
```

3.2 Descriptive statistics

We use `tidySEM::descriptives()` to describe the data numerically. Because all scales are continuous, we select only columns for continuous data to de-clutter the table:

```
desc <- tidySEM::descriptives(df)
desc <- desc[, c("name", "n", "missing", "unique",
                 "mean", "median", "sd", "min", "max",
                 "skew_2se", "kurt_2se")]
desc
```

	name	n	missing	unique	mean	median	sd	min
1	burdened	509	0.007797271	509	3.383978	3.385765	0.7468695	1.1958284
2	trapped	505	0.015594542	505	1.725223	1.787354	0.8976391	-0.8578806
3	negaaffect	506	0.013645224	506	2.502401	2.519129	0.6901424	0.7086115
4	loneliness	510	0.005847953	510	2.658997	2.692829	0.6176745	0.9845240

	max	skew_2se	kurt_2se
1	5.282460	0.17234198	-0.4102692
2	3.805728	-1.03438503	-1.5170521
3	4.983359	0.08437372	-0.3823452
4	4.217298	-0.32661167	-0.6222047

The table indicates two potential causes for concern: there is a small percentage of missingness, and all variables have relatively high kurtosis. Since there are some missing values, we can conduct an MCAR test using `mice::mcar(df)`:

```
test_mcar <- mcar(df)
```

```
iter imp variable
```

1	1	burdened	trapped	negaaffect	loneliness
1	2	burdened	trapped	negaaffect	loneliness
1	3	burdened	trapped	negaaffect	loneliness
1	4	burdened	trapped	negaaffect	loneliness
1	5	burdened	trapped	negaaffect	loneliness
2	1	burdened	trapped	negaaffect	loneliness
2	2	burdened	trapped	negaaffect	loneliness
2	3	burdened	trapped	negaaffect	loneliness
2	4	burdened	trapped	negaaffect	loneliness
2	5	burdened	trapped	negaaffect	loneliness
3	1	burdened	trapped	negaaffect	loneliness
3	2	burdened	trapped	negaaffect	loneliness
3	3	burdened	trapped	negaaffect	loneliness
3	4	burdened	trapped	negaaffect	loneliness
3	5	burdened	trapped	negaaffect	loneliness
4	1	burdened	trapped	negaaffect	loneliness
4	2	burdened	trapped	negaaffect	loneliness
4	3	burdened	trapped	negaaffect	loneliness
4	4	burdened	trapped	negaaffect	loneliness
4	5	burdened	trapped	negaaffect	loneliness
5	1	burdened	trapped	negaaffect	loneliness
5	2	burdened	trapped	negaaffect	loneliness
5	3	burdened	trapped	negaaffect	loneliness
5	4	burdened	trapped	negaaffect	loneliness
5	5	burdened	trapped	negaaffect	loneliness

```
test_mcar
```

Missing data patterns: 3 used, 3 removed.

Cases used: 506

Hawkins' test: median χ^2 (6) = 4.369945, median p = 0.626746

Interpretation of results:

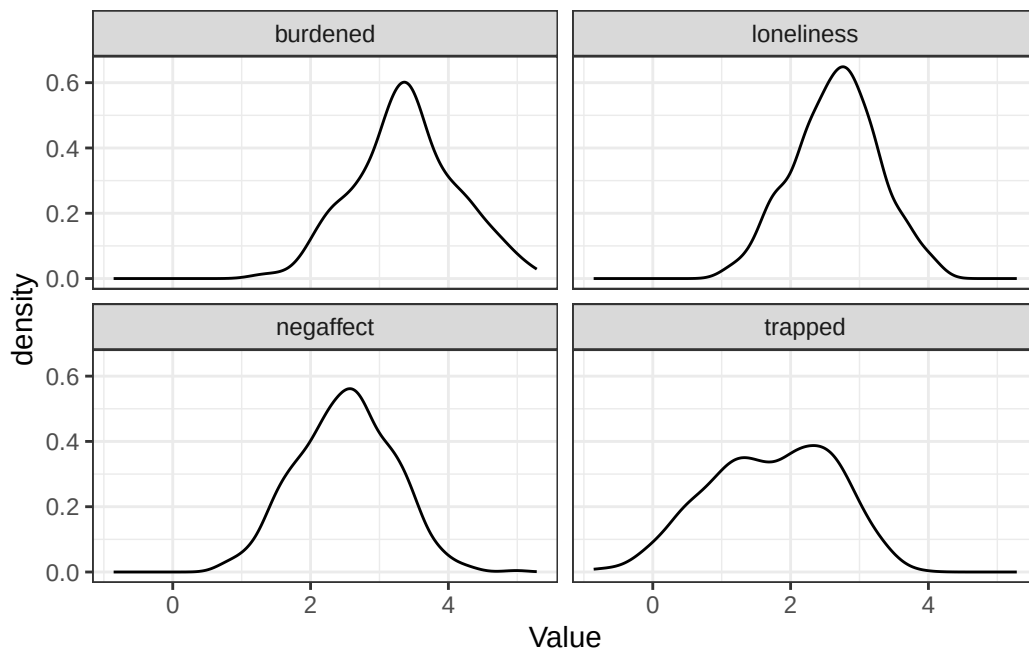
Hawkins' test is not significant; there is no evidence to reject the assumptions of multivariate normality and MCAR.

You should find that, according to Hawkins' test, there is no evidence to reject the assumptions of multivariate normality and MCAR. In `tidySEM`, missing data is accounted for using FIML by default, meaning that available data will be used without imputation. This is appropriate if data are missing completely at random, or if the missingness is contingent on the observed

variables - but not if missingness is contingent on the missing data (which is impossible to diagnose, and for which no adequate solution exists).

Additionally, we can plot the data. The `ggplot2` function `geom_density()` is useful for continuous data. Visual inspection confirms the conclusions from the `descriptives()` table: the data are kurtotic (peaked).

```
df_plot <- df
names(df_plot) <- paste0("Value.", names(df_plot))
df_plot <- reshape(df_plot, varying = names(df_plot), direction = "long",
                  timevar = "Variable")
ggplot(df_plot, aes(x = Value)) +
  geom_density() +
  facet_wrap(~Variable)+
  theme_bw()
```



3.3 Conducting Latent Profile Analysis

As all variables are continuous, we can use the convenience function `tidySEM::mx_profiles()`, which is a wrapper for the generic function `mx_mixture()` optimized for continuous indicators. Its default settings are appropriate for LPA, assuming fixed variances across classes and zero covariances. Its arguments are `data` and number of `classes`. All variables in `data` are included

in the analysis, which is why we first selected the indicator variables. As this is a confirmatory LCA, we do not follow a strictly data-driven class enumeration procedure. We will set the maximum number of classes K to one more than the theoretically expected number. We set a seed to ensure replicable results.

```
set.seed(123) # setting seed
res <- mx_profiles(data = df,
                  classes = 1:5)
saveRDS(res, "res_conf.rds")
```

```
res <- readRDS("res_conf.rds")
```

This analysis should produce some messages about cluster initialization. These relate to the selection of starting values, which relies on the K-means algorithm and is not robust to missing data. The algorithm automatically switches to hierarchical clustering, no further action is required.

3.4 Class Enumeration

To compare the fit of the theoretical model against other models, we create a model fit table using `table_fit()` and retain relevant columns. We also determine whether any models can be disqualified.

In this example, all models converge without issues. If, for example, the two-class solution had not converged, we could use the function `res[[2]] <- mxTryHard(res[[2]])` to aid convergence.

Next, we check for local identifiability. The sample size is consistently reported as 513, which means that partially missing cases were indeed included via FIML. The smallest class size occurs in the 5-class model, where the smallest class is assigned 7% of cases, or 38 cases. This model has 28 parameters, approximately 6 per class. We thus have at least five observations per parameter in every class, and do not disqualify the 5-class model.

There are concerns about theoretical interpretability of all solutions, as the entropies and minimum classification probabilities are all low. However, in this confirmatory use case, we address this when interpreting the results.

```
fit <- table_fit(res) # model fit table
fit[, c("Name", "LL", "Parameters", "n",
       "BIC", "Entropy",
       "prob_min", "prob_max",
       "n_min", "n_max",
       "np_ratio", "np_local")]
```

	Name	LL	Parameters	n	BIC	Entropy	prob_min	prob_max
1	equal var 1	-2241.982	8	513	4533.886	1.0000000	1.0000000	1.0000000
2	equal var 2	-2031.365	13	513	4143.854	0.7404912	0.9116044	0.9298845
3	equal var 3	-1951.349	18	513	4015.023	0.7797238	0.8860997	0.9146913
4	equal var 4	-1916.253	23	513	3976.033	0.7548658	0.8142116	0.9151926
5	equal var 5	-1912.370	28	513	3999.469	0.7878139	0.8109998	0.9155951
	n_min	n_max	np_ratio	np_local				
1	1.000000000	1.0000000	64.12500	64.1250000				
2	0.419103314	0.5808967	39.46154	35.8333333				
3	0.187134503	0.5438596	28.50000	18.0000000				
4	0.155945419	0.3430799	22.30435	16.0000000				
5	0.003898635	0.3411306	18.32143	0.4166667				

3.4.1 Using ICs

the 4-class solution has the lowest BIC, which means it is preferred over all other solutions including a 1-class solution and a solution with more classes. Note that a scree plot for the BIC can be plotted by calling `plot(fit)`. Following the elbow criterion, a three-class solution would also be defensible. The function `ic_weights(fit)` allows us to compute IC weights; it indicates that, conditional on the set of models, the 4-class model has a posterior model probability of nearly 100%.

3.4.2 Using LMR tests

Note that the use of LMR tests is discouraged; these tests make assumptions that are known to be violated for latent class analyses (Van Lissa et al., 2024). Nevertheless - if we do conduct LMR tests, we find that the tests are significant for all pairwise model comparisons, except for the 5-class model. Note that, as this analysis takes a long time, you don't have to run it - you can load its results from a file.

```
test_lmr <- lr_lmr(res)
saveRDS(test_lmr, "test_lmr.rds")
```

```
test_lmr <- readRDS("test_lmr.rds")
format_numeric(test_lmr)
```

Lo-Mendell-Rubin adjusted Likelihood Ratio Test:

```
null alt    lr df    p    w2 p_w2
mix1 mix2 10.25  5 0.00 0.82 0.00
```

```

mix2 mix3  5.30  5 0.00 0.44 0.00
mix3 mix4  4.14  5 0.00 0.14 0.00
mix4 mix5  0.88  5 0.19 0.04 0.00

```

3.4.3 Using BLRT tests

We can also use the BLRT test, which is the best practice solution for class enumeration, but is very computationally expensive. You can load the test results from a file, and we used a low number of replications to make it more feasible. In practice, one might use a much higher number (1000+) for published research. Keep in mind that the p-value of the BLRT is subject to Monte Carlo error; if it fluctuates when analyses are replicated or its value is very close to the critical threshold, consider increasing the number of replications.

To accelerate computations, we can use the `future` package for parallel computing (see `?plan` to select the appropriate back-end for your system). To track the function's progress, we use the `progressr` ecosystem, which allows users to choose how they want to be informed. The example below uses a progress bar:

```

library(future)
library(progressr)
plan(multisession) # Parallel processing for Windows
handlers("progress") # Progress bar
set.seed(1)
test_blrt <- BLRT(res, replications = 100)
saveRDS(test_blrt, "test_blrt.rds")

```

```

test_blrt <- readRDS("test_blrt.rds")
format_numeric(test_blrt)

```

Bootstrapped Likelihood Ratio Test:

	null	alt	lr	df	blrt_p	samples
mix1	mix2	421.23	5	0.00	100	
mix2	mix3	160.03	5	0.00	100	
mix3	mix4	70.19	5	0.00	100	
mix4	mix5	7.77	5	0.36	100	

In sum, across all class enumeration criteria, there is strong support for a 4-class solution.

3.5 Comparing Competing Theoretical Model Specifications

In the case of confirmatory LCA, the theory would be refuted by strong evidence against the hypothesized model and number of classes. In the preceding, we only compared the theoretical model against models with different number of classes. Imagine, however, that a Reviewer argues that variance ought to be freely estimated across classes. We could compare our theoretical model against their competing model as follows. Note that we can put two models into a list to compare them.

```
res_alt <- mx_profiles(df, classes = 4, variances = "varying")
saveRDS(res_alt, "res_alt.rds")
```

```
res_alt <- readRDS("res_alt.rds")
compare <- list(res[[4]], res_alt)
table_fit(compare)
```

	Name	Classes	LL	n	Parameters	AIC	BIC	saBIC	Entropy
1	1	4	-1916.253	513	23	3878.507	3976.033	3903.028	0.7548658
2	2	4	-1909.232	513	35	3888.464	4036.874	3925.778	0.7816344
	prob_min	prob_max	n_min	n_max	np_ratio	np_local			
1	0.8142116	0.9151926	0.1559454	0.3430799	22.30435	16.00			
2	0.8445960	0.9210385	0.1598441	0.3157895	14.65714	10.25			

The alternative model incurs 12 additional parameters for the free variances. Yet, it has a higher BIC, which indicates that this additional complexity does not outweigh the increase in fit.

You can obtain a significance test for this model comparison by using the BLRT:

```
test_compare <- BLRT(compare, replications = 100)
saveRDS(test_compare, "test_compare.rds")
```

```
test_compare <- readRDS("test_compare.rds")
test_compare
```

Bootstrapped Likelihood Ratio Test:

null	alt	lr	df	blrt_p	samples	
mix4	free var	4	14	12	0.39	100

3.6 Interpreting the Final Class Solution

To interpret the final class solution, we first reorder the 4-class model by class size. This helps prevent label switching.

```
res_final <- mx_switch_labels(res[[4]])
```

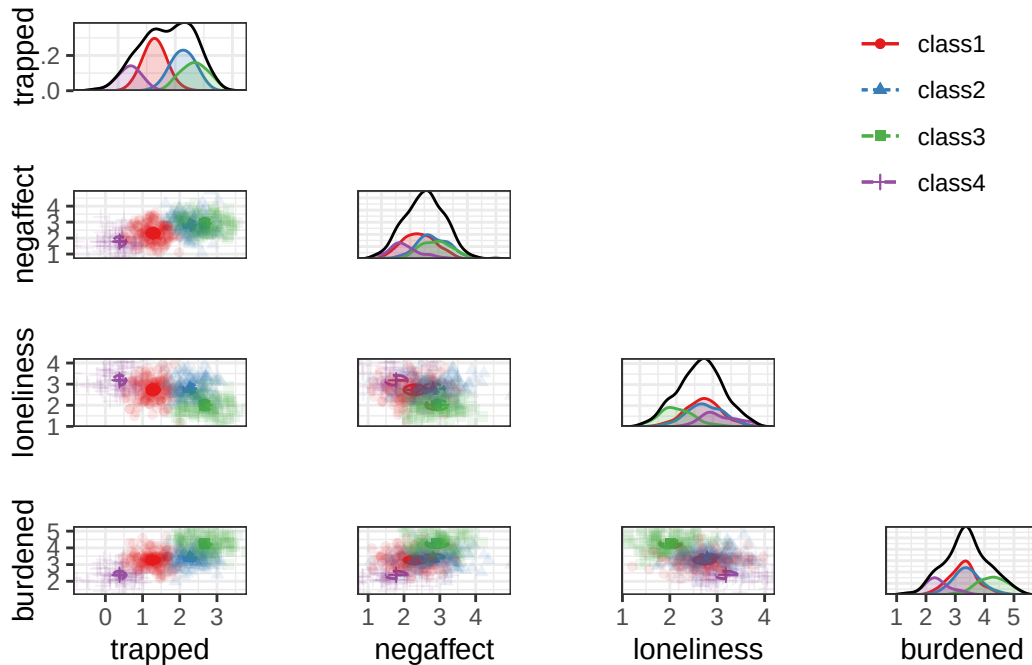
The 4-class model yielded classes of reasonable size; the largest class comprised 33%, and the smallest comprised 16% of cases. However, the entropy was low, $S = .75$, indicating poor class separability. Furthermore, the posterior classification probability ranged from $[\.81, .92]$, which means that at least some classes had a high classification error. We produce a table of the results below.

```
table_results(res_final, columns = c("label", "est", "se", "confint", "class"))
```

	label	est	se	confint	class
1	Means.burdened.class1	3.27	0.04	[3.18, 3.36]	class1
2	Means.trapped.class1	1.28	0.05	[1.18, 1.38]	class1
3	Means.negaaffect.class1	2.31	0.06	[2.20, 2.42]	class1
4	Means.loneliness.class1	2.73	0.04	[2.64, 2.82]	class1
5	Variances.burdened.class1	0.23	0.02	[0.19, 0.27]	class1
6	Variances.trapped.class1	0.17	0.02	[0.14, 0.20]	class1
7	Variances.negaaffect.class1	0.31	0.02	[0.27, 0.36]	class1
8	Variances.loneliness.class1	0.24	0.02	[0.20, 0.28]	class1
9	Means.burdened.class2	3.40	0.06	[3.28, 3.52]	class2
10	Means.trapped.class2	2.27	0.06	[2.15, 2.38]	class2
11	Means.negaaffect.class2	2.81	0.06	[2.70, 2.93]	class2
12	Means.loneliness.class2	2.79	0.06	[2.66, 2.91]	class2
13	Means.burdened.class3	4.25	0.07	[4.12, 4.38]	class3
14	Means.trapped.class3	2.67	0.05	[2.58, 2.77]	class3
15	Means.negaaffect.class3	2.92	0.06	[2.80, 3.03]	class3
16	Means.loneliness.class3	2.01	0.06	[1.89, 2.14]	class3
17	Means.burdened.class4	2.38	0.06	[2.26, 2.50]	class4
18	Means.trapped.class4	0.38	0.05	[0.28, 0.49]	class4
19	Means.negaaffect.class4	1.78	0.07	[1.65, 1.91]	class4
20	Means.loneliness.class4	3.18	0.06	[3.07, 3.30]	class4
21	mix4.weights[1,1].NA	1.00	<NA>	<NA>	<NA>
22	mix4.weights[1,2].NA	0.86	0.15	[0.56, 1.15]	<NA>
23	mix4.weights[1,3].NA	0.66	0.11	[0.44, 0.88]	<NA>
24	mix4.weights[1,4].NA	0.47	0.08	[0.32, 0.63]	<NA>

The results are best interpreted by examining a plot of the model and data, however. Relevant plot functions are `plot_bivariate()`, `plot_density()`, and `plot_profiles()`. However, we omit the density plots, because `plot_bivariate()` also includes them.

```
plot_bivariate(res_final)
```



On the diagonal of the bivariate plot are weighted density plots: normal approximations of the density function of observed data, weighed by class probability. On the off-diagonal are plots for each pair of indicators, with the class means indicated by a point, class standard deviations indicated by lines, and covariances indicated by circles. As this model has zero covariances, all circles are round (albeit warped by the different scales of the X and Y axes)

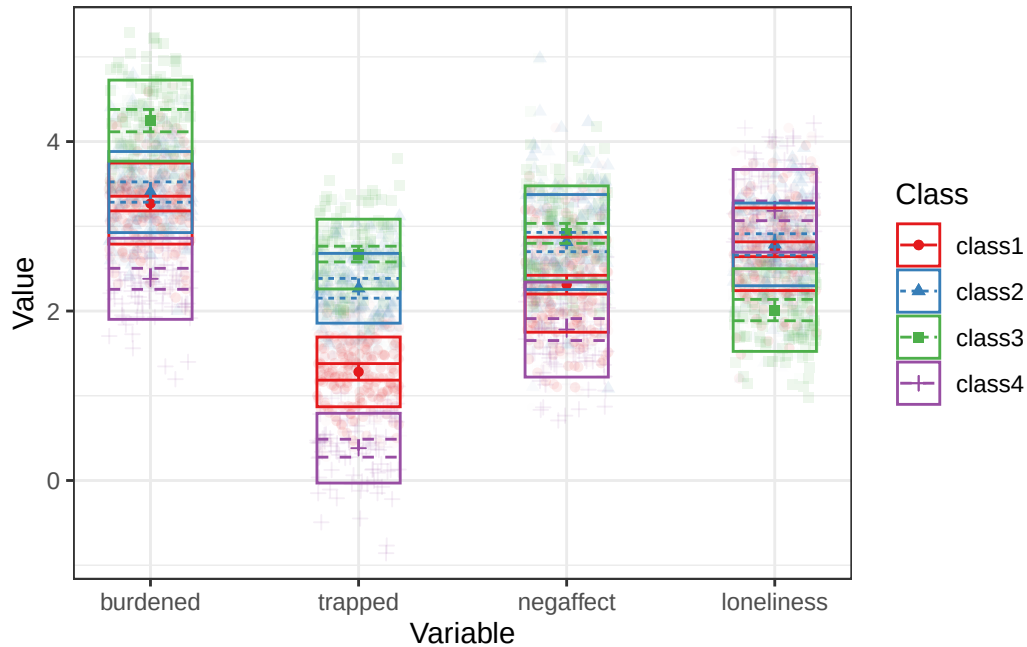
The marginal density plots show that trappedness distinguishes classes rather well. For all other indicators, groups are not always clearly separated in terms of marginal density: class 2 and 3 coalesce on negative affect, 1 and 2 coalesce on loneliness, and 1 and 2 coalesce on burden. Nevertheless, the off-diagonal scatterplots show reasonable bivariate separation for all classes.

We can obtain a more classic profile plot using `plot_profiles(res_final)`. This plot conveys less information than the bivariate plot, but is readily interpretable. Below is a comparison between the most common type of visualization for LPA, and the best-practices visualization provided by `tidySEM`. Note that the best practices plot includes class means and error bars, standard deviations, and a ribbon plot of raw data weighted by class probability to indicate how

well the classes describe the observed distribution. The overlap between the classes is clearly visible in this figure; this is why the entropy and classification probabilities are relatively low.

Based on the bivariate plot, we can label class 1 as the *balanced* type (33%), class 2 as the *imbalanced* type (29%), class 3 as the *entrapped* type (22%), and class 4 as the *lonely* type (16%). Note however that the observed classes do not match the hypothesized pattern of class parameters exactly.

```
plot_profiles(res_final)
```



We will save the final model for future reference:

```
saveRDS(res_final, "res_zegwaard.rds")
```

4 Auxiliary Variable Analyses

Latent class and latent profile models are often only the first step in an analysis. After identifying a useful class solution, researchers commonly want to know whether the classes differ on variables that were **not used to define the classes**. These variables are often called *auxiliary variables*, *distal outcomes*, or *external variables*.

In this vignette you will learn how to:

- distinguish proper auxiliary variable analysis from naive approaches assuming perfect class assignment;
- inspect classification uncertainty before conducting auxiliary analyses;
- use the Bolck-Croon-Hagenaars (BCH) method implemented in `tidySEM::BCH()`;
- use pseudo-class draws with `tidySEM::pseudo_class()` and `tidySEM::append_class_draws()`;
- test overall and pairwise class differences in BCH models using `tidySEM::lr_test()`;
- extract and report model parameters using `tidySEM::table_results()`; and
- communicate auxiliary-variable results without treating latent class membership as perfectly observed.

To follow along and run the code yourself, open the file `auxiliary_variables.qmd` in the `lca_course` project, as explained in Appendix [A](#).

4.1 Why auxiliary-variable analyses require special care

Suppose a latent profile model identifies three classes of students based on several indicators of study behavior. You may then want to ask whether the classes differ in academic achievement, well-being, or another external variable.

At face value, it seems sensible to assign each student to their most likely class, and then run an ANOVA or regression to examine class differences in academic achievement. The difficulty is that latent class membership is not directly observed, so there will be error/uncertainty in these class assignments. Treating the most likely class as if it were known without error discards this uncertainty, leading to biased estimates and standard errors in downstream analyses.

Proper methods for auxiliary variable analysis account for this classification uncertainty. This vignette focuses on two documented approaches:

1. **BCH analysis.** `BCH()` estimates an auxiliary model as a multi-group model. Every case contributes to every class-specific group, with contributions determined by group-specific BCH weights.
2. **Pseudo-class analysis.** `pseudo_class()` repeatedly draws a class value for each person according to that person's estimated class-membership probabilities, estimates the auxiliary model in each resulting dataset, and pools the results across datasets.

4.2 Example Data and Model

This vignette assumes that you have already selected and interpreted a latent class solution estimated with a `tidySEM` mixture-model function such as `mx_profiles()`, `mx_lca()`, or `mx_mixed_lca()`, see Chapter 1. You can use your own, or follow along with the examples below using the `zegwaard_carecompass` data included with `tidySEM`, see Chapter 2. We can load the final model constructed in that previous chapter from a file. This gives us four (theoretically expected) profiles of caregivers of close others (e.g., the elderly, or those suffering from chronic illness). The profiles are constructed based on self-reported burden, feelings of being trapped, negative affect, and loneliness. The dataset contains several other properties of the caregiver and the person being cared for, which we could analyze as auxiliary variables:

```
library(tidySEM)
library(OpenMx)
lpa_model <- readRDS("res_zegwaard.rds")
df <- zegwaard_carecompass
names(df)
```

```
[1] "burdened"      "trapped"       "negaaffect"    "loneliness"    "sex"
[6] "sexpatient"    "cohabiting"    "distance"      "freqvisit"     "relationship"
```

First examine the latent profile model.

```
table_fit(lpa_model)
```

	Minus2LogLikelihood	n	Parameters	observedStatistics	df	RMSEASquared		
1	3832.507	513	23	2030	2007	0		
	RMSEANull	modelName	AIC	BIC	saBIC	Classes	Entropy	prob_min
1	0.05	mix4	3878.507	3976.033	3903.028	4	0.75487	0.8142145
	prob_max	n_min	n_max	LL				
1	0.9151939	0.1559454	0.3430799	-1916.253				

```
table_results(lpa_model)
```

	label	est_sig	se	pval	confint	class
1	Means.burdened.class1	3.27***	0.04	0.00	[3.18, 3.36]	class1
2	Means.trapped.class1	1.28***	0.05	0.00	[1.18, 1.38]	class1
3	Means.negaaffect.class1	2.31***	0.06	0.00	[2.20, 2.42]	class1
4	Means.loneliness.class1	2.73***	0.04	0.00	[2.64, 2.82]	class1
5	Variances.burdened.class1	0.23***	0.02	0.00	[0.19, 0.27]	class1
6	Variances.trapped.class1	0.17***	0.02	0.00	[0.14, 0.20]	class1
7	Variances.negaaffect.class1	0.31***	0.02	0.00	[0.27, 0.36]	class1
8	Variances.loneliness.class1	0.24***	0.02	0.00	[0.20, 0.28]	class1
9	Means.burdened.class2	3.40***	0.06	0.00	[3.28, 3.52]	class2
10	Means.trapped.class2	2.27***	0.06	0.00	[2.15, 2.38]	class2
11	Means.negaaffect.class2	2.81***	0.06	0.00	[2.70, 2.93]	class2
12	Means.loneliness.class2	2.79***	0.06	0.00	[2.66, 2.91]	class2
13	Means.burdened.class3	4.25***	0.07	0.00	[4.12, 4.38]	class3
14	Means.trapped.class3	2.67***	0.05	0.00	[2.58, 2.77]	class3
15	Means.negaaffect.class3	2.92***	0.06	0.00	[2.80, 3.03]	class3
16	Means.loneliness.class3	2.01***	0.06	0.00	[1.89, 2.14]	class3
17	Means.burdened.class4	2.38***	0.06	0.00	[2.26, 2.50]	class4
18	Means.trapped.class4	0.38***	0.05	0.00	[0.28, 0.49]	class4
19	Means.negaaffect.class4	1.78***	0.07	0.00	[1.65, 1.91]	class4
20	Means.loneliness.class4	3.18***	0.06	0.00	[3.07, 3.30]	class4
21	mix4.weights[1,1].NA	1.00	<NA>	<NA>	<NA>	<NA>
22	mix4.weights[1,2].NA	0.86***	0.15	0.00	[0.56, 1.15]	<NA>
23	mix4.weights[1,3].NA	0.66***	0.11	0.00	[0.44, 0.88]	<NA>
24	mix4.weights[1,4].NA	0.47***	0.08	0.00	[0.32, 0.63]	<NA>

Quiz

Q1: How many classes does the model contain?

...

Q2: The class-specific model parameters include means and variances.

- ☐ True
- ☐ False

Q3: Auxiliary variables influence the parameter estimates of the class solution

- ☐ True

☐ False

Q4: The fit table suggests that class separation is:

- A. Poor
- B. Excellent
- C. Questionable

Answers

Q1: 4
Q2: FALSE
Q3: FALSE
Q4: Questionable

4.3 Inspect Classification Uncertainty

Before testing external differences, inspect how clearly observations are classified. Note that the fit table already gave us two metrics: entropy, and the range of the probabilities that observations in each class will be correctly classified (`prob_min` and `prob_max`). Note that these are all reasonably low, indicating poor classification accuracy. Thus, this is a prime example of an analysis where it's important to account for classification uncertainty! If entropy and `prob_min` and `prob_max` were all near 1, you could treat class membership as observed without introducing substantial bias, but that is not the case here.

The function `class_prob()` returns posterior classification probabilities for each individual, and several summaries and diagnostic statistics based on those:

```
cp <- class_prob(lpa_model)
```

Examine the individual posterior probabilities:

```
format_numeric(head(cp$individual))
```

```
      class1 class2 class3 class4 predicted
[1,] "0.87" "0.13" "0.00" "0.00" "1.00"
[2,] "0.28" "0.00" "0.00" "0.72" "4.00"
[3,] "0.81" "0.19" "0.00" "0.00" "1.00"
[4,] "0.96" "0.02" "0.00" "0.02" "1.00"
[5,] "0.92" "0.07" "0.00" "0.01" "1.00"
[6,] "0.01" "0.05" "0.93" "0.00" "3.00"
```

These probabilities make clear why an auxiliary analysis should not normally be based only on a hard class assignment. Whereas you might be relatively certain that person 4 belongs in class 1, there is substantial uncertainty about person 2, for example.

If you perform auxiliary analyses, the basic intuition is that each person contributes to estimating the parameters (e.g., auxiliary variable means) in each class in proportion to their posterior classification probability of belonging to that class. Thus, person 1 will contribute most to class 1, but also a little bit to class 2.

4.4 The BCH method

The BCH method performs a weighted multi-group analysis. All cases contribute to all groups, but their contributions are weighted by the most likely class membership matrix:

```
format_numeric(cp$mostlikely.class)
```

	assigned.1	assigned.2	assigned.3	assigned.4
avgprob.1	"0.88"	"0.08"	"0.00"	"0.03"
avgprob.2	"0.11"	"0.81"	"0.07"	"0.00"
avgprob.3	"0.01"	"0.13"	"0.86"	"0.00"
avgprob.4	"0.08"	"0.00"	"0.00"	"0.92"

4.4.1 Comparing class means on a continuous outcome

If the research question is simply whether latent classes differ in the mean of a continuous external variable, for example, to test whether the classes differ in their distance to the caree, we can call the BCH function and simply supply the auxiliary variable to the `data` argument, omitting the `model` argument:

```
bch_mean <- BCH(x = lpa_model,
                data = df$distance)
table_results(bch_mean)
```

	label	est_sig	se	pval	confint	group
1	Means.y.class1	155.25***	3.80	0.00	[147.80, 162.69]	class1
2	Variances.y.class1	2464.08***	266.68	0.00	[1941.39, 2986.77]	class1
3	Means.y.class2	159.52***	2.79	0.00	[154.05, 164.98]	class2
4	Variances.y.class2	1144.27***	133.43	0.00	[882.76, 1405.78]	class2
5	Means.y.class3	147.24***	6.09	0.00	[135.30, 159.18]	class3
6	Variances.y.class3	4200.66***	558.51	0.00	[3105.99, 5295.32]	class3

```

7      Means.y.class4 167.01*** 7.06 0.00 [153.17, 180.84] class4
8 Variances.y.class4 3989.68*** 630.60 0.00 [2753.71, 5225.64] class4

```

Quiz

Q1: Which class, on average, lives closest to the caree?

- A. Class 3
- B. Class 4
- C. Class 2
- D. Class 1

Q2: The parameters in this model are:

- A. Means and variances
- B. Means, variances, covariances
- C. Means
- D. Means and mixture proportions

Answers

Q1: Class 3
Q2: Means and variances

A BCH analysis is a multi-group OpenMx model but not a **test** of differences between groups. The function `lr_test()` can be used to compare parameters across groups. The function conducts overall and pairwise likelihood-ratio tests for a multi-group `MxModel` whose submodels have the same structure:

```
lr_test(bch_mean, compare = "M")
```

BCH test for equality across classes

Overall likelihood ratio test:

LL_baseline	LL_restricted	LL_dif	df	p
5426.959	5432.425	5.465735	3	0.1407025

Pairwise comparisons using likelihood ratio tests:

Model1	Model2	LL_baseline	LL_restricted	LL_dif	df	p
class1	class2	5426.959	5427.779	0.8197093	1	0.36526517
class1	class3	5426.959	5428.203	1.2440167	1	0.26469835
class2	class3	5426.959	5430.281	3.3217853	1	0.06836755
class1	class4	5426.959	5429.092	2.1334872	1	0.14411257

class2	class4	5426.959	5427.933	0.9735965	1	0.32378486
class3	class4	5426.959	5431.396	4.4373903	1	0.03515995

4.4.2 Comparing Proportions Across Classes

For a single ordinal variable, like the sex of patients, we could call:

```
aux_sex <- BCH(lpa_model, data = ordered(zegwaard_carecompass$sepatient, levels = c("male",
```

To obtain an omnibus likelihood ratio test of the significance of these sex differences across classes, as well as pairwise comparisons between classes, use:

```
lr_test(aux_sex)
```

BCH test for equality across classes

Overall likelihood ratio test:

LL_baseline	LL_restricted	LL_dif	df	p
695.2568	703.954	8.697276	3	0.03359867

Pairwise comparisons using likelihood ratio tests:

Model1	Model2	LL_baseline	LL_restricted	LL_dif	df	p
class1	class2	695.2568	695.7158	0.4590770	1	0.498055675
class1	class3	695.2568	700.2993	5.0425058	1	0.024732690
class2	class3	695.2568	703.0550	7.7982664	1	0.005229638
class1	class4	695.2568	695.9637	0.7068917	1	0.400477634
class2	class4	695.2568	697.1501	1.8933247	1	0.168827407
class3	class4	695.2568	696.4700	1.2132661	1	0.270686249

The results can be reported in probability scale:

```
table_prob(aux_sex)
```

	Variable	Category	Probability	group
1	y	1	0.4596910	class1
2	y	2	0.5403090	class1
3	y	1	0.4216089	class2
4	y	2	0.5783911	class2
5	y	1	0.5959575	class3
6	y	2	0.4040425	class3
7	y	1	0.5164841	class4
8	y	2	0.4835159	class4

Quiz

Q1: There are significant sex differences across classes.

- ☐ True
- ☐ False

Q2: Which class differs significantly from class 1?

- A. class 4
- B. class 3
- C. class 2

Q3: Which class cares for the largest proportion of female patients?

- A. class 4
- B. class 2
- C. class 1
- D. class 3

Answers

Q1: TRUE
Q2: class 3
Q3: class 2

4.4.3 Comparing Auxiliary Models Across Classes

We can also compare a simple model between classes. For example, we previously compared mean distances across classes - but this model also left variances free across classes. If we want to constrain those, we could use the following code:

```
df_aux <- zegwaard_carecompass[, "distance", drop = FALSE]
aux_mean <- BCH(lpa_model, model = "distance~1; distance ~~c*distance",
               data = df_aux)
table_results(aux_mean)
```

	label	est_sig	se	pval	confint	group
1	Means.distance.class1	155.25***	3.98	0.00	[147.44, 163.05]	class1
2	Variances.distance.class1	2707.70***	169.39	0.00	[2375.70, 3039.70]	class1
3	Means.distance.class2	159.52***	4.29	0.00	[151.11, 167.93]	class2
4	Means.distance.class3	147.24***	4.89	0.00	[137.65, 156.83]	class3
5	Means.distance.class4	167.01***	5.82	0.00	[155.61, 178.40]	class4

For a slightly more complex model, let's examine whether the distance predicts the frequency of visits differently across classes (treated as continuous).

```
df_aux <- zegwaard_carecompass[, c("freqvisit", "distance")]
df_aux$freqvisit <- as.numeric(df_aux$freqvisit)
aux_model <- BCH(lpa_model, model = "freqvisit ~ distance",
                 data = df_aux)
table_results(aux_model)
```

	label	est_sig	se	pval
1	Regressions.freqvisit.ON.distance.class1	0.00	0.00	0.80
2	Means.freqvisit.class1	3.99***	0.18	0.00
3	Means.distance.class1	155.25***	3.80	0.00
4	Variances.freqvisit.class1	0.53***	0.06	0.00
5	Variances.distance.class1	2464.07***	266.70	0.00
6	Regressions.freqvisit.ON.distance.class2	0.00	0.00	0.77
7	Means.freqvisit.class2	3.66***	0.43	0.00
8	Means.distance.class2	159.52***	2.79	0.00
9	Variances.freqvisit.class2	1.19***	0.14	0.00
10	Variances.distance.class2	1144.27***	133.42	0.00
11	Regressions.freqvisit.ON.distance.class3	-0.00	0.00	0.35
12	Means.freqvisit.class3	3.95***	0.27	0.00
13	Means.distance.class3	147.24***	6.09	0.00
14	Variances.freqvisit.class3	1.29***	0.17	0.00
15	Variances.distance.class3	4200.66***	558.63	0.00
16	Regressions.freqvisit.ON.distance.class4	-0.00	0.00	0.91
17	Means.freqvisit.class4	3.32***	0.38	0.00
18	Means.distance.class4	167.02***	7.06	0.00
19	Variances.freqvisit.class4	1.48***	0.23	0.00
20	Variances.distance.class4	3989.68***	630.52	0.00

	confint	group
1	[-0.00, 0.00]	class1
2	[3.64, 4.35]	class1
3	[147.80, 162.70]	class1
4	[0.42, 0.64]	class1
5	[1941.34, 2986.79]	class1
6	[-0.00, 0.01]	class2
7	[2.81, 4.51]	class2
8	[154.05, 164.98]	class2
9	[0.92, 1.46]	class2
10	[882.77, 1405.76]	class2
11	[-0.00, 0.00]	class3

```

12      [3.43, 4.47] class3
13 [135.30, 159.18] class3
14      [0.95, 1.63] class3
15 [3105.75, 5295.56] class3
16      [-0.00, 0.00] class4
17      [2.57, 4.08] class4
18 [153.18, 180.86] class4
19      [1.02, 1.93] class4
20 [2753.87, 5225.48] class4

```

To obtain an omnibus likelihood ratio test of the difference in regression coefficients across classes and pairwise comparisons between classes, use:

```
lr_test(aux_model, compare = "A")
```

BCH test for equality across classes

Overall likelihood ratio test:

LL_baseline	LL_restricted	LL_dif	df	p
6852.492	6853.473	0.9812781	3	0.805782

Pairwise comparisons using likelihood ratio tests:

Model1	Model2	LL_baseline	LL_restricted	LL_dif	df	p
class1	class2	6852.492	6852.521	0.02932549	1	0.8640297
class1	class3	6852.492	6853.328	0.83645365	1	0.3604130
class2	class3	6852.492	6853.043	0.55116219	1	0.4578432
class1	class4	6852.492	6852.540	0.04766352	1	0.8271800
class2	class4	6852.492	6852.582	0.08964307	1	0.7646314
class3	class4	6852.492	6852.721	0.22892426	1	0.6323226

The results indicate that there are no significant sex differences across classes, $LL(3) = 0.98, p = .81$.

4.5 Pseudo-class analysis

The pseudo-class approach represents classification uncertainty through repeated stochastic class assignments. For each generated dataset, a person's `class` value is drawn according to that person's posterior probability of belonging to each latent class. The auxiliary model is then fitted in each dataset, and the estimates are pooled across datasets.

For example, we can regress the external outcome on pseudo-class membership (treated as a factor via `I(factor())`), and dropping the intercept via `-1`:

```
pseudo_mean <- pseudo_class(
  x = lpa_model,
  model = lm(distance ~ I(factor(class)) -1, data = data),
  data = df[, "distance", drop = FALSE],
  m = 100)
pseudo_mean
```

	term	estimate	std.error	statistic	df	p.value
1	I(factor(class))1	155.8461	4.311103	36.14995	402.2322	1.997660e-128
2	I(factor(class))2	157.9415	4.864065	32.47109	338.1576	5.845376e-106
3	I(factor(class))3	149.5105	5.418970	27.59021	369.5279	9.118843e-92
4	I(factor(class))4	165.4974	6.334816	26.12505	398.4242	2.243163e-88

Note that the results are very similar, but not identical, to those of the BCH model with fixed variances:

```
tab_auxmean <- table_results(aux_mean, columns = NULL)
tab_auxmean[tab_auxmean$Category == "Means", c("est", "se")]
```

	est	se
1	155.25	3.98
3	159.52	4.29
5	147.24	4.89
7	167.01	5.82

The pooled coefficients summarize the estimated relationship between pseudo-class membership and the external outcome while incorporating the uncertainty produced by repeatedly drawing class membership.

The interpretation of regression coefficients depends on how the generated `class` variable is parameterized in the analysis. See `?pseudo_class` for more examples.

4.5.1 Using OpenMx Models

The `pseudo_class()` function is flexible about how the auxiliary model is supplied. The manual documents that `model` may be a model expression, a function that performs the analysis on each generated dataset, or a character string that can be interpreted as a structural equation model through `as_ram()`.

If this structural equation model does not explicitly refer to the `class` variable, it will be used to construct a multi-group model:

```
pct_mx <- pseudo_class(
  x = lpa_model,
  model = "distance ~ 1",
  data = df_aux,
  m = 20
)
pct_mx[grepl("^Mean", pct_mx$term), ]
```

	term	estimate	se	statistic	df	df_complete
1	Means.distance.1	155.6502	4.108376	37.88607	265.80708	505
3	Means.distance.2	157.8605	4.591046	34.38443	82.80627	505
5	Means.distance.3	149.6649	6.273283	23.85752	217.23762	505
7	Means.distance.4	165.7177	7.173601	23.10105	426.02956	505

	pvalue
1	3.817649e-109
3	8.553351e-51
5	1.303326e-62
7	3.848777e-77

Note that the results are, again, very close to the BCH analysis.

4.6 Generating pseudo-class draws explicitly

Sometimes you may want access to the generated pseudo-class datasets themselves. `append_class_draws()` generates multiple datasets containing a variable named `class`. Each person's class value is sampled according to their posterior class-membership probabilities.

```
draws <- append_class_draws(
  lpa_model,
  data = df_aux,
  m = 20
)
head(draws)
```

	id_dataset	freqvisit	distance	class
1	1	3	0.000000	2

2	1	4	225.103494	4
3	1	5	115.505252	1
4	1	6	104.147144	2
5	1	3	65.699952	1
6	1	5	5.915863	3

The returned object has class `class_draws` and can itself be supplied to workflows that support pseudo-class analyses.

A crucial practical requirement is that the rows of the auxiliary dataset must be in **exactly the same order** as the observations used to estimate the latent class model. The class probabilities belong to specific individuals; reordering the auxiliary data would associate probabilities with the wrong cases.

This is a good reason to prepare the indicator and auxiliary datasets together before model fitting, rather than independently sorting or filtering them later.

4.7 BCH or Pseudo-class?

Both methods are designed to avoid treating modal class membership as perfectly observed, but they do so differently.

The BCH method expresses the auxiliary analysis as a weighted multi-group model. This is particularly convenient when the external question can be represented as an OpenMx/RAM model and when you want direct class-specific estimates plus likelihood-ratio comparisons across groups. Each case contributes proportionally to all groups' estimates.

The pseudo-class method instead creates repeated plausible class assignments and pools the resulting analyses. Its main practical advantage is flexibility: the manual allows the `model` argument to be an expression, function, or structural-model syntax. This can make it useful for auxiliary analyses that are naturally handled by familiar R modeling functions. Each case contributes to one group's estimates only, which introduces bias; pooling the results across multiple sets of class draws averages out this bias, but introduces Monte Carlo error.

The choice between them should therefore be guided by the inferential question and the model you want to fit, rather than by which method gives the most favorable p-value.

4.8 Predicting class membership

So far, we have asked whether latent classes differ on auxiliary variables. We can also reverse this question: **Which variables predict class membership?**

In principle, a test of class differences on an auxiliary variable also provides evidence about the association between that variable and class membership. If an auxiliary variable is significantly associated with class membership, this association will be evident when tested in either direction, whether `class -> auxiliary` or `auxiliary -> class`. The statistical parameters and their interpretation differ, but the significance test is informative. The interpretation of this significant effect is determined by the assumed direction of causality. For example, if age is assumed to be a predictor of class membership, then a significant BCH analysis of class differences in age tells you that the reverse effect is also significant.

In some cases, researchers might prefer to model class membership as the outcome and age as the predictor. With `tidySEM`, this type of analysis can conveniently be conducted using `pseudo_class()`. For example, we can use the `nnet` package to perform a multinomial regression, predicting class membership while accounting for uncertainty in class assignment.

```
library(nnet)
pct_pred <- pseudo_class(x = lpa_model,
  model = function(data){
    multinom(class ~ freqvisit + distance, data = data)
  },
  data = df_aux,
  m = 10)
```

```
# weights:  16 (9 variable)
initial  value 704.237535
iter  10 value 675.522391
final   value 675.274240
converged
# weights:  16 (9 variable)
initial  value 704.237535
iter  10 value 674.743691
final   value 674.644298
converged
# weights:  16 (9 variable)
initial  value 704.237535
iter  10 value 674.260378
final   value 674.111698
converged
# weights:  16 (9 variable)
initial  value 704.237535
iter  10 value 677.891825
final   value 676.649974
converged
# weights:  16 (9 variable)
```

```

initial value 704.237535
iter 10 value 679.649226
final value 679.072486
converged
# weights: 16 (9 variable)
initial value 704.237535
iter 10 value 669.617188
final value 669.166829
converged
# weights: 16 (9 variable)
initial value 704.237535
iter 10 value 675.253480
final value 674.958769
converged
# weights: 16 (9 variable)
initial value 704.237535
iter 10 value 677.425213
final value 677.211155
converged
# weights: 16 (9 variable)
initial value 704.237535
iter 10 value 680.157095
final value 680.011250
converged
# weights: 16 (9 variable)
initial value 704.237535
iter 10 value 674.809898
final value 674.783535
converged

```

pct_pred

	term	estimate	std.error	statistic	df	p.value
1	(Intercept)	0.1731164002	0.646520772	0.2677662	128.26881	0.7893093996
2	freqvisit	-0.1255594817	0.118622975	-1.0584752	263.05981	0.2908098026
3	distance	0.0009013541	0.002650792	0.3400320	69.25029	0.7348616377
4	(Intercept)	0.8154864556	0.629977874	1.2944684	273.01742	0.1965974506
5	freqvisit	-0.2139114828	0.128083447	-1.6700947	201.67747	0.0964519072
6	distance	-0.0025474514	0.002543961	-1.0013719	178.04853	0.3180059852
7	(Intercept)	0.4991259554	0.727181500	0.6863843	176.87877	0.4933695984
8	freqvisit	-0.4803074023	0.141265734	-3.4000277	265.52875	0.0007774103
9	distance	0.0031882574	0.002970690	1.0732381	147.56960	0.2849161667

Note that the results are a bit messy, but essentially, we get four sets of regression coefficients, predicting membership of classes 1:4. In this case, only `freqvisit` is a significant predictor of membership of class 4, nothing else.

An important advantage of this approach is that several predictors can be included simultaneously. Their coefficients then represent **partial associations** with class membership: the association of each predictor with class membership while holding the other predictors constant.

This is a real difference from testing class differences separately for each auxiliary variable via the BCH. For example, testing whether classes differ in age can establish a univariate association between age and class membership. However, a multinomial regression can address a more specific question: *Does age predict class membership after accounting for sex and education?*

4.9 Forecasting Class for New Cases

This LCA model was developed to help classify care providers in a clinical context, so that mental healthcare professionals can provide tailored support to those who take care of their clients. In `tidySEM`, it is possible to predict class membership for new data. Imagine that we administer the care compass questionnaire to a new individual. We can assign their scale scores to a `data.frame`, and supply it to the `predict_class()` function (in previous versions, we overloaded the `predict()` function) via the `newdata` argument. The result includes the individual's most likely class, as well as posterior probabilities for all classes.

```
df_new <- data.frame(  
  burdened = 2,  
  trapped = 0.5,  
  negaffect = 1.5,  
  loneliness = 4  
)  
predict_class(lpa_model, newdata = df_new)
```

5 Latent Class Growth Analysis

This vignette illustrated `tidySEM`'s ability to perform latent class growth analysis, or growth mixture modeling. To follow along and run the code yourself, open the file `lca_lcga.qmd` in the `lca_course` project, as explained in [Appendix A](#).

The simulated data used for this example are inspired by work in progress by Plas and colleagues, on heterogeneity in depression trajectories among Dutch military personnel who were deployed to Afghanistan. The original data were collected as part of the *Prospection in Stress-related Military Research (PRISMO)* study, which examined of psychological problems after deployment in more than 1,000 Dutch military personnel from 2005-2019.

First, we load all required packages:

```
library(tidySEM)
library(OpenMx)
library(ggplot2)
library(MASS)
```

5.1 Data preprocessing

We first examined the descriptive statistics for the sum score scales:

```
# Get descriptives
df <- plas_depression
desc <- descriptives(df)
desc <- desc[, c("name", "mean", "median", "sd", "min", "max",
"skew_2se", "kurt_2se")]
]
desc
```

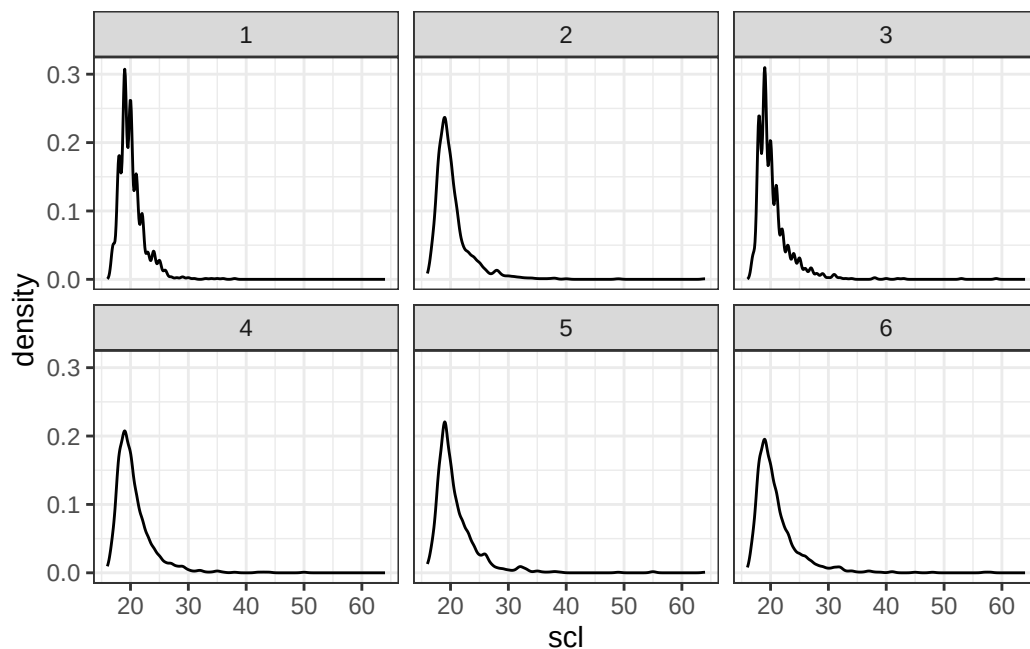
	name	mean	median	sd	min	max	skew_2se	kurt_2se
1	scl.1	20.19836	20	2.357834	17	38	14.54319	30.96859
2	scl.2	20.42025	19	3.511391	16	64	25.93999	102.90751
3	scl.3	20.48569	20	3.410847	17	59	26.44721	98.28237
4	scl.4	20.61043	20	3.359183	16	50	18.25903	44.90008

5	scl.5	20.92740	20	4.062189	16	64	24.57309	84.24579
6	scl.6	21.06851	20	4.100879	16	58	20.47371	56.46625

Note that all variables were extremely right-skewed due to censoring at the lower end of the scale.

We can examine these distributions visually as well:

```
df_plot <- reshape(df, direction = "long", varying = names(df))
ggplot(df_plot, aes(x = scl)) +
  geom_density() +
  facet_wrap(~time) + theme_bw()
```



As this type of skew can result in convergence problems in LCGA, we compared several transformations to reduce skew: The square and cube root, log, inverse, and Box-Cox transformations.

```
df_scores <- df_plot
# Store original range of SCL
rng_scl <- range(df_scores$scl)
# Log-transform
df_scores$log <- scales::rescale(log(df_scores$scl), to = c(0, 1))
# Square root transform
```

```

df_scores$sqrt <- scales::rescale(sqrt(df_scores$scl), to = c(0, 1))
# Cube root transform
df_scores$qrt <- scales::rescale(df_scores$scl^.33, to = c(0, 1))
# Reciprocal transform
df_scores$reciprocal <- scales::rescale(1/df_scores$scl, to = c(0, 1))
# Define function for Box-Cox transformation
bc <- function(x, lambda){
  (((x ^ lambda) - 1) / lambda)
}
# Inverse Box-Cox transformation
invbc <- function(x, lambda){
  ((x*lambda)+1)^(1/lambda)
}
# Box-Cox transform
b <- MASS::boxcox(lm(df_scores$scl ~ 1), plotit = FALSE)
lambda <- b$x[which.max(b$y)]
df_scores$boxcox <- bc(df_scores$scl, lambda)
# Store range of Box-Cox transformed data
rng_bc <- range(df_scores$boxcox)
df_scores$boxcox <- scales::rescale(df_scores$boxcox, to = c(0, 1))
# Rescale SCL
df_scores$scl <- scales::rescale(df_scores$scl, to = c(0, 1))

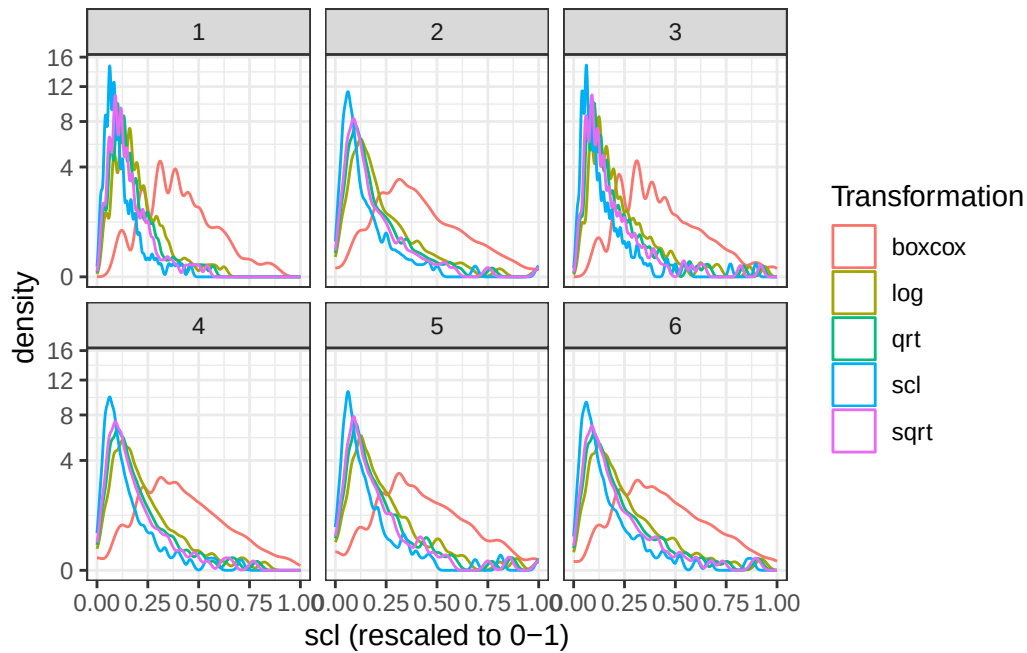
```

We can plot these transformations:

```

# Make plot data
df_plot <- do.call(rbind, lapply(c("scl", "log", "sqrt", "qrt", "boxcox"), function(n){
  data.frame(df_scores[c("time", "id")],
    Value = df_scores[[n]],
    Transformation = n)
}))
# Plot
ggplot(df_plot, aes(x = Value, colour = Transformation)) +
  geom_density() +
  facet_wrap(~time) +
  scale_y_sqrt() +
  xlab("scl (rescaled to 0-1)") +
  theme_bw()

```



Evidently, the Box-Cox transformation reduced skew the most. Consequently, we proceeded with the Box-Cox transformed scores for analysis.

```
dat <- df_scores[, c("id", "time", "boxcox")]
dat <- reshape(dat, direction = "wide", v.names = "boxcox", timevar = "time", idvar = "id")
names(dat) <- gsub("boxcox.", "scl", names(dat))
```

6 Latent Class Growth Analysis

Next, we estimated a latent class growth model for SCL. The model included an overall intercept, centered at T1, *i*. To model the potential effect of deployment on depression, we also included a dummy variable that was zero before deployment, and 1 after deployment, *step*. Finally, to model potential change (or recovery) in depression post-deployment, we included a linear slope from T2-T6, *s*. All variances of growth parameters were fixed to zero due to the sparse nature of the data. In this vignette, we do not consider more than 5 classes, because the analyses are computationally very intensive and the data were simulated from a 3-class model.

It is important to highlight that in LCGA, the subgroups will be limited by the specify growth structure, meaning that LCA will identify distinctive growth patterns within the intercept, step, and slope growth. For example, if there is a subgroup that follows a quadratic growth pattern this models will not be able to identify it.

NOTE: The time scales in this model are not correct; it currently assumes that all measurements are equidistant. Feel free to experiment with adjusting this.

```
set.seed(27796)
dat[["id"]] <- NULL
res_step <- mx_growth_mixture(
  model =
    "
    i =~ 1*scl1 + 1*scl2 + 1*scl3 +1*scl4 +1*scl5 +1*scl6
    step =~ 0*scl1 + 1*scl2 + 1*scl3 +1*scl4 +1*scl5 +1*scl6
    s =~ 0*scl1 + 0*scl2 + 1*scl3 +2*scl4 +3*scl5 +4*scl6
    scl1 ~~ vscl1*scl1
    scl2 ~~ vscl2*scl2
    scl3 ~~ vscl3*scl3
    scl4 ~~ vscl4*scl4
    scl5 ~~ vscl5*scl5
    scl6 ~~ vscl6*scl6
    i ~~ 0*i
    step ~~ 0*step
    s ~~ 0*s
    i ~~ 0*s
    i ~~ 0*step
```

```

s ~~ 0*step
scl1~0*1
scl2~0*1
scl3~0*1
scl4~0*1
scl5~0*1
scl6~0*1
i~NA*1
s~NA*1
step~NA*1
", classes = 1:5,
data = dat)
saveRDS(res_step, "res_step.rds")

```

```

res_step <- readRDS("res_step.rds")

```

Note that the first model showed convergence problems, throwing the error: *The model does not satisfy the first-order optimality conditions to the required accuracy, and no improved point for the merit function could be found during the final linesearch.* To address this problem, we performed additional iterations to

find a better solution, using `OpenMx::mxTryHardWideSearch()`. This also illustrates that `tidySEM` mixture models inherit from `OpenMx`'s `MxModel`, and thus, different `OpenMx` functions can be used to act on models specified via `tidySEM`.

The fifth model also evidenced convergence problems, but this (as we will see) is because the solution is overfitted.

6.1 Class enumeration

To determine the correct number of classes, we considered the following criteria:

1. We do not consider classes with, on average, fewer than 5 participants per parameter in a class due to potential local underidentification
2. Lower values for information criteria (AIC, BIC, saBIC) indicate better fit
3. Significant Lo-Mendell-Rubin LRT test indicates better fit for k vs $k - 1$ classes
4. We do not consider solutions with entropy $< .90$ because poor class separability compromises interpretability of the results
5. We do not consider solutions with minimum posterior classification probability $< .90$ because poor class separability compromises interpretability of the results

```
# Get fit table fit
tab_fit <- table_fit(res_step)
tab_fit[, c("Name", "Classes", "LL", "Parameters", "BIC", "Entropy", "prob_min", "n_min")]
```

	Name	Classes	LL	Parameters	BIC	Entropy	prob_min	n_min
1	1	1	2593.452	9	-5124.935	1.0000000	1.0000000	1.00000000
2	2	2	3875.587	13	-7661.663	0.9435157	0.9661069	0.24437628
3	3	3	4173.550	17	-8230.047	0.9262506	0.9316538	0.06441718
4	4	4	4278.297	21	-8411.999	0.8889950	0.8533680	0.03680982
5	5	5	4314.668	25	-8457.199	0.8637094	0.7271848	0.04089980

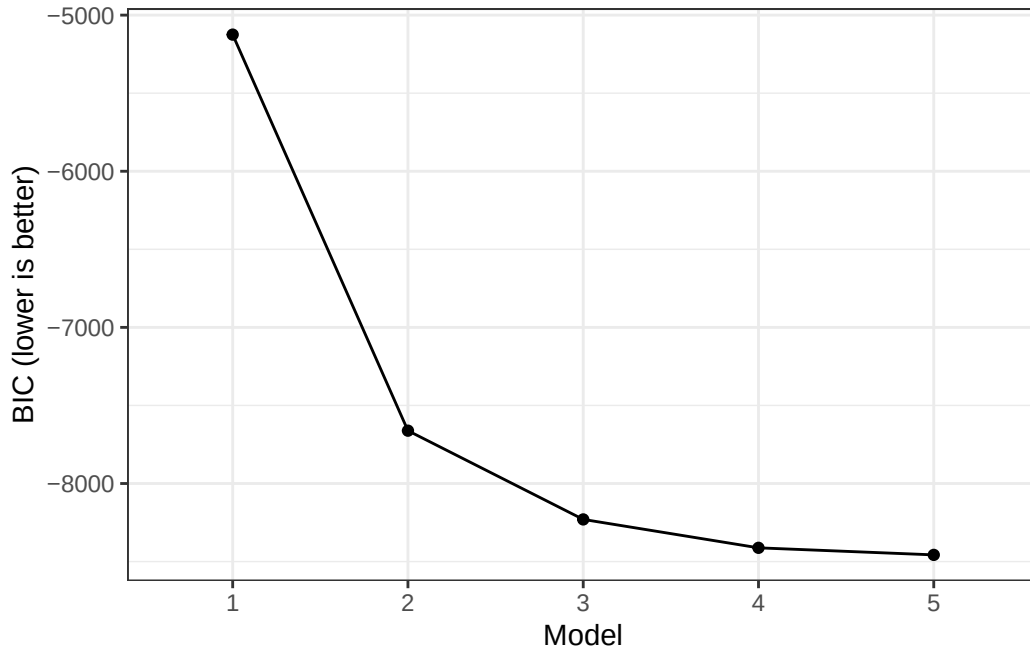
According to the Table, increasing the number of classes keeps increasing model fit according to all ICs except the BIC, which increased after 3 classes.

The first two LMR tests are significant, indicating that a 2- and 3-class solution were a significant improvement over a 1- and 2-class solution, respectively. However, solutions with >3 classes had entropy and minimum posterior classification probability below the pre-specified thresholds. Models with >3 solutions also had fewer than five observations per parameter. This suggests that the preferred model should be selected from 1-3 classes.

6.1.1 Scree plot

A scree plot indicates that the largest decrease in ICs occurs from 1-2 classes, and the inflection point for all ICs is at 3 classes. Moreover, the BIC increased after 3 classes. A three-class solution thus appears to be the most parsimonious solution with good fit.

```
plot(tab_fit, statistics = "BIC")
```



Based on the aforementioned criteria, we selected a 3-class model for further analyses. First, to prevent label switching, we re-order these classes by the value of the intercept i . Then, we report the estimated parameters.

```
res_final <- mx_switch_labels(res_step[[3]], param = "M[1,7]", decreasing = FALSE)
tab_res <- table_results(res_final, columns = NULL)
# Select rows and columns
tab_res <- tab_res[tab_res$Category %in% c("Means", "Variances"), c("Category", "lhs", "est"
tab_res
```

	Category	lhs	est	se	pval	confint	name
16	Means	i	0.33	0.00	0.00	[0.32, 0.33]	class1.M[1,7]
17	Means	step	-0.02	0.01	0.00	[-0.03, -0.01]	class1.M[1,8]
18	Means	s	0.00	0.00	0.00	[0.00, 0.01]	class1.M[1,9]
19	Variances	scl1	0.01	0.00	0.00	[0.01, 0.01]	class1.S[1,1]
20	Variances	scl2	0.01	0.00	0.00	[0.01, 0.01]	class1.S[2,2]
21	Variances	scl3	0.01	0.00	0.00	[0.01, 0.01]	class1.S[3,3]
22	Variances	scl4	0.01	0.00	0.00	[0.01, 0.01]	class1.S[4,4]
23	Variances	scl5	0.01	0.00	0.00	[0.01, 0.01]	class1.S[5,5]
24	Variances	scl6	0.01	0.00	0.00	[0.01, 0.02]	class1.S[6,6]
40	Means	i	0.45	0.01	0.00	[0.43, 0.46]	class2.M[1,7]
41	Means	step	0.03	0.01	0.00	[0.01, 0.05]	class2.M[1,8]
42	Means	s	0.02	0.00	0.00	[0.01, 0.02]	class2.M[1,9]

43	Variances	scl1	0.01	0.00	0.00	[0.01, 0.01]	class2.S[1,1]
44	Variances	scl2	0.01	0.00	0.00	[0.01, 0.01]	class2.S[2,2]
45	Variances	scl3	0.01	0.00	0.00	[0.01, 0.01]	class2.S[3,3]
46	Variances	scl4	0.01	0.00	0.00	[0.01, 0.01]	class2.S[4,4]
47	Variances	scl5	0.01	0.00	0.00	[0.01, 0.01]	class2.S[5,5]
48	Variances	scl6	0.01	0.00	0.00	[0.01, 0.02]	class2.S[6,6]
64	Means	i	0.60	0.01	0.00	[0.57, 0.63]	class3.M[1,7]
65	Means	step	0.10	0.02	0.00	[0.07, 0.14]	class3.M[1,8]
66	Means	s	0.01	0.00	0.08	[-0.00, 0.02]	class3.M[1,9]
67	Variances	scl1	0.01	0.00	0.00	[0.01, 0.01]	class3.S[1,1]
68	Variances	scl2	0.01	0.00	0.00	[0.01, 0.01]	class3.S[2,2]
69	Variances	scl3	0.01	0.00	0.00	[0.01, 0.01]	class3.S[3,3]
70	Variances	scl4	0.01	0.00	0.00	[0.01, 0.01]	class3.S[4,4]
71	Variances	scl5	0.01	0.00	0.00	[0.01, 0.01]	class3.S[5,5]
72	Variances	scl6	0.01	0.00	0.00	[0.01, 0.02]	class3.S[6,6]

As evident from these results, Class 1 started at a relatively lower level of depressive symptoms, experienced a decrease after deployment, followed by increase over time. Class 2 started at a moderate level of depressive symptoms, experienced an increase after deployment, followed by significant increase over time from T2-T6. Class 3 started at a relatively higher level, experienced an increase after deployment, followed by stability.

6.2 Wald tests

To test whether parameters are significantly different between classes, we can use Wald tests. Wald tests can be specified for all parameters in the model, using the hypothesis syntax from the **bain** package for informative hypothesis testing.

To identify the names of parameters in the model, we can use the **name** column of the results table above. Alternatively, to see all parameters in the model, run:

```
names(coef(res_final))
```

```
[1] "mix3.weights[1,2]" "mix3.weights[1,3]" "vsc11"
[4] "vsc12"             "vsc13"             "vsc14"
[7] "vsc15"             "vsc16"             "class1.M[1,7]"
[10] "class1.M[1,8]"     "class1.M[1,9]"     "class2.M[1,7]"
[13] "class2.M[1,8]"     "class2.M[1,9]"     "class3.M[1,7]"
[16] "class3.M[1,8]"     "class3.M[1,9]"
```

Next, specify equality constrained hypotheses. For example, a hypothesis that states that the mean intercept is equal across groups is specified as follows:

```
"class1.M[1,7] = class2.M[1,7] & class1.M[1,7] = class3.M[1,7]"
```

It is also possible to consider comparisons between two classes at a time. When conducting many significance tests, consider correcting for multiple comparisons however.

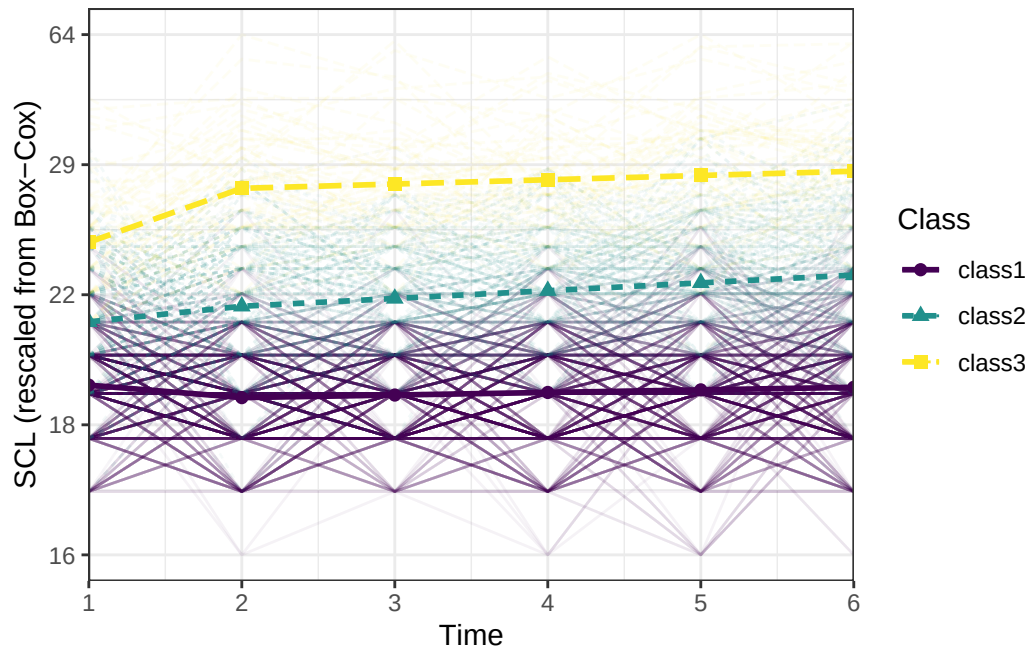
```
# Load the bain library to parse complex hypothesis syntax:
library(bain)
wald_tests <- wald_test(res_final,
  "
    class1.M[1,7] = class2.M[1,7]&
    class1.M[1,7] = class3.M[1,7];
    class1.M[1,8] = class2.M[1,8]&
    class1.M[1,8] = class3.M[1,8];
    class1.M[1,9] = class2.M[1,9]&
    class1.M[1,9] = class3.M[1,9]"
# Rename the hypothesis
wald_tests$Hypothesis <- c("Mean i", "Mean step", "Mean slope")
wald_tests
```

All Wald tests are significant, indicating that there are significant differences between the intercepts, step function, and slopes of the three classes.

6.3 Trajectory plot

Finally, we can plot the growth trajectories. This can help interpret the results better, as well as the residual heterogeneity around class trajectories.

```
p <- plot_growth(res_step[[3]], rawdata = TRUE, alpha_range = c(0, .05))
# Add Y-axis breaks in original scale
brks <- seq(0, 1, length.out = 5)
labs <- round(invbc(scales::rescale(brks, from = c(0, 1), to = rng_bc), lambda))
p <- p + scale_y_continuous(breaks = seq(0, 1, length.out = 5), labels = labs) + ylab("SCL (")
p
```



Note that the observed individual trajectories show very high variability within classes.

7 Specifying Custom Models

The convenience functions in `tidySEM`, such as `mx_profiles()`, `mx_lca()`, and `mx_mixed_lca()`, are designed to make common latent class and latent profile models easy to specify. Sometimes, however, the model you want does not fit one of these standard templates. You may want a regression coefficient to vary across classes, allow a residual covariance only in one class, impose a particular equality constraint, or combine latent class analysis with another structural model. For these situations, `tidySEM` provides `mx_mixture()`.

To follow along and run the code yourself, open the file `custom_models.qmd` in the `lca_course` project, as explained in [Appendix A](#).

```
library(tidySEM)
library(OpenMx)
set.seed(1300)
```

`mx_mixture()` dynamically constructs finite mixture models in `OpenMx`. Its basic interface is:

```
mx_mixture(
  model,
  classes = 1L,
  data = NULL,
  run = TRUE,
  ...
)
```

The essential idea is simple: instead of choosing from a predefined latent class model, you describe the model that should be fitted *within each latent class*. `mx_mixture()` then creates the class-specific models and combines them into a finite mixture model.

This flexibility makes `mx_mixture()` useful for models that go beyond conventional LCA or latent profile analysis.

7.1 Three ways to specify a mixture model

The `model` argument can be supplied in three ways:

1. a single character string containing `lavaan`-style syntax;
2. a list of character strings, with one model specification for each class; or
3. a list of `OpenMx::mxModel()` objects constructed manually.

These options provide increasing levels of control. Most custom models can be specified conveniently with the first two approaches. The third approach is useful when the desired model cannot be represented easily using `lavaan`-style syntax.

7.2 Using `lavaan` Syntax with `{C}`

The most convenient way to build a custom mixture model is to use a single model specification in `lavaan` syntax, but adding the special placeholder `{C}`.

Inside the model syntax, `{C}` is replaced by the class number. Thus,

```
m{C}
```

is translated to `m1`, `m2`, `m3`, and so forth in a model with three classes.

This makes it easy to assign a different parameter label to the same parameter in every class.

For example, a single-indicator model with free means and variances across classes can be specified as:

```
"x ~ m{C}*1  
x ~~ v{C}*x"
```

To fix the variance across classes, assign it a single, not class-specific label:

```
"x ~ m{C}*1  
x ~~ v*x"
```

Use the `{C}` placeholder to generate *different* labels, thus freeing that parameter across classes. Conversely, use a label **without** `{C}` to constrain a parameter to be equal across classes.

```
df <- iris[, c("Sepal.Width", "Sepal.Length")]
names(df) <- c("x", "y")

fit <- mx_mixture(
  model = "
    x ~ m{C}*1
    x ~~ v{C}*x
  ",
  classes = 2,
  data = df
)
table_results(fit)
```

	label	est_sig	se	pval	confint	class
1	Means.x.class1	3.81***	0.30	0.00	[3.23, 4.40]	class1
2	Variances.x.class1	0.08	0.08	0.29	[-0.07, 0.24]	class1
3	Means.x.class2	2.98***	0.07	0.00	[2.86, 3.11]	class2
4	Variances.x.class2	0.14***	0.03	0.00	[0.08, 0.20]	class2
5	mix2.weights[1,1].NA	1.00	<NA>	<NA>	<NA>	<NA>
6	mix2.weights[1,2].NA	10.38	12.21	0.40	[-13.55, 34.32]	<NA>

Verify that the parameters are as expected.

7.3 Example 1: A latent class regression model

A conventional latent profile model focuses on class differences in means and variances. A more unusual model allows the **relationship between two variables** to differ across latent classes.

For example, suppose that x predicts y , but we hypothesize that the strength of this relationship differs between otherwise unobserved subpopulations.

```
fit_regression <- mx_mixture(
  model = "
    y ~ b{C}*x
    y ~ m{C}*1
    y ~~ vy{C}*y
    x ~~ vx*x
  ",
  classes = 2,
```

```
data = df
)
table_fit(fit_regression)
```

```

      Minus2LogLikelihood    n Parameters observedStatistics    df RMSEASquared
1          451.8777 150          10          300 290          0
      RMSEANull modelName      AIC      BIC    saBIC Classes    Entropy  prob_min
1      0.05      mix2 471.8777 501.9841 470.336      2 0.9709188 0.9929033
      prob_max      n_min      n_max      LL
1 0.998904 0.3266667 0.6733333 -225.9389
```

```
table_results(fit_regression)
```

```

              label est_sig    se pval      confint  class
1  Regressions.y.ON.x.class1 1.13*** 0.17 0.00 [0.79, 1.47] class1
2              Means.y.class1 2.98*** 0.50 0.00 [2.00, 3.96] class1
3              Means.x.class1 2.87*** 0.03 0.00 [2.80, 2.93] class1
4      Variances.y.class1 0.32*** 0.05 0.00 [0.23, 0.42] class1
5      Variances.x.class1 0.11*** 0.01 0.00 [0.09, 0.14] class1
6  Regressions.y.ON.x.class2 0.75*** 0.10 0.00 [0.55, 0.95] class2
7              Means.y.class2 2.43*** 0.35 0.00 [1.75, 3.11] class2
8              Means.x.class2 3.46*** 0.05 0.00 [3.36, 3.55] class2
9      Variances.y.class2 0.05*** 0.01 0.00 [0.03, 0.08] class2
10      mix2.weights[1,1].NA    1.00 <NA> <NA>      <NA>  <NA>
11      mix2.weights[1,2].NA 0.48*** 0.08 0.00 [0.31, 0.64]  <NA>
```

The regression coefficient is labeled $b\{C\}$, so the slope is estimated separately in each class. Conceptually, the model asks whether there are latent subpopulations based on the size of the regression coefficient. One class might show a strong positive association between x and y , while another might show a weak or even negative association. For example, you could use this to identify subgroups for whom a type of therapy is or is not effective.

As an exercise:

- Constrain some parameters in this model, and free others
- Consider how this impacts fit
- Use model comparison techniques you learned before

7.4 Specifying Class-specific Models

Sometimes the classes should differ not merely in parameter values, but in their **model structure**. In that case, provide a list of model specifications. The package documentation illustrates this approach with class-specific regression models:

```
fit <- mx_mixture(  
  model = list(  
    "y ~ x",  
    "y ~~ x"  
  ),  
  meanstructure = TRUE,  
  data = df  
)  
table_results(fit)
```

	label	est_sig	se	pval	confint	class
1	Regressions.y.ON.x.class1	1.13***	0.17	0.00	[0.79, 1.47]	class1
2	Means.y.class1	2.99***	0.50	0.00	[2.00, 3.97]	class1
3	Means.x.class1	2.87***	0.03	0.00	[2.80, 2.93]	class1
4	Variances.y.class1	0.32***	0.05	0.00	[0.23, 0.42]	class1
5	Variances.x.class1	0.11***	0.02	0.00	[0.08, 0.14]	class1
6	Means.y.class2	5.02***	0.05	0.00	[4.92, 5.11]	class2
7	Means.x.class2	3.45***	0.05	0.00	[3.36, 3.55]	class2
8	Variances.y.class2	0.12***	0.02	0.00	[0.07, 0.17]	class2
9	Covariances.y.WITH.x.class2	0.09***	0.02	0.00	[0.05, 0.13]	class2
10	Variances.x.class2	0.12***	0.02	0.00	[0.07, 0.17]	class2
11	mix2.weights[1,1].NA	1.00	<NA>	<NA>	<NA>	<NA>
12	mix2.weights[1,2].NA	0.48***	0.08	0.00	[0.31, 0.64]	<NA>

You could use this approach when you have a hypothesis that two variables are associated in one group, but not in another. Here, the regression from x to y is estimated in Class 1 but fixed to zero in Class 2:

```
fit_structural_difference <- mx_mixture(  
  model = list(  
    "  
    y ~ b1*x  
    y ~ m1*1  
    y ~~ v1*y  
    ",
```

```

"
  y ~ 0*x
  y ~ m2*1
  y ~~ v2*y
"
),
meanstructure = TRUE,
data = df
)

```

7.5 Modeling Zero Inflation

Sometimes an apparent latent class does not represent a substantively distinct population, but instead captures an unusual feature of an observed distribution. **Zero inflation** is a useful example.

Consider depressive symptoms. In a general-population sample, many people may report no depressive symptoms at all and consequently receive a score of zero. Among people who do experience depressive symptoms, however, scores may follow an approximately continuous distribution.

Suppose we are also interested in smoking. For illustration, assume that smoking is approximately normally distributed and that, among people with depressive symptoms, higher depression causes an increase in smoking.

This gives us a distribution that cannot be represented very naturally by an ordinary Gaussian latent profile model. Instead, we can formulate two different within-class models:

1. a **structural-zero class**, which captures the excess observations at zero on depression; and
2. a **continuous depression class**, in which depression varies continuously and predicts smoking.

7.5.1 Simulating the data

We first generate data with the structure just described.

```

n <- 300
# Depression is approximately normally distributed
depression <- pmax(0, rnorm(n, mean = 8, sd = 3))
# But about 60% of the sample has no depressive symptoms
depression[sample(c(TRUE, FALSE), size = n, prob = c(.4, .6), replace = TRUE)] <- 0

```

```
# Smoking is approximately normally distributed.
smoking <- rnorm(n, mean = 5, sd = 2)
# Among people with depressive symptoms, depression has a small effect.
smoking <- smoking + .15 * depression

df_zip <- data.frame(
  depression = depression,
  smoking = smoking
)
```

7.5.2 Specifying the Models

The important feature of the simulated depression variable is the spike at zero.

```
hist(df_zip$depression)
```



An ordinary normal mixture could attempt to approximate this distribution by estimating one Gaussian component with a very low mean and variance. But if our substantive hypothesis is specifically that one component represents **structural zeros**, we can encode that hypothesis directly.

For the first class, depression is fixed at zero. Because everyone represented by this component is assumed to have a depression score of zero, both its mean and variance are fixed to zero.

Smoking, however, is still allowed to vary normally in this class.

For the second class, depression has an estimated mean and variance. Smoking is regressed on depression, allowing us to estimate the association between depressive symptoms and smoking among individuals belonging to this continuous component.

```
model_zero <- "
  # Depression is a structural zero
  depression ~ 0*1
  depression ~~ 0*depression

  # Smoking remains normally distributed
  smoking ~ smoke_mean1*1
  smoking ~~ smoke_var1*smoking
"

model_depressed <- "
  # Continuous depression distribution
  depression ~ dep_mean*1
  depression ~~ dep_var*depression

  # Smoking depends on depression
  smoking ~ smoke_mean2*1
  smoking ~ b*depression
  smoking ~~ smoke_var2*smoking
"

fit_zip <- mx_mixture(
  model = list(
    model_zero,
    model_depressed
  ),
  data = df_zip
)
table_fit(fit_zip)
```

	Minus2LogLikelihood	n	Parameters	observedStatistics	df	RMSEASquared		
1	2844.468	300	8	600	592	0		
	RMSEANull	modelName	AIC	BIC	saBIC	Classes	Entropy	prob_min
1	0.05	mix2	2860.468	2890.098	2864.727	2	0.9111211	0.9791151

```

      prob_max      n_min      n_max      LL
1  0.98411 0.4433333 0.5566667 -1422.234

```

```
table_results(fit_zip)
```

	label	est_sig	se	pval	confint
1	Means.smoking.class1	4.98***	0.18	0.00	[4.63, 5.33]
2	Variances.depression.class1	1.98	<NA>	<NA>	<NA>
3	Variances.smoking.class1	4.03***	0.51	0.00	[3.03, 5.02]
4	Regressions.smoking.ON.depression.class2	0.11	0.06	0.05	[-0.00, 0.22]
5	Means.smoking.class2	5.42***	0.50	0.00	[4.45, 6.39]
6	Means.depression.class2	8.14***	0.25	0.00	[7.64, 8.64]
7	Variances.smoking.class2	4.10***	0.46	0.00	[3.20, 5.00]
8	Variances.depression.class2	8.93***	1.17	0.00	[6.64, 11.21]
9	mix2.weights[1,1].NA	1.00	<NA>	<NA>	<NA>
10	mix2.weights[1,2].NA	1.25***	0.15	0.00	[0.95, 1.55]


```

      class
1  class1
2  class1
3  class1
4  class2
5  class2
6  class2
7  class2
8  class2
9   <NA>
10  <NA>

```

Note that the regression coefficient is estimated quite precisely.

7.5.3 Why not simply use `mx_profiles()`?

A conventional call such as

```

fit_lpa_zip <- mx_mixture(
  model = "
    depression ~ dep_mean{C}*1
    depression ~~ dep_var{C}*depression

    # Smoking depends on depression
    smoking ~ smoke_mean2{C}*1

```

```

smoking ~ b{C}*depression
smoking ~~ smoke_var2{C}*smoking
",
  data = df_zip,
  classes = 2
)
table_results(fit_lpa_zip)

```

	label	est_sig	se	pval
1	Regressions.smoking.ON.depression.class1	-88175604.40	<NA>	<NA>
2	Means.smoking.class1	4.95	7.60	0.51
3	Means.depression.class1	-0.00	8278.93	1.00
4	Variances.smoking.class1	4.04	<NA>	<NA>
5	Variances.depression.class1	0.00	6936309.37	1.00
6	Regressions.smoking.ON.depression.class2	0.10	111.27	1.00
7	Means.smoking.class2	5.46	12.93	0.67
8	Means.depression.class2	8.08	8.75	0.36
9	Variances.smoking.class2	4.07	4.53	0.37
10	Variances.depression.class2	8.89***	2.07	0.00
11	mix2.weights[1,1].NA	1.00	<NA>	<NA>
12	mix2.weights[1,2].NA	1.30	13.17	0.92
	confint	class		
1	<NA>	class1		
2	[-9.95, 19.85]	class1		
3	[-16226.40, 16226.40]	class1		
4	<NA>	class1		
5	[-13594916.55, 13594916.55]	class1		
6	[-217.98, 218.19]	class2		
7	[-19.87, 30.80]	class2		
8	[-9.06, 25.23]	class2		
9	[-4.81, 12.95]	class2		
10	[4.83, 12.95]	class2		
11	<NA>	<NA>		
12	[-24.52, 27.13]	<NA>		

Notice that this model returns nonsense coefficients.

7.6 Using OpenMx Models

The third interface to `mx_mixture()` is to supply a list of `OpenMx::mxModel()` objects.

This interface gives the user direct access to OpenMx matrices, algebra, constraints, and fit functions. It is the most flexible option, but it also requires substantially more knowledge of OpenMx.

Use this approach when the desired model cannot be expressed adequately using the `lavaan`-style syntax handled by `as_ram()`.

7.7 Building models without immediately estimating them

By default, `mx_mixture()` estimates the model. Internally, `run = TRUE` causes tidySEM to obtain mixture starting values and run the OpenMx model.

During model development, it can be useful to create the model first without estimating it:

```
model_object <- mx_mixture(  
  model = "  
    x ~ m{C}*1  
    x ~~ v{C}*x  
  ",  
  classes = 2,  
  data = df,  
  run = FALSE  
)  
  
model_object
```

```
MxModel 'mix2'  
type : default  
$matrices : 'weights'  
$algebras : NULL  
$penalties : NULL  
$constraints : NULL  
$intervals : NULL  
$latentVars : none  
$manifestVars : none  
$data : 150 x 2  
$data means : NA  
$data type: 'raw'  
$submodels : 'class1' and 'class2'  
$expectation : MxExpectationMixture  
$fitfunction : MxFitFunctionML  
$compute : NULL
```

```
$independent : FALSE
$options :
$output : FALSE
```

This allows advanced users to inspect or modify the generated OpenMx model before estimation.

These models typically contain the following matrices:

The ‘A’ argument refers to the A or asymmetric matrix in the RAM approach. This matrix consists of all of the asymmetric paths (one-headed arrows) in the model. A free parameter in any row and column describes a regression of the variable represented by that row regressed on the variable represented in that column.

The ‘S’ argument refers to the S or symmetric matrix in the RAM approach, and as such must be square. This matrix consists of all of the symmetric paths (two-headed arrows) in the model. A free parameter in any row and column describes a covariance between the variable represented by that row and the variable represented by that column. Variances are covariances between any variable at itself, which occur on the diagonal of the specified matrix.

The ‘F’ argument refers to the F or filter matrix in the RAM approach. If no latent variables are included in the model (i.e., the A and S matrices are of both of the same dimension as the data matrix), then the ‘F’ should refer to an identity matrix. If latent variables are included (i.e., the A and S matrices are not of the same dimension as the data matrix), then the ‘F’ argument should consist of a horizontal adhesion of an identity matrix and a matrix of zeros.

The ‘M’ argument refers to the M or means matrix in the RAM approach. It is a 1 x n matrix, where n is the number of manifest variables + the number of latent variables. The M matrix must be specified if either the mxData type is “cov” or “cor” and a means vector is provided, or if the mxData type is “raw”. Otherwise the M matrix is ignored.

You can access them as follows:

```
model_object$class1$A
model_object$class1$S
model_object$class1$F
model_object$class1$M
```

Each has several properties, including (starting) value, whether it is freely estimated or not, upper/lower bounds, labels, each of which you could modify by hand:

```
model_object$class1$A$values  
model_object$class1$A$labels  
model_object$class1$A$free  
model_object$class1$A$lbound  
model_object$class1$A$ubound
```

Once the specification is satisfactory, the model can be estimated using tidySEM's OpenMx workflow.

```
fit <- run_mx(model_object)
```

7.8 Conclusion

`mx_mixture()` is the general-purpose mixture-modeling function underlying much of tidySEM's OpenMx mixture functionality. It accepts a single dynamic model specification, class-specific syntax, or fully specified OpenMx models.

Use `mx_mixture()` when the scientific hypothesis concerns something that the standard wrappers do not expose directly: class-specific regression coefficients, particular equality constraints, or entirely different models between classes.

References

- Celeux, G., & Soromenho, G. (1996). An entropy criterion for assessing the number of clusters in a mixture model. *Journal of Classification*, *13*(2), 195–212. <https://doi.org/10.1007/BF01246098>
- Van Lissa, C. J., Garnier-Villarreal, M., & Anadria, D. (2024). Recommended Practices in Latent Class Analysis Using the Open-Source R-Package tidySEM. *Structural Equation Modeling: A Multidisciplinary Journal*, *31*(3), 526–534. <https://doi.org/10.1080/10705511.2023.2250920>

A Setup your Computer

This is a tutorial on how to set up your personal computer for the course. It guides you through the installation of several software packages. This vignette does not assume a prior installation of R, so it is suitable for novice users. You only have to perform these steps once for every computer you intend to use, and the entire process should take approximately 30 minutes if you start from scratch. In case some of the software is already installed on your system, you can skip those related steps.

Follow these steps in order:

1. Install [R \(free\)](#)
2. Install [‘RStudio’ Desktop \(Free\)](#)
3. Download the [RStudio project](#) for this course into a folder where you’d like to keep your working files for assignments.
4. Unzip the compressed folder, open it, and open `lca_course.Rproj` in RStudio.
5. Once RStudio has loaded, you’ll see a panel in the bottom right labeled “Files”. Click the file named `setup.R`. It will load. Select all code in this file, then either press [CONTROL+ENTER], or click the Run button.

The exercises for the different chapters are found in their respective `.qmd` files.