Skalierungsanleitung für das NEPSroutines Paket (Version 1.2.0)

Jana Welling, Christian Diegel und Tanja Kutscher

2024-20-11

# Table of Contents

[Table of Contents 1](#_Toc138861295)

[1. Allgemeine Informationen 4](#_Toc138861296)

[Skalierung von Kompetenztests im NEPS 4](#_Toc138861297)

[Kompetenzerhebungen im NEPS 4](#_Toc138861298)

[Kompetenzitems 5](#_Toc138861299)

[Das Messmodell der Item Response Theorie 5](#_Toc138861300)

[Linking 5](#_Toc138861301)

[Multistagetests 6](#_Toc138861302)

[Grundlagen in R 6](#_Toc138861303)

[Skalierung mit dem Paket NEPSroutines 7](#_Toc138861304)

[Beispieldaten 7](#_Toc138861305)

[2. Vorbereitungen 8](#_Toc138861306)

[Vorbereitungen in R 8](#_Toc138861307)

[Ein R-Projekt im Studienordner erstellen 8](#_Toc138861308)

[Das Paket NEPSroutines installieren und laden 8](#_Toc138861309)

[RMarkdown 9](#_Toc138861310)

[SessionInfo 9](#_Toc138861311)

[Benötigte Daten und Dateien 9](#_Toc138861312)

[Kompetenzdaten 9](#_Toc138861313)

[Informationen über Kompetenzitems 9](#_Toc138861314)

[Soziodemographische Daten 10](#_Toc138861315)

[Informationen zum Testheftdesign 10](#_Toc138861316)

[3. Arbeiten mit dem Paket NEPSroutines 11](#_Toc138861317)

[Benötigte Datenstruktur 11](#_Toc138861318)

[Gruppierungen 11](#_Toc138861319)

[Datensatz resp 12](#_Toc138861320)

[Datensatz vars 14](#_Toc138861321)

[Beispieldaten 17](#_Toc138861322)

[Aufbau des Pakets 18](#_Toc138861323)

[Output der Funktionen 18](#_Toc138861324)

[Häufige Funktionsargumente 19](#_Toc138861325)

[Hilfreiche allgemeine Funktionen 21](#_Toc138861326)

[Debugging 22](#_Toc138861327)

[4. Vorbereitung der Daten 23](#_Toc138861328)

[Vorbereitung der Kompetenzitems 23](#_Toc138861329)

[Laden der Daten und erste Anpassungen 23](#_Toc138861330)

[Dichotomes Item Scoring 23](#_Toc138861331)

[Polytomes Item Scoring 24](#_Toc138861332)

[User defined missing values 26](#_Toc138861333)

[Vorbereitung der DIF-Variablen 28](#_Toc138861334)

[Laden der Daten und Zusammenführung mit Kompetenzdaten 28](#_Toc138861335)

[Berechnen des Alters 28](#_Toc138861336)

[Dichotomisieren der soziodemographischen Variablen 29](#_Toc138861337)

[Erstellen der Variablen zum Testheftdesign 29](#_Toc138861338)

[Labels für die Variablen 29](#_Toc138861339)

[Sonstige Vorbereitungen 30](#_Toc138861340)

[Valide Fälle 30](#_Toc138861341)

[Gruppenvariablen 30](#_Toc138861342)

[Variablen für die Scores und den SUF 31](#_Toc138861343)

[Weitere benötigte Variablen 31](#_Toc138861344)

[Check der Datenstruktur 32](#_Toc138861345)

[Speichern der Daten 33](#_Toc138861346)

[Veränderungen an den Items während der Skalierung 33](#_Toc138861347)

[5. Deskriptive Analysen 34](#_Toc138861348)

[Deskriptive Analyse der Stichprobe 34](#_Toc138861349)

[Deskriptive Analyse nach Subgruppen (falls zutreffend) 35](#_Toc138861350)

[6. Analyse fehlender Werte 37](#_Toc138861351)

[Allgemeine Informationen 37](#_Toc138861352)

[Analyse fehlender Werte pro Person 37](#_Toc138861353)

[Übergreifende Funktion 37](#_Toc138861354)

[Untergeordnete Funktionen 39](#_Toc138861355)

[Analyse fehlender Werte pro Item 39](#_Toc138861356)

[Übergreifende Funktion 39](#_Toc138861357)

[Untergeordnete Funktionen 41](#_Toc138861358)

[Analyse fehlender Werte bei Multistagetests 42](#_Toc138861359)

[7. IRT-Analyse 44](#_Toc138861360)

[Allgemeine Informationen 44](#_Toc138861361)

[Übergreifende Funktion irt\_analysis 44](#_Toc138861362)

[Übergreifende Funktion grouped\_irt\_analysis 46](#_Toc138861363)

[Untergeordnete Funktionen 47](#_Toc138861364)

[8. DIF-Analyse 49](#_Toc138861365)

[Allgemeine Information 49](#_Toc138861366)

[Übergreifende Funktion 49](#_Toc138861367)

[Untergeordnete Funktionen 51](#_Toc138861368)

[9. Distraktorenanalyse 52](#_Toc138861369)

[Allgemeine Information 52](#_Toc138861370)

[Übergreifende Funktion 52](#_Toc138861371)

[Untergeordnete Funktionen 53](#_Toc138861372)

[10. Dimensionalitätsanalyse 54](#_Toc138861373)

[Allgemeine Information 54](#_Toc138861374)

[Übergreifende Funktion 54](#_Toc138861375)

[Untergeordnete Funktionen 55](#_Toc138861376)

[11. Linking 57](#_Toc138861377)

[12. Scores berechnen 58](#_Toc138861378)

[13. SUF erstellen 61](#_Toc138861379)

[Übergreifende Funktion 61](#_Toc138861380)

[Untergeordnete Funktionen 61](#_Toc138861381)

[14. Literatur 62](#_Toc138861382)

# Allgemeine Informationen

Die vorliegende Skalierungsanleitung ist vor allem dazu gedacht, in Skalierungen von Kompetenztests der National Educational Panel Study (NEPS) mit Hilfe des R-Pakets NEPSroutines einzuführen. Sie ist dahingegen nicht als ausführliche Einleitung in die methodischen Grundlagen der Skalierung oder Grundlagen in R gedacht. Die folgenden Abschnitte enthalten zwar eine kurze Übersicht über die Skalierungen im NEPS und das Arbeiten mit R, sie sind jedoch nur oberflächlich gehalten und geben vor allem Tipps zum weiteren Einlesen.

## Skalierung von Kompetenztests im NEPS

Die Skalierung von Kompetenztests des NEPS beinhaltet in den meisten Fällen folgende Schritte:

* Datensuche und -vorbereitung
* Deskriptive Analyse
* Analyse fehlender Werte (pro Person & pro Item)
* Item Response Theory (IRT)-Analysen
* Differential Item Functioning (DIF)-Analysen
* Distraktorenanalysen
* Dimensionalitätsanalysen
* Linking
* Schätzung der Weighted maximum Likelihood Estimates (WLE) und Erstellen der Scientific Use File (SUF)
* Dokumentation der Ergebnisse im Technical Report (TR)

Einzelne Schritte können bei bestimmten Skalierungen wegfallen, wenn die Voraussetzungen nicht gegeben sind (z.B. keine Strukturen im Test erkennbar, die eine Dimensionalitätsanalyse rechtfertigen würden). Die vorliegende Anleitung führt dich ab Kapitel 4 Schritt für Schritt durch jeden einzelnen dieser Abschnitte. In den Kapiteln 2 und 3 werden die wichtigsten Informationen zu den Vorbereitungen in R, den benötigten Daten und das Arbeiten mit dem NEPSroutines Paket zusammengefasst. Dieses Kapitel enthält weitere Informationen zu den Kompetenzerhebungen und dem verwendeten Messmodell in NEPS.

### Kompetenzerhebungen im NEPS

Allgemeine Informationen über die Kompetenzerhebungen im NEPS findet man im folgenden Dokument: <https://www.neps-data.de/Portals/0/NEPS/Datenzentrum/Forschungsdaten/Kompetenzen/Overview_NEPS_Competence-Data.pdf>. Dort werden z.B. Namensgebungskonventionen erklärt und ein Überblick über alle bisherigen Kompetenzerhebungen gewährt. Möchte man über eine bestimmte Kompetenzerhebung mehr wissen, kann man unter dem Link <https://www.neps-data.de/Datenzentrum/Daten-und-Dokumentation> bei der entsprechenden Startkohorte und der entsprechenden Welle weitere hilfreiche Dokumente zu den Erhebungen und Tests finden.

### Kompetenzitems

Die Aufgaben in einem Kompetenztest werden Items genannt. Diese liegen zunächst in den Rohdaten **ungescort** vor, sie beinhalten also noch die von der Testperson gegebene Antwortmöglichkeit (z.B. “3” wenn die Person die dritte Antwortmöglichkeit gewählt hat). Am Anfang des Skalierungsprozesses werden die Kompetenzitems **gescort**: Jedes Item wird in ein neues Item umgewandelt, das als Information nur noch enthält, ob die Antwort richtig (1) oder falsch (0) ist. Es ist also nun **dichotom**, da es nur zwei mögliche Ausprägungen besitzt.

Manche Aufgaben eines Kompetenztests bestehen aus mehreren einzelnen (Sub-)Items (z.B. Complex Multiple Choice oder Matching Aufgaben). Diese Aufgaben werden **polytome** Items genannt. Den Score eines polytomen Items erhält man, indem man alle (dichotomen) Scores der Subitems aufsummiert. Polytome Items haben also im Gegensatz zu einfachen Items mehr als zwei Ausprägungen. Ein polytomes Item mit drei Subitems hat beispielsweise vier Ausprägungen: Es können insgesamt 0, 1, 2 oder 3 richtige Antworten vorliegen - das Item wird also entsprechend mit 0, 1, 2 oder 3 gescort.

Im Skalierungsprozess wird oft zwischen dichotomen und polytomen Items unterschieden. Zu den dichotomen Items zählen in diesem Zusammenhang zunächst alle gescorten, einfachen Items (z.B. Multiple Choice Aufgaben) und alle gescorten Subitems von polytomen Items. Zudem zählen dazu ursprünglich polytom gescorte Items, die nach der Zusammenlegung der Antwortkategorien nur noch zwei Ausprägungen besitzen (s. Abschnitt 4.1.3 *Polytomes Item Scoring*). Zu den polytomen Items zählen alle gescorten Items mit mehr als zwei Ausprägungen.

### Das Messmodell der Item Response Theorie

Die Skalierungsgrundlage für das NEPS und der übliche Skalierungsprozess wurde im NEPS Working Paper No. 14 beschrieben (Pohl & Carstensen, 2012). Man findet es unter folgendem Link: <https://www.neps-data.de/Portals/0/Working%20Papers/WP_XIV.pdf>. Die Kompetenztests im NEPS wurden entwickelt, um dem Rasch- (Rasch, 1960) bzw. dem Partial-Credit-Modell (PCM; Masters, 1982) zu entsprechen. Es wird zudem ein 2PL-Modell (Birnbaum, 1968) oder ein Generalized-Partial-Credit-Modell (GPCM; Muraki, 1992) geschätzt, um den Fit des Rasch-Modells bzw. PCMs zu testen. Welche Messmodelle verwendet werden, hängt von der Art der Items ab. Bei ausschließlich dichotomen Items wird das Rasch- und das 2PL-Modell berechnet. Wenn jedoch mindestens ein polytomes Item im Itemsatz enthalten ist, wird das PCM und GPCM berechnet.

### Linking

Häufig sollen Kompetenzerhebungen mit früheren Kompetenzerhebungen gelinkt werden, d.h. die Ergebnisse späterer Erhebungswellen werden mit denen früherer Erhebungswellen vergleichbar gemacht. Eine ausführliche Beschreibung des Linkingprozesses und dessen Grundlage werden im NEPS Survey Paper No. 1 (Fischer, Rohm, Gnambs, & Carstensen, 2016) beschrieben. Man findet es unter folgendem Link: <https://www.neps-data.de/Portals/0/Survey%20Papers/SP_I.pdf>. In dieser Version des Pakets ist das Linking jedoch noch nicht implementiert.

### Multistagetests

In neueren Kompetenzerhebungen werden häufig sogenannte Multistagetests eingesetzt. Multistagetests sind in mehrere Abschnitte geteilt, die jeweils unterschiedlich schwierige Itemblöcke enthalten können. Nach dem Prinzip des adaptiven Testens wird am Ende jedes Abschnitts die Kompetenz der Testpersonen automatisch geschätzt, um ihnen im darauffolgenden Abschnitt denjenigen Itemblock vorlegen zu können, der am besten zu ihren Fähigkeiten passt. Das NEPSroutines Paket ist größtenteils auch auf Multistagetests anwendbar, nur die Analyse fehlender Werte gestaltet sich etwas schwieriger. Wie man in so einem Fall vorgehen kann, wird im Abschnitt 6.4 *Analyse fehlender Werte in Multistagetests* genauer erklärt.

## Grundlagen in R

Für die Skalierungen und das Verständnis der vorliegenden Anleitung sind Grundkenntnisse in R nötig, z.B. das Laden von Paketen, das Einlesen von Dateien und grundlegende Datenmanipulationen wie die Auswahl von Daten und das Zusammenführen von Datensätzen.

Folgende Grundkenntnisse sind besonders wichtig für das Verständnis der Syntax:

* NA (= not available) als fehlender Wert
* Verschiedene Datentypen: logical, numeric, integer, double, factor, character
* for-Schleifen und if-clauses
* Logische Operatoren: &, &&, |, ||, !, ==, !=, <=, >=, <, >
* Der Operator %in%
* Die Pipe %>% aus dem Paket dplyr
* Aufruf einer Funktion, ohne das entsprechende Paket vorher laden zu müssen: paket::funktion()
* Verschiedenen Indizierungsarten für R-Objekte der Klasse data.frame. Zeilen könnnen über numerische Indizes oder über logische Abfragen ausgewählt werden. Für Spalten gilt dasselbe, zusätzlich können hier über String-Vektoren Variablennamen ausgewählt werden.
  + df[Zeilen, Spalten]
  + df[[Spalten]]

Wer die Grundlagen in R noch nicht gut genug beherrscht, findet viele Informationen über R im Internet (**Google** führt meist zu **stackoverflow**). Auch sehr empfehlenswert sind die sog. **Cheat Sheets** von R, in denen zu verschiedenen Themen die wichtigsten Informationen zusammengefasst werden. Diese Cheat Sheets findet man hier: <https://www.rstudio.com/resources/cheatsheets/>.

Schrittweises Ausführen der Syntax hilft bei der Fehlerbehandlung!

## Skalierung mit dem Paket NEPSroutines

Für die Skalierung wird das eigens für unsere Analysen erstellte Paket NEPSroutines benutzt. Beim Installieren und Laden des NEPSroutines Pakets werden automatisch alle abhängigen Pakete mit installiert und geladen. Weitere Pakete sollten also nicht manuell installiert werden müssen. Die Voraussetzung dafür ist aber, das Paket NEPSroutines am Anfang jeder R-Session zu laden. Sollten Fragen zu einzelnen Funktionen bestehen, kann Hilfe wie folgt aufgerufen werden:

help(FunktionName)  
?FunktionName

Das Paket wurde für den internen Gebrauch entwickelt und kann an der einen oder anderen Stelle noch zu Problemen führen. Sollten irgendwelche Fehler(meldungen) bei der Skalierung auftauchen oder euch Verbesserungsvorschläge einfallen, gerne an Jana Welling (jana.welling@lifbi.de) weiterleiten.

## Beispieldaten

Die Skalierungsanleitung orientiert sich am Beispiel der Studie B129 Lesen in der Startkohorte 2. Die entsprechenden Daten, den dazugehörige Technical Report (der aber noch auf einer alten Skalierung ohne das Paket NEPSroutines beruht) und ein Beispielskript findet man im Ordner der aktuellen Paketversion unter Z:/Projektgruppen\_(08)/Kompetenzen\_BA\_Hiwi\_(p000012)/Methoden/Skalierung/Skalierungspaket/Paketversionen/. Wenn du noch keinen Zugriff auf den Ordner haben solltest, wende dich bitte an uns.

Die Syntax kann durch Ersetzungen und Veränderung der Funktionsargumente an neue Studien angepasst werden.

# Vorbereitungen

## Vorbereitungen in R

Für die Skalierung sollte die neueste Version von R installiert werden (s. <https://cran.r-project.org/> für die neueste R-Version bzw. update.packages). Am einfachsten kann man mit R in RStudio arbeiten, das ist eine GUI (= graphical user interface) für die Statistik-Programmiersprache R. Wenn es nicht auf dem PC vorinstalliert sein sollte, kann man die neueste Version hier herunterladen: <https://www.rstudio.com/products/rstudio/>.

### Ein R-Projekt im Studienordner erstellen

Für jede Skalierung sollte ein eigenes R-Projekt im entsprechenden Studienordner auf dem Netzlaufwerk angelegt werden. Den übergreifenden Studienordner findest du unter //wipo.lifbi.de/daten/Projektgruppen\_(08)/Kompetenzen\_BA\_Hiwi\_(p000012)/Methoden/Skalierung/HE. Für jede Teilstudie (Lesen, Mathematik etc.) existiert wiederum ein Unterordner. Das Projekt wird im entsprechenden Unterordner angelegt. Das geht in RStudio über: **File → New Project… → New Directory → New Project → Create Project** (wo Name und Pfad angegeben werden müssen). Das Projekt sollte auf dem Netzlaufwerk des LIfBi erstellt werden, da nur hier die NEPS-Daten gespeichert und bearbeitet werden dürfen. Beachte jedoch, dass es bei einer Verbindung über VPN zu starken Verzögerungen kommen kann, wenn man umfangreiche Ergebnisse (das Problem sind meistens die Ergebnisdaten, nicht die Ergebnistabellen oder Plots) auf dem Netzlaufwerk abspeichern möchte. Sollte dieses Problem auftauchen, kann man im Paket das Speichern der Daten unterbinden. Nur im Notfall sollten Ergebnisse auch mal lokal abgespeichert werden. Diese müssen aber, sobald sie nicht mehr gebraucht werden, direkt und zuverlässig wieder gelöscht werden!

### Das Paket NEPSroutines installieren und laden

Das Paket NEPSroutines ist nicht als CRAN Paket erhältlich, sondern in einem tar.gz-Verzeichnis auf dem Netzlaufwerk gespeichert. Um es zu installieren, muss man folgenden Befehl eingeben:

install.packages(Pfad/NameVerzeichnis, type="source", repos=NULL)

Der Pfad richtet sich danach, wo das Verzeichnis zu finden ist. Die aktuellste Version befindet sich immer in diesem Ordner: Z:/Projektgruppen\_(08)/Kompetenzen\_BA\_Hiwi\_(p000012)/Methoden/Skalierung/Skalierungspaket/Paketversionen. Man kann das Paket natürlich auch in den Projektordner direkt kopieren, dann spart man sich die Eingabe eines Pfades und muss nur den Namen des Verzeichnisses selber angeben. Das Verzeichnis der Paketversion 0.1.0 heißt z.B. “NEPSroutines\_0.1.0.tar.gz”. Das aktuellste Verzeichnis ist immer das mit der höchsten Nummer.

Nachdem das Paket einmal installiert ist, kannst du es am Anfang jeder R-Session wie alle anderen Pakete auch über die library() Funktion laden:

library('NEPSroutines')

### RMarkdown

Für die Skalierung kann man RMarkdown verwenden (muss man aber natürlich nicht). In Rmd-Skripte können Text, Code und Output in einem Dokument zusammen erstellt und gespeichert werden. Wenn du RMarkdown verwendest, ist es wichtig, am Anfang unter den Optionen des Rmd-Skripts (klicke hierfür auf das Zahnrad am oberen Rand des Skriptes) die **Option “Chunk Output in Console”** auszuwählen, und nicht “Chunk Output Inline”. Ansonsten kann es sein, dass einzelne Ergebnisse der Skalierung nicht in R angezeigt werden.

### SessionInfo

Damit man auch im Nachhinein nachvollziehen kann, mit welcher Version von R und den verschiedenen R-Paketen die Skalierung durchgeführt wurde, sollte man sich diese Informationen vor oder nach der Skalierung ausgeben lassen. Das kann man über folgenden Befehl tun:

sessionInfo()

## Benötigte Daten und Dateien

Für die Skalierung benötigst du viele verschiedene Daten, die man meistens selber zusammensuchen und -setzen muss. In dieser Anleitung geben wir dir einen Überblick über diejenigen Daten, die man für die meisten Skalierungen braucht, und darüber, wo sie im Normalfall zu finden sind. Oft gibt es jedoch Ausnahmen, die dann im Einzelfall geklärt werden müssen.

Generell sollten die meisten Informationen im Studienordner über folgenden Pfad zu finden sein: //wipo.lifbi.de/daten/Projektgruppen\_(08)/Kompetenzen\_BA\_Hiwi\_(p000012)/Methoden/Skalierung/HE. Der Unterordner der entsprechenden Teilstudie enthält z.B. alle relevanten SPSS-Dateien und Codebooks im Ordner **rawdata** (ggf. zu finden im Ordner **preloads**).

Sollte die Datei mit den entsprechenden Informationen nicht vorliegen, einfach nachfragen!

### Kompetenzdaten

Die Kompetenzdaten sind in der SPSS-Datei mit xTargetCompetencies im Namen enthalten. Benötigt wird die ID-Variable (ID\_t) und alle Kompetenzitems. Die Namen der Kompetenzitems sind entweder dem Dokument zur entsprechenden Erhebung oder dem Codebook zu entnehmen. Zudem sollten nur die Daten der entsprechenden Welle behalten werden.

### Informationen über Kompetenzitems

Für die Skalierung werden zudem einige Informationen über die Kompetenzitems benötigt. Dies sind meistens: **Itemposition** (kann sich zwischen verschiedenen Testheften unterscheiden), **Aufgabentyp** (z.B. MC, CMC), **korrekte Antwort**, zu welchem **Testheft** die Items gehören, **Aufgabengruppe**. Je nach Erhebung sind aber noch weitere Variablen denkbar (z.B. Inhalt der Aufgabe). Diese Informationen muss man sich meistens aus Testheften und Dokumenten zum Studiendesign oder zur Erhebung selbst zusammensuchen und sie im besten Fall in eine Excel-Tabelle übertragen. Viele der Dokumente sind unter folgendem Ordner zu finden: //wipo.lifbi.de/daten/Projekte\_(07)/NEPS-Kompetenzen\_(14)/Netzwerk\_(05).

### Soziodemographische Daten

Zusätzlich zu den Kompetenzitems werden in den meisten Skalierungen, je nach Studienart, folgende soziodemographische Variablen für die deskriptive Statistik und die DIF-Analysen benötigt: **Geschlecht**, **Migrationshintergrund**, **kulturelles Kapital** (Anzahl Bücher Zuhause, als Teil des sozioökonomischen Status), **Schulform** (bei Schüler\*innen der Sekundarstufe), **Bildungsabschluss** (bei Erwachsenen), und **Alter** (bei Erwachsenen, da in Schulerhebungen durch den Klassenrahmen sehr homogen). Die entsprechenden Daten sind in den SPSS-Dateien CohortProfile und pParent (bei Kindern) bzw. pTarget (bei Erwachsenen) zu finden. Zuletzt waren die Variablen unter folgenden Namen zu finden (keine Garantie auf Richtigkeit):

* **Geschlecht**: tx80501 in CohortProfile
* **Alter**: wird berechnet (s. Abschnitt Vorbereitung der Daten) aus tx8050y (Geburtsjahr), tx8050m (Geburtsmonat) und dem Erhebungsjahr und -monat der entsprechenden Welle in CohortProfile
* **Migrationshintergrund**: z.B. p400500\_g1 in pParent (bei Kindern)
* **kulturelles Kapital**: z.B. p34005a in pParent (bei Kindern)

Alle anderen Variablen sind dem jeweiligen Codebook zu entnehmen.

### Informationen zum Testheftdesign

Zusätzlich zu den Kompetenz- und soziodemographischen Daten werden noch Informationen zur Erhebung und zu den Testheften benötigt. Häufig gibt es verschiedene **Testhefte** (z.B. leicht & schwierig), und/oder bei der Erhebung lag ein **Rotationsdesign** vor, d.h. mehrere Kompetenztests wurden verschiedenen Gruppen in unterschiedlicher Reihenfolge administriert (z.B. Gruppe 1: Lesen - Mathe, Gruppe 2: Mathe - Lesen). Die entsprechenden Informationen findet man in der SPSS-Datei xTargetCompetencies bei einer Variablen, deren Name mit tx beginnt und das Label “Befragungs-/Testinstrument” trägt. Die Werte sind kodiert, d.h. die Variable zeigt erstmal nur Zahlen an. Die entsprechenden Labels zu den Werten kann man entweder in SPSS bei der Variablenansicht sehen oder sich mit folgendem Befehl in R aufrufen lassen, wenn der Datensatz mit dem Paket haven (ist im Paket NEPSroutines enthalten) eingelesen wurde:

attributes(dat$name\_variable)$labels[attributes(dat$name\_variable)$labels %in% names(table(dat$name\_variable))]

Falls unklar sein sollte, welche Informationen des Erhebungsdesigns nun für die Skalierung gebraucht werden, kannst du auch immer nachfragen.

# Arbeiten mit dem Paket NEPSroutines

## Benötigte Datenstruktur

Die Arbeit mit dem NEPSroutines Paket setzt voraus, dass eine bestimmte Datenstruktur vorliegt. Diese soll im kommenden Abschnitt erläutert werden. Wie man die Rohdaten in die gewünschte Datenstruktur umwandeln kann und welche Funktionen des NEPSroutines Pakets dabei hilfreich sein können, wird in Kapitel 4 *Vorbereitung der Daten* erklärt.

Das Paket basiert hauptsächlich auf den zwei Datensätzen resp und vars. Die **fett markierten** Variablennamen müssen übernommen werden. In allen anderen Fällen sind die Namen der Variablen frei wählbar.

### Gruppierungen

Manchmal werden Teile der Skalierung für verschiedene Gruppen getrennt durchgeführt, z.B. bei verschiedenen Testheften, Settings, Messzeitpunkten oder Kohorten. Die Bezeichnungen *Gruppe* und *Gruppierung* beziehen sich auf solche Fälle. Die Funktionen des Pakets können aber immer nur eine Gruppierung gleichzeitig berücksichtigen. Sollten mehrere Gruppierungen vorliegen (z.B. verschiedene Testhefte *und* verschiedene Kohorten), gibt es zwei verschiedene Vorgehensweisen:

* Einerseits können die Analysen getrennt durchgeführt werden. Da die Funktionen des Pakets immerhin eine Gruppierung berücksichtigen können, reicht es, die Analysen getrennt für die Gruppen der zweiten Gruppierung durchzuführen (z.B. einmal für Kohorte A und einmal für Kohorte B). Die erste Gruppierung (in unserem Fall Testhefte) kann dann ganz normal in den Funktionen berücksichtigt werden. Denke in diesem Fall daran, den Analysen mit dem Funktionsargument name\_group einen Gruppennamen zu geben, damit die Ergebnisse mit unterschiedlichen Namen abgespeichert werden und eindeutig den Gruppen (Kohorte A, Kohorte B) zugeordnet werden können.
* Andererseits kann eine Gruppierungsvariable erstellt werden, die *alle* möglichen Gruppierungskombinationen enthält. Diese kann in den Funktionen dann ganz normal als Gruppierungsvariable berücksichtigt werden. Bei zwei Gruppierungen mit jeweils zwei Gruppen (z.B. Kohorten A und B, Testhefte A und B) müsste also eine neue Gruppierungsvariable mit insgesamt vier Ausprägungen erstellt werden:
  + Gruppe 1 für Kohorte A & Testheft A
  + Gruppe 2 für Kohorte A & Testheft B
  + Gruppe 3 für Kohorte B & Testheft A
  + Gruppe 4 für Kohorte B & Testheft B

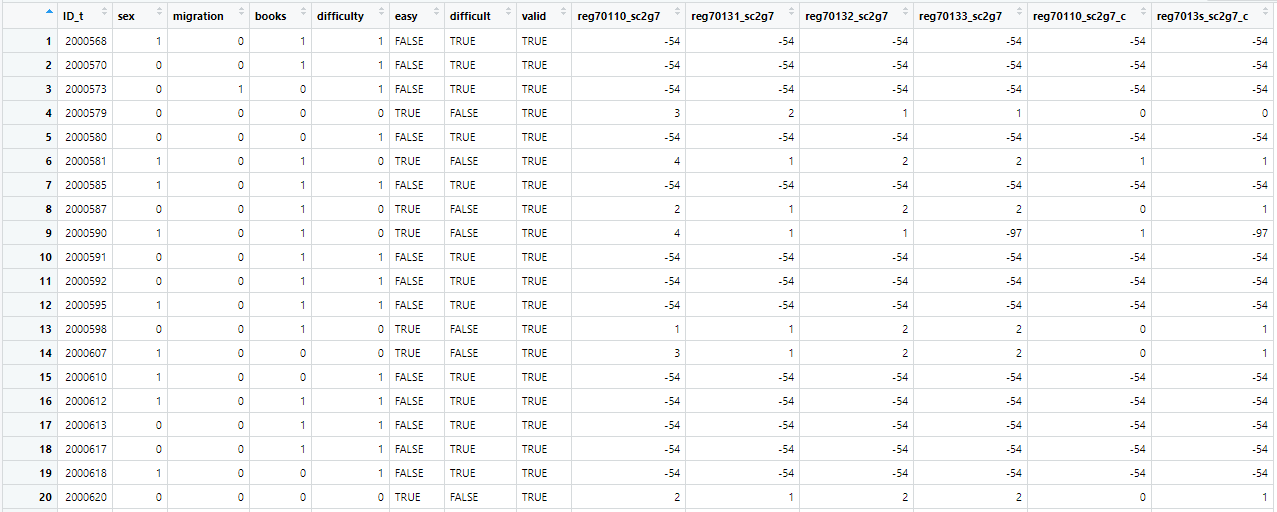
### Datensatz resp

Dieser Datensatz (data.frame) beinhaltet alle Kompetenzitems - gescort und ungescort - und alle Informationen über die Testpersonen, z.B. die soziodemographische Variablen. Die Informationen zu einer Testperson stehen jeweils in einer Zeile. Ein Beispiel für diesen Datensatz ist weiter unten für die Studie B129 zu sehen. Benötigt werden insbesondere folgende Variablen:

| **Variable** | **Verwendung** |
| --- | --- |
| **ID\_t** | Eine Variable, die die ID der Personen beinhaltet. |
| Alle soziodemographischen und methodischen Variablen, die pro Testperson vorliegen | * Die Variablen müssen numerisch sein. Alle in den DIF-Analysen verwendeten Variablen müssen dichotom y in {0, 1} sein. Nicht inhärent dichotome Variablen können dichotomisiert werden z.B. Einteilung der Anzahl Bücher Zuhause in ≥ 100 Bücher und < 100 Bücher. Variablen, die lediglich für die deskriptiven Analysen gebraucht werden z.B. Alter bei Schulkindern, können auch kontinuierlich sein. * Fehlende Werte werden als NA definiert * Für eine Übersicht über alle für die Skalierung benötigten Variablen s. Abschnitt 2 *Vorbereitungen*. |
| Alle ungescorten, gescorten und zusammengefassten Kompetenzitems | * Alle ungescorten Items können numerics, characters oder factors sein. * Alle gescorten Items müssen integers sein. Entweder dichotom y in {0, 1} oder bei polytomen Items mit k Kategorien y in {0, 1, …, k-1}. * Fehlende Werte werden mit negativen ganzen Zahlen kodiert, die zwischen verschiedenen Fällen unterscheiden z.B. -91 = test aborted, -55 = not administered. Diese Werte werden *user-defined missing values* genannt und sollten schon so in den Rohdaten enthalten sein. |
| Weitere Variablen, die bestimmte Personen im Datensatz identifizieren | * **Validitätsvariable** (z.B. valid): Identifiziert diejenigen Personen, die eine bestimmte Mindestanzahl an validen Antworten aufweisen (in unseren Skalierungen standardmäßig *drei* valide Antworten). Nur diese Personen sollten in den Analysen berücksichtigt werden.   + Die Variable muss eine logische Variable sein.   + TRUE für alle Personen mit Mindestanzahl an validen Antworten * **Gruppen- und Gruppierungsvariablen**, wenn eine oder mehrere Gruppierungen vorliegen   + Pro Gruppierung muss eine numerische oder Faktorvariable angelegt werden, die jeder Person eine Gruppe zuordnet (z.B. difficulty, mit 0=easy & 1=difficult)   + Pro Gruppe muss zusätzlich eine logische Variable angelegt werden (z.B. easy *und* difficult), die alle Personen identifiziert, die zu dieser Gruppe gehören   + Wenn es Items gibt, die beiden Gruppen administriert wurden, wird normalerweise auch eine IRT-Analyse für alle gemeinsamen Items durchgeführt. Dafür muss in resp eine Variable definiert werden (z.B. common), die TRUE für alle Personen ist, denen die gemeinsamen Items administriert wurden - also für alle Personen! * Im Einzelfall können weitere Variablen benötigt werden |

Beispiel für resp (aus der Studie B129, nur die ersten Variablen werden angezeigt):

| **Variable** | **Verwendung** |
| --- | --- |
| **ID\_t** | Enthält die IDs der Personen |
| sex, migration und books | Dichotome (bzw. dichotomisierte) soziodemographische Variablen für die deskriptiven und DIF-Analysen |
| difficulty | Gruppierungsvariable, die die Personen zu den verschiedenen Testheften zuteilt |
| easy und difficult | Gruppenvariablen, die diejenigen Personen identifizieren, die das leichte bzw. schwere Testheft erhalten haben |
| valid | Validitätsvariable, die alle Personen identifiziert, die mindestens drei valide Antworten gegeben haben |
| reg70110\_sc2g7 | Ungescortes einfaches Item mit user-defined missing values |
| reg70131\_sc2g7, reg70132\_sc2g7, reg70133\_sc2g7 | Ungescorte Subitems eines polytomen Items mit user-defined missing values |
| reg70110\_sc2g7\_c | Das zu reg70110\_sc2g7 gehörende dichotom gescorte Item (zu erkennen am Suffix \_c) mit user-defined missing values |
| reg7013s\_sc2g7\_c | Das zu den Subitems gehörende polytom gescorte Item (zu erkennen am s am Ende des Itemstamms und dem Suffix \_c) mit user-defined missing values |



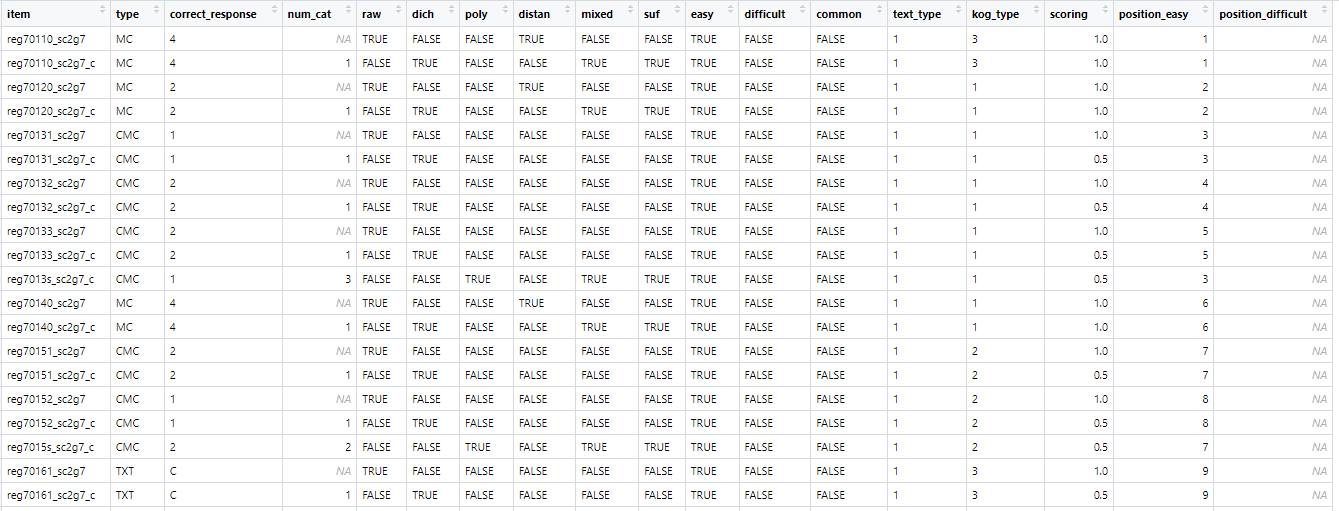
### Datensatz vars

Dieser Datensatz (data.frame) beinhaltet alle Informationen über die Kompetenzitems, z.B. Itempositionen, Zugehörigkeit zu einem Testheft und Itemtyp. Jede Version eines Kompetenzitems (z.B. gescort und ungescort) steht in einer eigenen Zeile (für eine Übersicht über die verschiedenen Itemtypen s. Abschnitt 1.1.2 *Kompetenzitems*). Ein Beispiel für den Datensatz vars ist weiter unten für die Studie B129 zu sehen. Wie die Variablen jeweils erstellt werden, kann dem Kapitel 4 *Vorbereitungen der Daten* entnommen werden. Benötigt werden insbesondere folgende Variablen:

| **Variable** | **Verwendung** |
| --- | --- |
| **item** | Eine Variable, die den Namen des Items beinhaltet |
| **correct\_response** | Eine Variable, die die richtige(n) Antwort(en) des jeweiligen Items enthält (z.B. “3” wenn die dritte Option die richtige ist). Kann ein character oder numerischer Vektor sein. Bei Items, für die es mehr als eine richtige Antwortmöglichkeit gibt, müssen die richtigen Antworten in einem String und getrennt durch ein Satzzeichen dargestellt werden (z.B. „3;4“ wenn die dritte und vierte Option richtig sind). |
| type | Eine Variable, die den Itemtyp (“MC”, “CMC”, etc.) definiert |
| num\_cat | Eine numerische Variable, die die Anzahl der Antwortkategorien angibt. Bei dichotomen Items ist das immer 1, bei polytomen Items die Anzahl der Subitems. |
| Positionsvariablen | Gibt die Position des jeweiligen Items im Testheft an   * Falls es mehrere Gruppen gibt, eine Positionsvariable pro Gruppe, auch wenn die Position in allen Gruppen gleich ist * Ansonsten nur eine Positionsvariable |
| Logische Variablen, die bestimmte Items identifizieren | * raw: TRUE für alle *ungescorten* Items (Rohdaten) * distan: TRUE für alle *ungescorten* *Multiple-Choice* Items (wird für die Distraktorenanalyse benötigt) * **dich**: TRUE für alle *dichotom gescorten* Items (wird z.B. für die dichotomen IRT- und DIF-Analysen benötigt). Dazu zählen auch Subitems von polytomen Items! * Wenn das Itemset auch polytome Items enthält:   + poly: TRUE für alle *polytom gescorten* Items (wird für die Datenvorbereitung benötigt)   + mixed: TRUE für alle in der Skalierung zu verwendenden Items (wird z.B. für die polytomen IRT- und DIF-Analysen benötigt). Das sind zunächst alle *dichotom gescorten* Items *ohne* Subitems von polytomen Items und alle *polytom gescorten* Items (falls ein Item kollabiert wurde, dann die Version *mit* kollabierten Antwortkategorien). * Wenn sich das Itemset im Laufe der Skalierung ändert, da z.B. bestimmte Items ausgeschlossen werden, müssen neue Variable angelegt werden (z.B. dich\_new oder mixed\_new), die die oben genannten Variablen für das neue Itemset ersetzen (wird für alle Analysen benötigt) * suf: TRUE für alle *gescorten* Items ohne kollabierte Antwortkategorien, die tatsächlich im Test verwendet wurden (wird für die Berechnung der Summen- und Metakognitionscores sowie die Erstellung des SUF benötigt) |
| Je nach Skalierung weitere Variablen | * **Gruppenvariablen**, wenn nicht alle Items in jeder Gruppe präsentiert wurden (z.B. haben verschiedene Testhefte meistens auch unterschiedliche Itemsets)   + Gruppenvariablen sind immer logische Variablen   + Pro Gruppe eine Variable (z.B. die Variablen easy und difficult wenn ein leichtes und ein schweres Testheft vorliegen)   + Wenn es Items gibt, die von allen Gruppen bearbeitet wurden, muss eine zusätzliche Variable (z.B. common) angelegt werden, die diese Items identifziert   + Pro Gruppenvariable (z.B. easy, difficult & common) und Analysevariable (z.B. dich, mixed & suf) muss zudem eine neue, gruppenspezifische Analysevariable angelegt werden (z.B. die Variable dich\_easy, die alle dichotom gescorten Items des leichten Testhefts identifiziert) * **Scoringvariablen**, wenn polytome Items im Test enthalten sind (z.B. scoring). Diese braucht man zur Berechnung des PCM und GPCM. Das Scoring eines Items gibt an, mit welchem Gewicht das jeweilige Item in die Analyse einfließen soll. Standardmäßig ist dies bei polytomen Items 0.5 und bei dichotomen Items 1. Bei ursprünglich polytomen Items, die durch Zusammenlegen von Antwortkategorien nur noch zwei Ausprägungen besitzen, sollte das Scoring auch auf 1 festgelegt werden. Im Einzelfall kann von diesen Vorgaben aber abgewichen werden. * **Dimensionsvariablen**, wenn im Test Strukturen vorliegen, die die Eindimensionalität des Tests infrage stellen könnten. Dies ist z.B. bei Lesetests der Fall, die aus mehreren Aufgabengruppen bestehen, die jeweils einem Text aber mehrere Items beinhalten * **Linkingvariablen**, wenn anschließend an die Skalierung ein Linking durchgeführt werden soll. Welche Variablen im einzelnen benötigt werden, hängt von der Art des Linkings und der Daten ab (siehe hierzu Survey Paper No. 1 Fischer et al., 2016; Link s.o.) * Im Einzelfall können weitere Variablen benötigt werden |

Ein Beispiel für vars (aus der Studie B129, nur die ersten Variablen werden angezeigt):

| **Argument** | **Verwendung** |
| --- | --- |
| item | Enthält die Namen der Items |
| type, num\_cat und correct\_response | Variablen, die den Aufgabentyp, die Anzahl der Subitems bzw. die korrekte Antwort enthalten |
| raw , dich und poly | Logische Variablen, die jeweils die ungescorten, dichotom gescorten und polytom gescorten Items identifizieren |
| distan | Logische Variable, die alle für die Distraktorenanalyse benötigten Items identifizeirt (ungescorte Multiple-Choice Items) |
| mixed | Identifiziert als logische Variable alle Items, die in die IRT-Analysen einfließen sollen. Enthalten sind alle inhärent dichotomen Items und alle polytomen Items. Wenn im Laufe der Skalierung Items aus der Analyse ausgeschlossen oder zusammengefasst werden, kann diese Variable aktualisiert oder eine neue Variable (z.B. mixed\_new) erstellt werden |
| easy, difficult und common | Logische Gruppenvariablen, die diejenigen Items identifizieren, die im leichten, schweren oder beiden Testheften enthalten sind |
| suf | Logische Variable, die alle Items identifiziert, die zur Erstellung der Scores verwendet werden und in den SUF kommen |
| text\_type und kog\_type | Variablen für die Dimensionalitätsanalysen |
| scoring | Scoringvariable, die für das PCM und GPCM benötigt wird |
| position\_easy und position\_difficult | Positionsvariablen, die für beide Testhefte getrennt die jeweilige Position des Items im Testheft angeben |
| dich\_easy,  dich\_difficult, dich\_common | Logische Variablen, die alle dichotom gescorten Items aus dem leichten, schweren oder beiden Testheften identifizieren (nicht im Screenshot zu sehen) |
| mixed\_easy,  mixed\_difficult, mixed\_common | Logische Variablen, die alle polytom gescorten Items und inhärent dichotomen Items aus dem leichten, schweren oder beiden Testheften identifizieren (nicht im Screenshot zu sehen) |
| suf\_easy und suf\_difficult | Logische Variablen, die alle für die Scores und den Suf benötigten Items aus dem leichten und schweren Testheft identifizieren (nicht im Screenshot zu sehen) |



### Beispieldaten

Die Beispieldaten können auch in R geladen werden (setze für X die aktuelle Paketversion ein):

load("Z:/Projektgruppen\_(08)/Kompetenzen\_BA\_Hiwi\_(p000012)/Methoden/Skalierung/Skalierungspaket/Paketversionen/VersionX/B129.Rdata")

## Aufbau des Pakets

Das Paket ist stark an die Skalierung der Kompetenztests des NEPS orientiert und beinhaltet Funktionen zu folgenden Abschnitten des Skalierungsprozesses:

* Datenvorbereitung
* Deskriptive Analysen
* Analyse fehlender Werte (pro Person & pro Item)
* IRT-Analysen
* DIF-Analysen
* Distraktorenanalyse
* Dimensionalitätsanalyse
* Linking
* Scores berechnen
* SUF erstellen

Zudem enthält das Paket allgemeine Funktionen, die bei der Skalierung außerhalb des vorgegebenen Rahmens hilfreich sein können. Mit Ausnahme der Datenvorbereitung und der deskriptiven Analysen gibt es pro Skalierungsabschnitt eine **übergreifende Funktion**, die automatisiert alle notwendigen Berechnungen durchführt und die Ergebnisse in Plots, Tabellen und Messages ausgibt. Es existieren jedoch auch immer **untergeordnete Funktionen**, mit denen man einzelne Schritte des Abschnitts durchführen und so die ursprünglichen Analysen nachbessern oder eher an die eigenen Bedürfnissen anpassen kann. In der vorliegenden Anleitung werden jeweils die übergreifenden Funktionen genauer vorgestellt und die untergeordneten Funktionen mit einer kurzen Erklärung erwähnt.

### Output der Funktionen

Der Output der übergreifenden Funktionen besteht meistens wahlweise aus den zusammengefassten Ergebnissen als Excel-Tabelle (Funktionsargument save=TRUE), den originalen Ergebnissen im RDS-Format (Funktionsargument save=TRUE) und zusätzlichen Informationen, die in der Konsole ausgegeben werden (Funktionsargument print=TRUE). Je nach Funktion können auch Plots erstellt werden (Funktionsargument plots=TRUE). Die Tabellen sind meistens so gestaltet, dass sie direkt für den TR verwendet werden können. Die originalen Ergebnisse werden meistens in einer Liste gespeichert, die den Output der verwendeten Funktionen (z.B. TAM-Funktionen) unverändert enthält. Man kann sich diese Ergebnisse auch immer direkt in R ausgeben lassen (Funktionsargument return=TRUE). Wenn man sie in einem R-Objekt speichert (indem man ihnen einen Namen zuweist), kann man mit ihnen direkt weiterarbeiten. Zum Beispiel:

irt\_results <- NEPSroutines::irt\_analysis(…)

Bei untergeordneten Funktionen, die Analysen durchführen, wird die Liste mit den Ergebnissen immer ausgegeben. Der Output oder Teile des Outputs werden meistens für darauf aufbauende Funktionen (z.B. Plots oder Ergebnistabelle erstellen lassen) benötigt. Es ist also sinnvoll, den Output in einem R-Objekt abzuspeichern (s.o.). Achte bei der Verwendung von weiterführenden Funktionen genau auf die Angaben in dieser Anleitung und in den Hilfefunktionen (s. Abschnitt 1.3 *Grundlagen in R*), welcher Output bzw. Teil eines Outputs tatsächlich verlangt wird! In untergeordneten Funktionen, die mit den Ergebnissen Plots oder Tabellen erstellen, kann man sich aussuchen, ob diese Dateien gespeichert werden sollen oder nicht (Funktionsargument save=TRUE).

## Häufige Funktionsargumente

Die Funktionen des Pakets sind teilweise sehr ähnlich aufgebaut, sodass sich viele Funktionsargumente wiederholen. Die häufigsten Argumente werden in diesem Abschnitt vorgestellt, aber in der restlichen Anleitung nicht mehr näher erläutert.

**Input**-Argumente:

| **Argument** | **Art** | **Beschreibung** | **Default** |
| --- | --- | --- | --- |
| resp und vars | data. frame | Diese Funktionsargumente erfordern die im vorherigen Abschnitt vorgestellten data.frames resp und vars. Sie können so wie sie sind übergeben werden und müssen nicht mehr verändert werden. |  |
| select | string | Die Analysen basieren fast immer auf nur einem Teil der vorhandenen Items (oder nur auf einer bestimmten Version der Items). Dieses Funktionsargument erfordert den Namen der logischen Variablen in vars, die die für diese Analyse benötigten Variablen identifiziert (z.B. dich bei nur dichotomen Items oder mixed bei dichotomen *und* polytomen Items). |  |
| valid | string | Optional in den meisten Funktionen. Dieses Funktionsargument erfordert den Namen der logischen Variablen in resp, die alle validen Fälle identifiziert (s. Abschnitt 4.3.1 *Valide Fälle*). | alle Fälle werden  verwendet |
| mvs | named  character vector | Optional in den meisten Funktionen (Ausnahme: Analyse fehlender Werte). Erfordert einen named character vector, der alle user-defined missing values enthält (s. Abschnitt 4.1.4 *User defined missing values*). **Vorsicht**: Sollte dieser Vektor nicht vollständig sein, können gravierende Probleme und Fehler in der Skalierung auftreten! | -999 bis -1 |
| missing\_by\_design | integer | User-defined missing value für Antworten missing by design (z.B. wenn manche Items nicht allen Personen administriert wurden). In manchen Analysen (v.a. IRT und fehlende Werte) werden diese Antworten gefiltert, daher ist die korrekte Angabe sehr wichtig! Der Wert muss trotzdem auch im Argument mvs enthalten sein. | -54 |
| grouping | character vector | Optional in manchen Funktionen. Manchmal müssen Teile der Skalierung für verschiedene Gruppen getrennt durchgeführt werden (z.B. bei verschiedenen Testheften). In diesem Fall erfordert dieses Funktionsargument einen character vector mit den Namen der logischen Variablen in resp und vars, die alle Personen bzw. Items einer Gruppe identifizieren. Pro Gruppe wird ein Name angegeben. Die Variablennamen müssen zwischen resp und vars übereinstimmen (z.B. identifiziert die Variable easy im Datensatz vars alle Items, die im leichten Testheft administriert wurden, und die Variable easy im Datensatz resp alle Testpersonen, die das leichte Testheft bearbeitet haben). | NULL |

**Prozess**-Argumente:

| **Argument** | **Art** | **Beschreibung** | **Default** |
| --- | --- | --- | --- |
| digits | integer | Dezimalstelle, auf die die Ergebnisse gerundet werden sollen. | meistens 3 |
| warn | logical | Ob Warnungen in die Konsole geschrieben werden sollen. | meistens TRUE |
| verbose | logical | Ob der Bearbeitungsprozess in die Konsole geschrieben werden soll. | meistens TRUE |

**Output**-Argumente:

| **Argument** | **Art** | **Beschreibung** | **Default** |
| --- | --- | --- | --- |
| plots | logical | In übergreifenden Funktionen kann man sich entscheiden, ob man sich die Plots der Analysen (z.B bei IRT-Analysen ICC-Plots und Wright Map) ausgeben lassen möchte. | meistens FALSE |
| print | logical | In übergreifenden Funktionen kann man sich entscheiden, ob man die (zusätzlichen) Ergebnisse in die Konsole bzw. das HTML-Dokument schreiben lassen möchte. | meistens TRUE |
| save | logical | In fast allen Funktionen kann man sich entscheiden, ob man die Ergebnisse (als Excel und/oder RDS) speichern lassen möchte. | meistens TRUE |
| return | logical | In übergreifenden Funktionen kann man sich entscheiden, ob man sich die Ergebnisse (als Rdata) in R ausgeben lassen möchte. Das ist vor allem sinnvoll, wenn man mit den Ergebnissen direkt in R weiter arbeiten möchte. | meistens FALSE |
| path | string | Pfad, an dem die Ergebnisse gespeichert werden sollen. Wenn der angegebene Ordner nicht existiert, wird er neu erstellt. Bei übergreifenden Funktionen in path\_results, path\_plots und path\_table unterteilt. | “Results”, “Plots”, “Tables” |
| overwrite | logical | Ob die Excel-Tabellen überschrieben werden sollen, wenn sie schon existieren. | FALSE |
| name\_group | string | Bezeichnung der *Gruppe*, auf die sich die Analysen beziehen (z.B. “easy”, s. 3.1.1. *Gruppierungen*), um die Ergebnisse, Tabellen und Grafiken entsprechend zu benennen. Nicht zu verwechseln mit name\_grouping (= Name der *Gruppierung*, z.B. “test version”) in mv\_person und mv\_item! | NULL |
| suf\_item\_name | logical | Ob die Namen von polytomen Items mit kollabierten Kategorien in Excel-Tabellen ohne das Suffix \_‘collapsed‘ ausgegeben werden sollen. Per Default (FALSE) werden die Namen solcher Items mit dem Suffix ‚\_collapsed‘ ausgegeben. | FALSE |

In einigen Funktionen kommen zusätzlich spezifische Argumente vor. Diese werden in der vorliegenden Anleitung beschrieben und sind auch jederzeit über die R-Dokumentation (mittels Hilfefunktionen, s. Abschnitt 1.3 *Grundlagen in R*) abrufbar.

## Hilfreiche allgemeine Funktionen

Wenn man Teile der Skalierung ohne die entsprechenden Funktionen des Pakets durchführt, können einige der folgenden Funktionen hilfreich sein.

| **Funktion** | **Verwendung** |
| --- | --- |
| convert\_mv | Konvertiert die user-defined missing values in NAs. Input-Argumente sind resp, items und mvs. Ausgegeben wird resp mit den konvertierten fehlenden Werten. |
| prepare\_resp | Bereitet resp für kommende Analysen vor. Input-Argumente sind resp, vars, items, valid, mvs, der logical use\_only\_valid (ob nur valide Fälle benutzt werden sollen) und der logical convert (ob user-defined missing values in NAs konvertiert werden sollen). Ausgegeben wird resp mit nur noch den in items definierten Items als Variablen, ggf. nur den validen Fällen und ggf. in NAs konvertierten fehlenden Werten. |
| check\_folder | Prüft, ob der im Pfad angegebene Ordner existiert und wenn nicht, erstellt er ihn neu. |
| meht | Führt den Minimum Effect Hypothesis Test durch. Input-Argumente sind verschiedene statistische Größen. |

## Debugging

Da jede Skalierung ein bisschen anders ist, kann es gut sein, dass bei der Benutzung des Pakets NEPSroutines Fehler auftreten. Häufig sind aber auch einfach Fehler in den Daten(-sätzen) die Ursache für Fehlermeldungen. Falls eine Funktion also eine Fehlermeldung ausgeben sollte, prüfe zuerst, ob deine Daten auch korrekt sind: Sind im Datensatz vars alle Itemeigenschaften korrekt definiert? Kommt jedes Item aus vars auch in resp vor und vice versa? Stimmen die Klassen der Variablen mit den Vorgaben überein? Wurden die Variablen richtig benannt? Die benötigte Datenstruktur wird in 3.3.1 *Benötigte Datenstruktur* genau beschrieben und kann teilweise mit einigen dafür entwickelten Funktionen überprüft werden, s. 4.3.3 *Check der Datenstruktur*.

Falls die Daten korrekt sein sollten und der Fehler trotzdem noch auftritt, kannst du dich mit der Fehlermeldung, dem Skript und den Daten an [jana.welling@lifbi.de](mailto:jana.welling@lifbi.de) wenden.

# Vorbereitung der Daten

## Vorbereitung der Kompetenzitems

### Laden der Daten und erste Anpassungen

Als Erstes müssen die benötigten Daten geladen werden. Zunächst brauchen wir bestimmte Informationen über die Kompetenzitems (z.B. korrekte Antwort, Itemtyp, Position, Testheft; s Abschnitt 2.2. *Benötigte Daten und Dateien*). Diese sollten im Vorfeld für jedes ungescorte Item rausgesucht und in einer Excel-Tabelle zusammengetragen werden, sodass sie nun als data.frame vars in die R Environment geladen werden können. Beachte hierbei die Namenskonventionen des Pakets NEPSroutines für die Variablen im data.frame vars (s. Abschnitt 3.1. *Datenstruktur des Pakets NEPSroutines*). Welche Items im Kompetenztest enthalten sind, kann man entweder im Codebook oder in einem Dokument zum entsprechenden Test nachlesen (s. Abschnitt 2.2 *Benötigte Daten und Dateien*).

vars <- openxlsx::read.xlsx(file = "path/filename.xlsx")

Die Rohdaten der Kompetenzitems findet man im SPSS-Datensatz xTargetCompetencies. Dieser wird mit der Funktion read\_sav() als data.frame in die R Environment geladen und resp genannt. Das Argument use\_na = FALSE verhindert, dass in SPSS spezifizierte fehlende Werte als NA importiert werden. Bei sehr großen Datensätzen kann es zu Speicherplatzproblemen kommen; hier kann evtl. der Gebrauch von foreign::read.spss helfen.

resp <- haven::read\_sav(file = "path/filename.sav", use\_na = FALSE)

Wir benötigen nur diejenigen Items, die im zu skalierenden Kompetenztest vorkamen, und für die DIF-Analysen die Variable, die das Testheftdesign angibt (tx… , s. Abschnitt 2.2.4). Diese kann man mit Hilfe der Funktion select() auswählen.

item\_names <- vars$item  
design <- "NameVariableTestheftdesign"  
resp <- dplyr::select(resp, ID\_t, design, all\_of(item\_names))

Das Paket NEPSroutines setzt außerdem logische Variablen im Datensatz voraus, die Informationen über die Version des Items geben (raw, dich, poly). Da bisher nur ungescorte Items vorliegen, ist die Zuordnung einfach und kann für alle Items gleichzeitig vorgenommen werden. Beachte, dass die Variable poly nur gebraucht wird, wenn im Test auch polytome Items vorkamen.

vars$raw <- TRUE  
vars$dich <- FALSE  
vars$poly <- FALSE

### Dichotomes Item Scoring

Im nächsten Schritt müssen die ungescorten Items gescort werden. Es wird also pro Item eine neue Variable erstellt, die angibt, ob das Item richtig (1) oder falsch (0) beantwortet wurde. Dazu kann die Funktion dichotomous\_scoring() verwendet werden. Als Input übergibt man an die Funktion die data.frames resp und vars und den character vector old\_names mit den Namen der ungescorten Items. Man kann prinzipiell zusätzlich den character vector new\_names mit den gewünschten Namen für die gescorten Items an die Funktion übergeben. Wenn die Skalierung jedoch für eine Erhebung des NEPS durchgeführt wird, braucht man das nicht, da die Namensgebungskonvention des NEPS (Name des ungescorten Items + Suffix “**\_c**”) als Default in der Funktion hinterlegt ist. Für die Funktion muss die Variable correct\_response in vars definiert sein. Falls mehr als eine Antwortoption pro Item richtig sei kann, übergibt man dem Funktionsargument sep als character das Satzzeichen, mit dem die korrekten Antworten in dieser Variable getrennt sind (s. 3.1.3 *Datensatz vars*, Default ist „;“). Die Funktion ergänzt den data.frame resp mit den neuen Items.

resp <- NEPSroutines::dichotomous\_scoring(resp, vars, old\_names = item\_names)

Im Paket NEPSroutines ist es notwendig, für jede Version eines Items (z.B. ungescort, gescort) eine eigene Zeile im data.frame vars anzulegen. Da die meisten Informationen der ungescorten und gescorten Items identisch sind, kann man dafür die Funktion duplicate\_items() anwenden. Diese dupliziert die Iteminformationen für eine beliebige Anzahl an Items, und nimmt nur explizit gewünschte Veränderungen vor (z.B. Itemname). Als Input übergibt man der Funktion den data.frame vars, den character vector old\_names mit den Namen der ursprünglichen Items, den character vector new\_names mit den Namen der neuen Items und optional die Liste change, wenn man außer den Itemnamen auch andere Informationen ändern möchte. Als Name eines Listenelements verwendet man dann den Namen der Variable, die sich zwischen den alten und den neuen Items unterscheidet. Der Inhalt jedes Listenelements ist dann entweder ein einziger Wert, der für alle neuen Items verwendet wird, oder ein Vektor mit einem Wert pro neuem Item. Die Funktion gibt den data.frame vars mit den neuen Items aus. Da sich zwischen den ungescorten und gescorten Items nur die Variablen raw und dich unterscheiden sollten, sieht der Funktionsaufruf dann folgendermaßen aus.

vars <- NEPSroutines::duplicate\_items(  
 vars,  
 old\_names = item\_names,  
 new\_names = paste0(item\_names, "\_c"),  
 change = list(raw = FALSE, dich = TRUE)  
)

### Polytomes Item Scoring

Wenn im Kompetenztest polytome Items enthalten sind, müssen diese auch gescort werden. Dazu kann die Funktion pc\_scoring() verwendet werden. Als Input übergibt man der Funktion den data.frame resp und die Liste poly\_items mit einem Element pro polytomen Item. Die Namen der Elemente entsprechen den Namen der polytomen Items und sie enthalten jeweils einen character vector mit den Namen aller Subitems. Im NEPS besteht der Name eines polytomen Items prinzipiell aus den Namen der Subitems. Lediglich die Ziffer, die sich zwischen den Items unterscheidet, wird mit einem “**s**” ersetzt. Die Funktion ergänzt den data.frame resp mit den neuen Items.

Vor dem Scoring führt die Funktion pc\_scoring() standardmäßig eine Imputation fehlender Werte auf den Subitems polytomer Items durch – vorausgesetzt, mindestens 50% der Subitems eines polytomen Items enthalten gültige Antworten. Diese Funktionalität ist per Default über die Argumente impute = TRUE und threshold = .50 aktiviert. Die Imputation basiert auf einem Rasch-Modell: Anhand der geschätzten Item- und Personenparameter (EAP-Werte) wird für jede Person die Lösungswahrscheinlichkeit pro Subitem berechnet und daraus eine individuelle Antwort vorhergesagt. Für die Imputation werden die Argumente vars und select benötigt. vars bezeichnet einen Datensatz mit Informationen über die Kompetenzitems. Mit dem Argument select wird eine logische Variable innerhalb von vars angegeben, die alle gescorten dichotomen Items identifizieren (z.B. select = "dich"). Das schließt sowohl Multiple-Choice-Items als auch Subitems polytomer Items ein. Die imputierten Werte werden ausschließlich für das Scoring der polytomen Items verwendet. Im zurückgegebenen Datensatz bleiben die ursprünglichen (nicht-imputierten) Subitem-Antworten unverändert. Informationen zur Imputation werden automatisch im Excel- und RDS-Format gespeichert.

poly\_items <- list(  
 BeispielItem1s\_c = paste0("BeispielItem1", c(1:3), "\_c"),  
 BeispielItem2s\_c = paste0("BeispielItem2", c(1:4), "\_c"),  
 etc.  
)  
resp <- NEPSroutines::pc\_scoring(resp, poly\_items = poly\_items,  
 vars = vars, select = "dich")

Wie auch beim dichotomen Item Scoring müssen auch die polytomen Items im data.frame vars hinterlegt werden. Dies kann abermals mit der Funktion duplicate\_items() durchgeführt werden. Dazu wird jeweils ein Subitem als Referenzitem verwendet.

first\_subitem <- sapply(poly\_items, function(x) x[1])  
vars <- NEPSroutines::duplicate\_items(  
 vars,  
 old\_names = first\_subitem,  
 new\_names = names(poly\_items),  
 change = list(raw = FALSE, dich = FALSE, poly = TRUE)  
)

Sind einzelne PCM-Kategorien spärlich besetzt, d.h. die Zellhäufigkeit in einer Häufigkeitstabelle liegt unter einem bestimmten Wert (im NEPS 200), muss diese mit der nächstunteren Kategorie zusammengelegt werden. Ist z.B. die Kategorie 0 zu spärlich besetzt, wird sie entsprechend mit der nächsthöheren Kategorie zusammengelegt. Das überprüft die Funktion collapse\_response\_categories() und setzt es im gegebenen Fall automatisch um. Man übergibt ihr den data.frame resp und mit select den Namen der Variablen in vars, die alle polytomen Items identifiziert. Der Default-Wert für die spärliche Zellbesetzung liegt bei 200, kann mit dem Funktionsargument per\_cat aber geändert werden. Die Funktion gibt den data.frame resp aus, der alle *originalen* Variablen und zusätzlich alle *kollabierten* Items enthält. Das heißt für jedes Item mit spärlichen Antwortkategorien sind nun zwei Variablen im Datensatz - das originale Item und ein neues Item mit kollabierten Antwortkategorien. Die Namen der neuen Items enden mit dem Suffix \_collapsed. Die Funktion gibt eine Liste mit den Namen aller veränderten Items in der Konsole aus. Wenn save=TRUE, dann wird diese Liste auch als Excel-Tabelle gespeichert. Für den seltenen Fall, dass durch das Zusammenlegen bei einem Item weniger als zwei Kategorien übrig bleiben würden, sieht die Funktion vom Zusammenlegen ab und gibt eine Warnung mit dem Namen des Items in der Konsole aus. Dieses Item sollte dann manuell zusammengelegt werden. Denke daran, dass die Namen der manuell kollabierten Items ebenfalls zur Liste der kollabierten Items hinzugefügt werden sollten.

resp <- NEPSroutines::collapse\_response\_categories(resp, vars, select = "poly")

Wurden Items kollabiert, müssen die neuen Items im data.frame vars hinterlegt werden. Die Namen der alten (mit dem Suffix \_c) und der neuen, kollabierten Items (mit dem Suffix \_collapsed) können der Konsole oder der Excel-Tabelle entnommen werden. Dazu kann erneut die Funktion duplicate\_items() verwendet werden. Beachte, dass die neuen Items auch als polytome Items gespeichert werden (poly=TRUE), wenn sie nur noch zwei Antwortkategorien besitzen.

original\_items <- c("original.item\_1", "original.item\_2")  
collapsed\_items <- c("collapsed.item\_1", "collapsed.item\_2")  
vars <- NEPSroutines::duplicate\_items(  
 vars,  
 old\_names = original\_items,  
 new\_names = collapsed\_items  
)

Wenn in einem Kompetenztest sowohl einfache als auch polytome Items vorkommen, erfolgt der größte Teil der Skalierung anhand des Itemsatzes, der aus den polytomen Items und den einfachen gescorten Items besteht. Die Subitems der polytomen Items werden jedoch nicht verwendet. Um diesen Itemsatz in den nachfolgenden Analysen verwenden zu können, wird nun eine logische Variable erstellt, die die entsprechenden Items identifiziert. In unserem Beispiel heißt die Variable mixed. Beachte, dass die Items, für die ein kollabiertes Item gebildet wurde, von den Analysen ausgeschlossen werden sollen.

vars$mixed <- ifelse(  
 vars$poly, TRUE,  
 ifelse(vars$dich & !vars$item %in% unlist(poly\_items), TRUE, FALSE)  
)  
vars$mixed[vars$item %in% original\_items] <- FALSE

Da nach Zusammenfassung der polytomen Items die Itemposition nun Lücken aufweist (z.B. wenn vorher Subitems 1 bis 4 auf Positionen 1 bis 4 waren, existiert anschließend nur noch ein polytomes Item auf Position 1), soll pro Gruppe eine neue Positionsvariable ohne Lücken erstellt werden. Dazu kann die Funktion pos\_new() verwendet werden. Als Input übergibt man der Funktion vars, items und den (named) character vector position, der bei mehreren Gruppen die Namen der Gruppenpositionsvariablen enthält (die Namen der Vektorelemente entsprechen den Namen der Gruppenvariablen) und bei nur einer Gruppe den Namen der einzigen Positionsvariablen enthält. Die Funktion gibt den data.frame vars mit den neuen Positionsvariablen aus.

vars <- NEPSroutines::pos\_new(  
 vars,  
 select = 'mixed',  
 position = c(easy = 'position\_easy', difficult = 'position\_difficult')  
)

Zuletzt muss nun noch die Scoring-Variable scoring erstellt werden, die für polytome IRT- und DIF-Analysen benötigt wird. Standardmäßig werden im NEPS dichotome Items mit 1 und polytome Items mit 0.5 gescort. Diejenigen Items, die eigentlich polytom sind, aber durch das Zusammenlegen der Antwortkategorien dichotom geworden sind, werden ebenfalls mit 1 gescort. Im Einzelfall kann davon aber abgewichen werden.

vars$scoring[vars$mixed] <- ifelse(

apply(resp[ , vars$item[vars$mixed]], 2, max, na.rm = TRUE) > 1,

0.5, 1

)

### User defined missing values

Bei den Kompetenzitems des NEPS liegen sog. user defined missing values (UDMV) vor. Fehlende Werte werden also mit ganzen, negativen Zahlen kodiert, um so verschiedene Arten fehlender Werte unterscheiden zu können. Im NEPS gibt es folgende UDMV:

* **-99**: filtered
* **-97**: omitted (OM)
* **-95**: not valid (NV)
* **-94**: not reached (NR)
* **-93**: trifft nicht zu
* **-91**: test aborted (TA)
* **-90**: missing without specified reason (UM)
* **-56**: not participated
* **-55**: not determinable (ND)
* **-54**: missing by design (MD)
* **-22**: übersprungen
* **-21**: Angabe zurückgesetzt (AZ)

Nicht in jedem Kompetenztest kommt jede Art fehlender Werte vor. Für die Skalierung sollte man also zunächst überprüfen, welche denn überhaupt vorliegen. Dies geht mit der table Funktion.

table(unlist(resp[ , vars$item]))

Die vorhandenen UDMV werden nun in einem named integer vector definiert. Die fehlenden Werte können beliebig benannt werden, eine Abkürzung ist aber am sinnvollsten. Zum Beispiel:

mvs = c(OM = -97, NV = -95, NR = -94, ND = -55, MD = -54)

In einem named character vector werden nun die Labels der UDMV festgehalten. Diese dienen später als Beschriftungen und Überschriften in Plots und in Tabellen. Die Namen müssen den Namen des zuvor definierten named integer vector entsprechen. Beachte, dass hier zusätzlich das Element “ALL” mit dem Label für alle fehlenden Werte benötigt wird.

labels\_mvs = c(  
 ALL = "total missing items",  
 OM = "omitted items",  
 NV = "not valid items",  
 NR = "not reached items",  
 ND = "not determinable items",  
 MD = "items missing by design"  
)

## Vorbereitung der DIF-Variablen

In den DIF-Analysen wird überprüft, ob die Items in verschiedenen Gruppen (z.B. Testhefte A und B) ein vergleichbares Antwortverhalten hervorrufen. Als sog. DIF-Variablen werden mehrere soziodemographische und methodische Variablen verwendet. Welche das in unseren Skalierungen im Normalfall sind, ist in den Abschnitten 2.2.3 *Soziodempgraphische Daten* und 2.2.4 *Informationen zum Testheftdesign* erklärt. Die Rohdaten muss man sich aus den verschiedenen SPSS-Dateien zusammensuchen.

### Laden der Daten und Zusammenführung mit Kompetenzdaten

Wie auch bei den Kompetenzdaten werden die SPSS-Dateien mit read\_sav() in R geladen und die benötigten Variablen (ID\_t, DIF-Variablen) mit select() ausgewählt (s. 4.1.1 *Laden der Daten und erste Anpassungen*). Generell werden die Daten aus der aktuellen Welle benutzt, fehlende Werte können jedoch auch aus früheren Wellen aufgefüllt werden. Wenn mehrere Werte vorliegen (z.B. Schulabschluss oder sozioökonomischer Status der Eltern), wird der höhere Wert verwendet.

Der data.frame mit den neuen Variablen wird mit dem data.frame resp durch die merge() Funktion zusammengeführt.

resp <- merge(  
 resp,  
 dif\_vars[, c("ID\_t", "dif\_var1", "dif\_var2", etc.)],  
 by = "ID\_t",  
 all.x = TRUE  
)

### Berechnen des Alters

Das Alter liegt im NEPS meistens nicht direkt vor, sondern muss erst aus dem Geburts- und dem Erhebungsdatum berechnet werden. Dies macht die Funktion calculate\_age(). Als Input übergibt man der Funktion den data.frame resp und die Namen der Variablen mit dem Geburtsdatum (Jahr, Monat und Tag). Bei den Variablen für das Testdatum kann man sich entscheiden, ob man der Funktion ebenfalls die Namen der entsprechenden Variablen in resp übergibt (wenn die Personen an unterschiedlichen Daten getestet wurden) oder alternativ jeweils nur eine Zahl definiert (wenn die Personen im/am gleichen Jahr/Monat/Jahr getestet wurden). Die Angabe des Geburtsjahres und -monats sowie des Testjahres und -monats ist verpflichtend, der Geburts- und Testtag kann dahingegen weggelassen werden. Die Funktion verwendet dann als Default den 15. des Monats. Sie gibt einen Vektor mit dem Alter in Jahren aus.

resp$age <- NEPSroutines::calculate\_age(  
 resp,  
 birth\_year = ‘birthy’,  
 birth\_month = ‘birthm’,  
 birth\_day = ‘birthd’,  
 test\_year = 2022,  
 test\_month = 10,  
 test\_day = 17  
)

### Dichotomisieren der soziodemographischen Variablen

Einige der DIF-Variablen (z.B. Alter) liegen noch mit mehr als zwei Ausprägungen vor und müssen daher dichotomisiert werden. Die Teilungskriterien für die Dichotomisierung sind:

* **Schultyp**: Nicht-Gymnasium (0) / Gymnasium (1)
* **Schulabschluss**: kein Abitur (0) / Abitur (1)
* **Migration**: > 2,25 Generation (0) / 1 bis 2 Generationen (1)
* **Anzahl Bücher Zuhause**: < 100 Bücher (0) / ≥ 100 Bücher (1)
* **Alter**: z.B. Mediansplit (nur bei altersheterogenen Gruppen notwendig)

Eine Dichotomisierung kann gut mit der ifelse() Funktion durchgeführt werden. Zum Beispiel:

resp$migration <- ifelse(  
 resp$migration %in% c(0, 4:10), 0,  
 ifelse(is.na(resp$migration), NA, 1)  
)

### Erstellen der Variablen zum Testheftdesign

Die Variable zum Testheftdesign aus den Rohdaten kann so noch nicht verwendet werden. Zunächst müssen aus ihr die nötigen Informationen rausgesucht werden (s. Abschnitt 2.2.4 *Informationen zum Testheftdesign*): Gab es verschiedene Rotationen der Testhefte? Gab es verschiedene Versionen der Testhefte? Gab es sonst noch für die Skalierung relevante Informationen? Für jede für die DIF-Analysen relevante Information wird im Datensatz resp eine dichotome Variable erstellt. Wenn die IRT-Modelle zudem für verschiedene Gruppen getrennt berechnet werden sollen (v.a. bei verschiedenen Testheften), wird zudem pro Gruppe eine logische Variable erstellt, die angibt, ob die Person zu der entsprechenden Gruppe gehört. Zum Beispiel sind dies in der B129 Studie ein einfaches und ein schweres Testheft.

# Create design variable for testlet version   
resp$testlet <- ifelse(resp[[design]] %in% c(241, 242, 243, 244, ...), "A", "B")  
# Create logical variables for testlet version  
resp$easy <- ifelse(resp$testlet == "A", TRUE, FALSE)  
resp$difficult <- ifelse(resp$testlet == "B", TRUE, FALSE)

### Labels für die Variablen

Um mit den Werten der dichotomen DIF-Variablen nicht durcheinander zu kommen, sollten den Variablen Labels vergeben werden. Das geht sehr unkompliziert über die Funktion labelled(). Zum Beispiel sieht der Code für die Variable Migrationshintergrund folgendermaßen aus:

resp$migration <- haven::labelled(  
 resp$migration,  
 c("no migration background" = 0, "migration background" = 1)  
)

## Sonstige Vorbereitungen

### Valide Fälle

Wenn Personen nur sehr wenige gültige Antworten gegeben haben, werden sie als invalide Fälle bezeichnet und aus den Analysen ausgeschlossen. Im NEPS ist die Mindestanzahl gültiger Antworten standardmäßig auf drei festgelegt. Um die (in)validen Fälle zu identifizieren, wird mit Hilfe der Funktion min\_val() eine neue logische Variable in resp erstellt. Als Input wird der Funktion resp, vars und select übergeben. Unter select sollte man diejenige Variable angeben, die alle für die Skalierung zu verwendenden Items identifiziert (also dich bei einem rein dichotomen Itemset und mixed bei einem gemischten Itemset). Zudem kann man mit dem Argument min.val die Mindestanzahl gültiger Antworten (Default ist 3) und mit dem Argument invalid einen Vektor mit allen als ungültig zu behandelnden Werten (Default sind alle negativen Werte und NAs) definieren. Ausgegeben wird ein logischer Vektor (TRUE für alle validen Fälle). Beachte, dass es bei Änderungen am Itemset (da z.B. einige Items ausgeschlossen wurden) notwendig sein kann, eine neue valid Variable anzulegen (z.B. valid\_new).

resp$valid <- NEPSroutines::min\_val(resp, vars, select = 'mixed')

### Gruppenvariablen

Liegt eine Gruppierung (z.B. Testhefte) vor, müssen weitere Variablen erstellt werden. Für diesen Abschnitt wird vorausgesetzt, dass sowohl in vars als auch in resp bereits für jede Gruppe eine logische Variable existiert, die die Items bzw. Personen der jeweiligen Gruppe identifiziert (z.B. jeweils easy & difficult).

Wurden den Gruppen nicht nur unterschiedliche Items administriert, werden u.a. für alle gemeinsamen Items IRT-Analysen durchgeführt. Dazu muss in vars eine logische Variable erstellt werden, die angibt, welche Items allen Gruppen administriert wurden. Analog dazu muss in resp eine logische Variable erstellt werden, die angibt, welchen Personen die gemeinsamen Items administriert wurden. Da die gemeinsamen Items *allen* Gruppen administriert wurden, wurden sie natürlich auch *allen* Personen administriert (die Variable ist also TRUE für alle Personen). In unserem Beispiel heißen die Variablen jeweils common.

vars$common <- vars$easy & vars$difficult  
resp$common <- TRUE

Zudem muss im Falle einer Gruppierung für jede Gruppe im Datensatz vars gruppenspezifische Analysevariablen (z.B. dich, mixed) definiert werden. Liegt z.B. ein leichtes und ein schwieriges Testheft mit teilweise gemeinsamen Items vor, sollte für jede benötigte Analysevariable eine neue Version erstellt werden, die dann nur die für die Gruppe relevanten Items identifiziert. Die mixed Variablen müssen natürlich nur erstellt werden, wenn im Test auch polytome Items enthalten sind.

# New dich variables  
vars$dich\_easy <- vars$dich & vars$easy  
vars$dich\_difficult <- vars$dich & vars$difficult  
vars$dich\_common <- vars$dich & vars$common  
  
# New mixed variables  
vars$mixed\_easy <- vars$mixed & vars$easy  
vars$mixed\_difficult <- vars$mixed & vars$difficult  
vars$mixed\_common <- vars$mixed & vars$common

### Variablen für die Scores und den SUF

Für die Berechnung der Summenscores und Metakognitionssores sowie für die Erstellung des SUF wird die logische Variable suf im Datensatz vars benötigt. Diese Variable gibt an, welche Items im Test verwendet wurden und in SUF übernommen werden sollen. Das sind gescorte dichotome und polytome Items *ohne* kollabierten Antwortkategorien.

vars$suf <- vars$mixed  
vars$suf[vars$item %in% original\_items] <- TRUE  
vars$suf[vars$item %in% collapsed\_items] <- FALSE  
vars$item[vars$suf]

Wenn die Scores für verschiedene Gruppen (z.B. Testhefte) getrennt berechnet werden sollen, muss pro Gruppe analog zu suf zusätzlich eine logische Variable erstellt werden, die alle für die Scores und den SUF benötigten Item *dieser Gruppe* identifiziert. Bei einem leichten und einem schweren Testheft können die Variablen z.B. suf\_easy und suf\_difficult genannt werden.

vars$suf\_easy <- ifelse(vars$suf & vars$easy, TRUE, FALSE)  
vars$suf\_difficult <- ifelse(vars$suf & vars$difficult, TRUE, FALSE)

Für die Berechnung der Summen- und Metakognitionsscores wird eine Variable (z.B. num\_cat) in vars benötigt, die Anzahl der maximal korrekten Antworten pro Item enthält. Bei polytomen Items entspricht das der Anzahl der Subitems und bei dichotomen ist sie 1. Die Funktion calculate\_num\_cat() berechnet die Anzahl der maximal korrekten Antworten pro Item. Sie benötigt dafür folgende Argumente: einerseits die weiter vorne erstellte Liste poly\_items und andererseits den Namen der logischen Variable in vars, die alle Items identifiziert, die zum Erstellen der Scores verwendet werden und in den SUF kommen (z.B. suf). Wenn der Test nur dichotome Variablen enthält, entfällt das Argument poly\_items.

vars$num\_cat <- NEPSroutines::calculate\_num\_cat(  
 vars,  
 poly\_items = poly\_items,  
 select\_suf = ‘suf’  
)

### Weitere benötigte Variablen

Für die Distraktorenanalyse braucht man eine logische Variable, die alle ungescorten Multiple-Choice Items identifiziert. In unserem Beispiel heißt sie distan.

vars$distan <- vars$raw & vars$type == 'MC'

Je nach Studiendesign werden noch weitere Variablen benötigt, z.B. für das Linking. Dafür liegen bisher noch keine Funktionen vor, sie müssen also eigenständig erstellt werden.

### Check der Datenstruktur

Wenn die Datenvorbereitung abgeschlossen ist, kann man die Datenstruktur mit Hilfe einiger Funktionen überprüfen. Das passiert zwar auch automatisiert im Rahmen der Funktionen der einzelnen Skalierungsabschnitte, eine vorgezogene Überprüfung kann jedoch einiges an Nachbearbeitung ersparen. Folgende Funktionen stehen zur Überprüfung zur Verfügung:

| **Funktion** | **Verwendung** |
| --- | --- |
| check\_logicals | Überprüft die genannten Variablen im definierten Datensatz auf Existenz, logischen Datentyp und keine fehlenden Werte. Der Funktion werden die Argumente df (der data.frame), name\_df (Name des data.frames als string) und logicals (character vector mit Namen der zu prüfenden Variablen) übergeben. |
| check\_numerics | Überprüft die genannten Variablen im definierten Datensatz auf Existenz, numerischer Datentyp und wenn gewollt, auch auf keine negativen Werte und / oder keine Werte größer als 1 (für dichotome Variablen). Der Funktion werden die Argumente df, name\_df, numerics (character vector mit Namen der zu prüfenden Variablen), check\_invalid (logical, ob auf negative Werte geprüft werden soll, Default ist FALSE) und dich (logical, ob auf Werte über 1 geprüft werden soll, Default ist FALSE) übergeben. |
| check\_variables | Grundfunktion für Variablen mit beliebigem Datentyp. Überprüft die genannten Variablen im definierten Datensatz nur auf Existenz. Der Funktion werden die Argumente df, name\_df und variables (character vector mit Namen der zu prüfenden Variablen) übergeben. |
| check\_pid | Überprüft, ob in der ID-Variable (**ID\_t**) Duplikate vorhanden sind. |
| check\_items | Überprüft, ob im Datensatz vars Itemnamen doppelt vorkommen. |

Beachte, dass bei korrekter Datenstruktur keine der Funktionen eine Meldung ausgibt. Wenn nach Ausführung der Funktionen also keine Fehlermeldungen erscheinen, passt soweit alles.

Eine Überprüfung der Datenstruktur könnte zum Beispiel folgendermaßen aussehen:

# Check vars  
NEPSroutines::check\_items(vars$item)  
NEPSroutines::check\_variables(vars, "vars", c("item", "type", "correct\_response"))  
NEPSroutines::check\_logicals(vars, "vars", c("raw", "dich", "poly", "mixed",  
"easy", "difficult"))  
NEPSroutines::check\_numerics(vars, "vars", c("scoring", "position\_easy", "position\_difficult"))  
  
# Check resp  
NEPSroutines::check\_variables(resp, "resp", c("ID\_t", vars$item[vars$raw]))  
NEPSroutines::check\_logicals(resp, "resp", c("valid", "easy", "difficult"))  
NEPSroutines::check\_numerics(resp, "resp", c("sex", "migration", "books"),  
dich = TRUE, check\_invalid = TRUE)  
NEPSroutines::check\_numerics(resp, "resp", vars$item[vars$dich], dich = TRUE)  
NEPSroutines::check\_numerics(resp, "resp", vars$item[vars$poly])  
NEPSroutines::check\_pid(resp$ID\_t)

### R Markdown und Variablen des Typs „haven\_labelled“

Leider kommt R Markdown aktuell (Stand: Oktober 2024) nicht immer mit Variablen des Typs haven\_labelled zurecht. Das kann beim Rendern zu Fehlern führen, die beim Ausführen des Codes ohne R Markdown nicht auftreten (z.B. Fehlermeldung „ungültiger Argumenttyp“). In diesem Fall muss man die Variablen konvertieren, aber ohne dass die UDMV verloren gehen. In den meisten Fällen klappt es mit dem folgenden Code, prüfe aber auf jeden Fall nach, dass die fehlenden Werte nicht in NAs umgewandelt wurden!

# Konvertiere “haven\_labelled” Variablen in “double”

resp <- haven::zap\_labels(resp)

# Prüfe nach, ob die user defined missing values noch bestehen  
table(unlist(resp[ , vars$item]), useNA = “ifany”)

### Speichern der Daten

Wenn die Vorbereitungen abgeschlossen sind, können die Daten gespeichert werden.

save(resp, vars, mvs, labels\_mvs, file = here::here("Data/dat.Rdata"))

Ab sofort kann man sich den Schritt der Datenvorbereitung sparen und die fertigen Daten direkt laden.

load(here::here("Data/dat.Rdata"))

## Veränderungen an den Items während der Skalierung

Es kommt häufig vor, dass sich der Itemsatz für die Analysen im Laufe der Skalierung ändert, da man aufgrund von schlechten Kennwerten einzelne Items ausschließt oder mehrere Items zu einem neuen Item zusammenfasst. In diesem Fall muss man bestimmte Teile der Datenvorbereitung erneut durchführen, z.B. neue Variablen zur Identifizierung des finalen Itemsatzes und für die finale Itempositionen erstellen oder vorhandene Variablen wie die Scoring- oder Validitätsvariable entsprechend ändern.

# Deskriptive Analysen

## Deskriptive Analyse der Stichprobe

Zunächst sollten deskriptive Analysen der Stichprobe durchgeführt werden. Dafür stehen im Rahmen des NEPSroutines Pakets mehrere kleinere Funktionen zur Verfügung. Sie fassen die zentralen Informationen durch Text und Tabellen zusammen und stellen die Ergebnisse in der Konsole dar.

Die folgende Tabelle gibt einen Überblick über die zur Verfügung stehenden Funktionen:

| **Funktion** | **Output** |
| --- | --- |
| n\_valid | Anzahl aller gültigen und ungültigen Fälle im Datensatz. |
| desc\_con | Tabelle mit deskriptiven Kennwerten (z.B. *M*, *SD*) für die in desc definierten kontinuierlichen Variablen. |
| desc\_abs | Tabelle mit Anzahl der Fälle jeder Kategorie für die in desc definierten kategorialen Variablen (z.B. Anzahl *male* und *female* im Datensatz bei Variable *sex*). |
| desc\_perc | Tabelle mit Prozentangaben der Fälle jeder Kategorie für die in desc definierten kategorialen Variablen (z.B. Prozent *male* und *female* im Datensatz bei Variable *sex*). Beim Funktionsargument useNA kann man wählen, ob die Prozentangaben die fehlenden Fälle berücksichtigen sollen (TRUE) oder nicht (FALSE). |
| desc\_nom | Vereint die beiden Funktionen desc\_abs und desc\_perc. Gibt also sowohl die absoluten als auch die relativen Zahlen für kategoriale Variablen als Tabelle aus. |
| show\_attributes | Gibt die Labels der Variablen an, um die Werte einer bestimmten Kategorie zuordnen zu können (z.B. 1 = *male* und 2 = *female*); funktioniert nur, wenn die Variablen als hvn\_lbl definiert sind. |

Die meisten dieser Funktionen verwenden zwei neue Funktionsargumente. Das Funktionsargument resp verlangt einen Datensatz, der für alle Teilnehmenden die soziodemographischen Informationen enthält. Wenn du die Daten wie in dieser Anleitung vorgeschlagen vorbereitet hast, ist das der Datensatz resp. Das Funktionsargument desc verlangt jeweils den oder die Namen der zu analysierenden Variablen, also z.B. alle kategorialen soziodemographischen Variablen bei der Funktion desc\_nom.

Die Syntax dafür könnte folgendermaßen aussehen:

# Sample size  
message("There are in total ", nrow(resp), " cases in the dataset.")  
  
# Number of (in)valid cases  
NEPSroutines::n\_valid(resp = resp, valid = 'valid')  
  
## Descriptives (mean, sd, etc.) of continuous variables  
NEPSroutines::desc\_con(resp = resp, desc = 'age', valid = 'valid')  
  
# Creating vectors with all categorical sociodemographic or booklet variables, respectively  
sociodemographics = c('sex', 'attendsGymnasium', 'migration', 'books')  
booklet = 'difficulty' # In other NEPSroutiness some more characteristics might apply here, e.g. position of booklet  
  
# Descriptives (attributes, frequency) of nominal sociodemographic variables  
NEPSroutines::desc\_nom(resp = resp, desc = sociodemographics, valid = 'valid')  
NEPSroutines::show\_attributes(resp = resp, desc = sociodemographics)  
  
# Descriptives (attributes, frequency) of nominal booklet variables  
NEPSroutines::desc\_nom(resp = resp, desc = booklet, valid = 'valid')  
NEPSroutines::show\_attributes(resp = resp, desc = booklet)  
  
# Delete variables  
rm(sociodemographics, booklet)

## Deskriptive Analyse nach Subgruppen (falls zutreffend)

Bei vielen Skalierungen liegt eine oder mehrere Gruppierungen vor (z.B. verschiedene Testhefte). Es ist sinnvoll, sich für diese Gruppen getrennt die Stichprobengröße sowie Anzahl und Art der enthaltenen Items ausgeben zu lassen. Vor allem für alle Dimensionalitätsvariablen (also diejenigen, die später für die Dimensionalitätsanalyse verwendet werden) sollte man sich die deskriptiven Statistiken getrennt nach Gruppe ansehen. Diese Informationen sind auch für den Technical Report wichtig. Das NEPSroutines Paket bietet zwei Funktionen zur deskriptiven Analyse der Testhefte: sample\_by\_group() und props\_by\_group().

Die Funktion sample\_by\_group() gibt eine Tabelle mit der Stichprobengröße pro Gruppe aus. Das Funktionsargument grouping\_variable enthält den Namen der Gruppierungsvariablen in resp. Das Funktionsargument labels ist optional und kann verwendet werden, um den Gruppen Namen zuzuordnen, falls die entsprechende Variable in resp keine eigenen Labels besitzt. Dazu übergibt man dem Argument einen named vector, der als Namen der Elemente die Namen der Gruppen enthält (s. Code als Beispiel).

Die Funktion props\_by\_group() gibt eine Tabelle mit den Eigenschaften der Items pro Gruppe aus, also z.B. wie viele Items welchen Typs in welcher Gruppe enthalten sind. Als Funktionsargument properties muss ein Character Vector übergegeben werden, der die Namen aller Variablen in vars enthält, die Itemeigenschaften beschreiben (z.B. text\_type, kog\_type und type). Zusätzlich können mit dem Funktionsargument labels benutzerdefinierte Labels für die Kategorien festgelegt werden, wenn die entsprechenden Variablen in vars nicht schon Labels besitzen.

Die Syntax könnte folgendermaßen aussehen:

# Sample size by test version  
NEPSroutines::sample\_by\_group(  
 resp,  
 grouping\_variable = 'difficulty',  
 labels = c('EasyTestlet' = 0, 'DifficultTestlet' = 1),  
 save = TRUE,  
 overwrite = TRUE,  
 path = "Tables"  
)  
  
# Testlet properties by test version  
lbls\_txt = c(Advertising = 1, Information = 2, Instruction = 3, Literary = 4, Commenting = 5)  
lbls\_kog = c(Finding\_information = 1, Drawing\_conclusions = 2, Reflecting = 3)  
lbls\_type = c(MC = 2, CMC = 1, TXT = 3)  
  
NEPSroutines::props\_by\_group(  
 vars,  
 select = 'mixed',  
 grouping = c('easy', 'difficult'),  
 properties = c('text\_type', 'kog\_type', 'type'),  
 labels = list(text\_type = lbls\_txt, kog\_type = lbls\_kog, type = lbls\_type),  
 save = TRUE, overwrite = TRUE,  
 path = "Tables"  
)  
  
# Delete variables  
rm(lbls\_txt, lbls\_kog, lbls\_type)

# Analyse fehlender Werte

## Allgemeine Informationen

Für die Skalierung eines Tests ist es wichtig, die verschiedenen Arten von fehlenden Werten zu analysieren, also zu untersuchen, aus welchen Gründen fehlende Werte wie oft vorliegen. Das machen wir für NEPS Tests einerseits pro Person (wie viele Items wurden durchschnittlich pro Person nicht beantwortet und warum) und andererseits pro Item (wie viele Personen haben ein Item beantwortet bzw. nicht beantwortet und warum). Um diese Analysen durchführen zu können, müssen die fehlenden Werte der Kompetenzitems als user-defined missing values kodiert sein. Welche das im NEPS normalerweise sind und wie man die Funktionsargumente mvs und labels\_mvs erstellt, kann Abschnitt 4.1.4 entnommen werden.

Die Analysen sollten nur für alle validen Fälle (Funktionsargument valid) durchgeführt werden. Es sollten nur diejenigen Items (Funktionsargument select) berücksichtigt werden, die auch für die finale IRT-Skalierung verwendet werden (in den meisten Fällen dich oder mixed, je nach Zusammensetzung des Itemsets). Die Ergebnisse der Analysen werden automatisch im RDS-Format gespeichert. Die Zusammenfassung der Ergebnisse wird als Excel-Tabelle gespeichert und in der Konsole ausgegeben. Um Plots zu erstellen und speichern, muss das Funktionsargument plots auf TRUE gesetzt werden.

## Analyse fehlender Werte pro Person

### Übergreifende Funktion

Mit Hilfe der Funktion mv\_person() werden die fehlenden Werte pro Person analysiert. Diese Analyse enthält Informationen über alle im Funktionsargument mvs definierten user-defined missing values. Außerdem werden die Kennwerte auch für alle in mvs definierten fehlenden Werte zusammen berechnet. Nicht in mvs definierte fehlende Werte gehen in diese Summe aber **nicht** mit ein. Es ist also wichtig, tatsächlich alle existierenden user-defined missing values in mvs einzubeziehen, wenn man den Summenwert (“ALL”) sinnvoll interpretieren möchte. Eine Ausnahme sind die design-bedingt fehlenden Antworten, die automatisch vor der Analyse aus dem Datensatz gefiltert werden. Dazu übergibt man dem Funktionsargument missing\_by\_design den entsprechenden user-defined missing value (Default ist -54). Es ist sehr wichtig, hier den korrekten Wert anzugeben! Sonst werden die falschen Antworten aus der Analyse gefiltert und auch die Anzahl der Personen, denen jedes Item administriert wurde, ist dann fehlerhaft.

Die Analysen können auch getrennt nach verschiedenen Gruppen (z.B. Testhefte durchgeführt werden. Diese müssen im Funktionsargument grouping definiert werden. Die fehlenden Werte werden dann pro Gruppe nur für die Items berechnet, die der Gruppe auch administriert wurden. Den Name der Gruppierung (z.B. “test version”) kann man mithilfe des Arguments name\_grouping für die Benennung der Grafiken festlegen. Wenn man sich nicht nur die Ergebnisse pro Gruppe, sondern auch die Ergebnisse für die Gesamtstichprobe ausgeben lassen möchte (sowohl in Tabellen als auch in Grafiken), muss das Funktionsargument show\_all auf TRUE gesetzt werden.

NEPSroutines::mv\_person(  
 resp = resp,  
 vars = vars,  
 select = 'mixed',  
 valid = 'valid',  
 grouping = c('easy', 'difficult'),  
 mvs = mvs,  
 labels\_mvs = labels\_mvs,  
 missing\_by\_design = -54,  
 plots = TRUE,  
 print = TRUE,  
 save = TRUE,  
 return = FALSE,  
 path\_results = ("Results"),  
 path\_table = ("Tables"),  
 path\_plots = ("Plots/Missing\_Responses/by\_person"),  
 show\_all = FALSE,  
 overwrite = TRUE,  
 name\_grouping = 'test version',  
 digits = 3  
)

Die Funktion beinhaltet folgende (optionale) besondere Funktionsargumente:

| **Argument** | **Art** | **Beschreibung** | **Default** |
| --- | --- | --- | --- |
| labels\_mvs | named character vector | Enthält die Labels für die UMVs als Elemente und die Abkürzungen der UMVs als Namen der Elemente (diese müssen identisch zu denen aus mvs sein). Die Labels werden für die spätere Bezeichnung in den Ergebnissen verwendet. | (zu lang um es hier darzustellen, s. Hilfefunktion oder Abschnitt 4.1.4 *User defined missing values*) |
| show\_all | logical | Ob bei Gruppierung die Gesamtstichprobe als zusätzliche Gruppe berücksichtigt werden soll. | TRUE |
| name\_grouping | string | Bezeichnung für die Gruppierung (nicht der einzelnen Gruppen!) für die Ergebnisdarstellung und -benennung. | “test version” |
| labels\_legend | character vector | Gruppenlabels für die Legende (achte aber auf die richtige Reihenfolge!). | Gruppierungs-variablennamen |
| color | character vector | Enthält die Farben für die Plots. Länge des Vektors muss mit Anzahl der Gruppen (z.B. drei bei zwei Testheften & show\_all = TRUE) übereinstimmen. Farben können durch Wörter (z.B. blue) oder Farbcodes (z.B. #FF0000) definiert werden. | NULL (wird mit blau gestufter Farbpalette ersetzt) |

### Untergeordnete Funktionen

Neben der übergreifenden Funktion können noch die folgenden untergeordneten Funktionen verwendet werden:

| **Funktion** | **Input** | **Output** |
| --- | --- | --- |
| mvp\_analysis | Dataframes resp und vars mit Informationen über Antworten, Items, Personenmerkmale und ggf. Gruppierung | Ergebnisse der MVP-Analysen (R-Objekt, ggf. RDS) |
| mvp\_plots | Output der Funktion mvp\_analysis bzw. Element mv\_p des Outputs der Funktion mv\_person | Plots für jede user defined missing value Art (PNG) |
| mvp\_table | Output der Funktion mvp\_analysis bzw. Element mv\_p des Outputs der Funktion mv\_person | Tabelle mit Ergebnissen der MVP-Analysen für den Technical Report (R-Objekt, ggf. Excel) |
| print\_mvp\_results | Output der Funktion mvp\_analysis bzw. Element mv\_p des Outputs der Funktion mv\_person | Zusammenfassung der Ergebnisse der MVP-Analysen (in Konsole) |

Wie die einzelnen Funktionsargumente genutzt werden, kann man über die Hilfe-Funktion (z.B. ?mvp\_analysis) herausfinden.

## Analyse fehlender Werte pro Item

### Übergreifende Funktion

Ähnlich wie bei der Funktion mv\_person() können auch mit der Funktion mv\_item() fehlende Werte untersucht werden. Der Unterschied zu der oben beschriebenen Funktion besteht darin, dass die Anteile bestimmter fehlender Werte nun für jedes Item und nicht pro Person dargestellt werden. Es werden wieder Informationen über alle im Funktionsargument mvs definierten user-defined missing values dargestellt. Wie auch zuvor werden die Kennwerte außerdem für alle in mvs definierten fehlenden Werte zusammen berechnet. Nicht in mvs definierte fehlende Werte gehen in diese Summe aber **nicht** mit ein. Es ist also wichtig, tatsächlich alle existierenden user-defined missing values in mvs einzubeziehen, wenn man den Summenwert (“ALL”) sinnvoll interpretieren möchte. Eine Ausnahme sind die design-bedingt fehlenden Antworten, die automatisch vor der Analyse aus dem Datensatz gefiltert werden. Dazu übergibt man dem Funktionsargument missing\_by\_design den entsprechenden user-defined missing value (Default ist -54). Es ist sehr wichtig, hier den korrekten Wert anzugeben! Sonst werden die falschen Antworten aus der Analyse gefiltert und auch die Anzahl der Personen, denen jedes Item administriert wurde, ist dann fehlerhaft.

Die Analysen können auch getrennt nach verschiedenen Gruppen (z.B. Testhefte) durchgeführt werden. Diese müssen im Funktionsargument grouping definiert werden. Die fehlenden Werte werden dann pro Gruppe nur für die Items berechnet, die der Gruppe auch administriert wurden. Den Namen der Gruppierung (z.B. “test version”) kann man mithilfe des Arguments name\_grouping für die Benennung der Grafiken festlegen. Die fehlenden Werte pro Item werden in Abhängigkeit der Itemposition und ggf. Gruppierung dargestellt.

Durch Aktivieren des Arguments save (mit save=TRUE) kann man die Ergebnisse der Analysen in .rds und .xlsx Dateien speichern. Die Ergebnisse der Analyse aus der ***Excel-Tabelle*** können z.B. in den Technical Report übernommen werden. Wenn das Argument suf\_item\_names auf TRUE gesetzt wird, werden die Namen von polytomen Items mit kollabierten Kategorien ohne das Suffix ´\_collapsed´ ausgegeben. Per Default (suf\_item\_names = FALSE) bleiben die Itemnamen unverändert. Beachte, dass in .rds Dateien die Itemnamen immer unverändert bleiben.

NEPSroutines::mv\_item(  
 resp = resp,  
 vars = vars,  
 select = 'mixed',  
 valid = 'valid',  
 grouping = c('easy', 'difficult'),  
 mvs = mvs,  
 labels\_mvs = labels\_mvs,  
 missing\_by\_design = -54,  
 position = c(easy = 'position\_easy\_mixed',  
 difficult = 'position\_difficult\_mixed'),  
 plots = TRUE,  
 print = TRUE,  
 save = TRUE,  
 return = FALSE,  
 path\_results = ("Results"),  
 path\_table = ("Tables"),  
 path\_plots = ("Plots/Missing\_Responses/by\_item"),  
 suf\_item\_names = TRUE,  
 show\_all = FALSE,  
 overwrite = TRUE,  
 name\_grouping = 'test version',  
 digits = 3  
)

Die Funktion beinhaltet folgende (optionale) besondere Funktionsargumente:

| **Argument** | **Art** | **Beschreibung** | **Default** |
| --- | --- | --- | --- |
| position | string | Name der numerischen Variable(n) in vars, die die Position der Items angibt/angeben. | NULL |
| labels\_mvs | named character vector | Enthält die Labels für die UMVs als Elemente und die Abkürzungen der UMVs als Namen der Elemente (diese müssen identisch zu denen aus mvs sein). Die Labels werden für die spätere Bezeichnung in den Ergebnissen verwendet. | (zu lang um es hier darzustellen, s. Hilfefunktion oder Abschnitt 4.1.4 *User defined missing values*) |
| show\_all | logical | Ob bei Gruppierung die Gesamtstichprobe als zusätzliche Gruppe berücksichtigt werden soll. | TRUE |
| name\_grouping | string | Bezeichnung für die Gruppierung (nicht der einzelnen Gruppen!) für die Ergebnisdarstellung und -benennung. | “test version” |
| labels\_legend | character vector | Gruppenlabels für die Legende (achte aber auf die richtige Reihenfolge!). | Gruppierungs-variablennamen |
| color | character vector | Enthält die Farben für die Plots. Länge des Vektors muss mit Anzahl der Gruppen (z.B. drei bei zwei Testheften & show\_all = TRUE) übereinstimmen. Farben können durch Wörter (z.B. blue) oder Farbcodes (z.B. #FF0000) definiert werden. | NULL (wird mit blau gestufter Farbpalette ersetzt) |

### Untergeordnete Funktionen

Neben der übergreifenden Funktion können noch die folgenden untergeordneten Funktionen verwendet werden:

| **Funktion** | **Input** | **Output** |
| --- | --- | --- |
| mvi\_analysis | Dataframes resp und vars mit Informationen über Antworten, Items, Personenmerkmale und ggf. Gruppierung | Ergebnisse der MVI-Analysen (R-Objekt, ggf. RDS) |
| mvi\_plots | Output der Funktion mvi\_analysis (bzw. Element mv\_i des Outputs der Funktion mv\_item) | Plots für jede user defined missing value Art (PNG) |
| mvi\_table | Output der Funktion mvi\_analysis (bzw. Element mv\_i des Outputs der Funktion mv\_item) | Tabelle mit Ergebnissen der MVI-Analysen für den Technical Report (R-Objekt, ggf. Excel) |
| print\_mvi\_results | Output der Funktion mvi\_analysis (bzw. Element mv\_i des Outputs der Funktion mv\_item) | Zusammenfassung der Ergebnisse der MVI-Analysen (in Konsole) |

Wie die einzelnen Funktionsargumente genutzt werden, kann man über die Hilfe-Funktion (z.B. ?mvi\_analysis) herausfinden.

## Analyse fehlender Werte bei Multistagetests

In Multistagetests kann man die Anzahl der nicht erreichten Items nicht so einfach interpretieren wie in anderen Tests. Wenn eine Person einen normalen Test abbricht, werden alle fehlenden Werte nach dem zuletzt bearbeiteten Item als „nicht erreicht“ definiert. Im Multistagetest wird das grundsätzlich auch so gehandhabt, aber nur bis zum Ende des jeweiligen Abschnitts (= Stage). Wenn eine Person also beispielsweise den Test schon im ersten Abschnitt abbricht, werden zwar alle darauffolgenden Items im gleichen Abschnitt als nicht erreicht, aber alle Items von späteren Abschnitten als nicht administriert definiert. Das führt dazu, dass die Anzahl nicht erreichter Items in späteren Abschnitten streng genommen nicht mehr korrekt ist (denn sie bezieht sich nur auf die Anzahl nicht erreichter Items von den Personen, die den Abschnitt auch begonnen haben).

Eine Lösung des Problems könnte daher sein, (a) die Anzahl nicht erreichter Items getrennt nach Abschnitt darzustellen und (b) zusätzlich die Anzahl der Personen, die die jeweiligen Abschnitte des Tests auch tatsächlich begonnen haben, zu berichten. Das kann mit dem NEPSroutines Paket auf zwei verschiedene Arten erreicht werden, die im Folgenden vorgestellt werden. Für beide Vorgehensweisen ist es aber zunächst wichtig, analog zum Vorgehen bei Gruppierungen (s. Abschnitt 3.1. *Benötigte Datenstruktur* und 4.2. *Vorbereitung der DIF-Variablen*) für jeden Testabschnitt jeweils eine logische Variable in resp und in vars zu erstellen (z.B. „stage1“), die diejenigen Personen und Items kennzeichnet, die den Abschnitt auch tatsächlich begonnen haben bzw. ihm angehören.

Einerseits kann man die verschiedenen Abschnitte als Gruppierung auffassen (z.B. grouping = c(‘stage1’, ‘stage2’)) und somit die Analysen analog zum normalen Vorgehen durchführen. In diesem Fall können die verschiedenen Itemblöcke innerhalb der Abschnitte nicht mehr in den Funktionen berücksichtigt werden. Beachte bitte außerdem, dass sich die Prozentwerte dann auf unterschiedlich große Stichproben beziehen (da vermutlich mehr Personen den ersten Abschnitt begonnen haben als den zweiten). Die verschiedenen Stichprobengrößen sollten deshalb auf jeden Fall auch berichtet werden.

NEPSroutines::mv\_person(  
 resp = resp,  
 vars = vars,  
 select = 'mixed',  
 valid = 'valid',  
 grouping = c('stage1', 'stage2'),  
 mvs = mvs,  
 labels\_mvs = labels\_mvs,  
 plots = TRUE  
)  
NEPSroutines::mv\_item(  
 resp = resp,  
 vars = vars,  
 select = 'mixed',  
 valid = 'valid',  
 grouping = c('stage1', 'stage2'),  
 position = c(stage1 = 'position\_stage1', stage2 = 'position\_stage2'),  
 mvs = mvs,  
 labels\_mvs = labels\_mvs,  
 suf\_item\_names = TRUE,  
 plots = TRUE  
)

Andererseits kann man die Analyse fehlender Werte für jeden Testabschnitt einzeln ausführen. Wichtig ist hierbei, dass man den Datensatz resp für diese Analyse **auf diejenigen Personen** **beschränkt**, die den jeweiligen Abschnitt tatsächlich auch begonnen haben. Die verschiedenen Itemblöcke innerhalb der Abschnitte können, wenn man das möchte, in diesem Fall als Gruppierung in den Funktionen berücksichtigt werden. Bei einem Test mit zwei Abschnitten könnte der Code folgendermaßen aussehen:

# Neue Datensätze mit Teilstichprobe erstellen  
resp\_stage1 <- resp[resp$stage1,]  
resp\_stage2 <- resp[resp$stage2,]

# Anzahl der Personen, die den jeweiligen Abschnitt begonnen haben  
nrow(resp\_stage1) # Abschnitt 1  
nrow(resp\_stage2) # Abschnitt 2  
  
# Analyse fehlender Werte pro Abschnitt  
  
## Abschnitt 1  
  
NEPSroutines::mv\_person(  
 resp = resp\_stage1,  
 vars = vars,  
 select = 'mixed\_stage1',  
 valid = 'valid',  
 grouping = c('block\_a', 'block\_b'),  
 mvs = mvs,  
 labels\_mvs = labels\_mvs,  
 plots = TRUE  
)  
  
NEPSroutines::mv\_item(  
 resp = resp\_stage1,  
 vars = vars,  
 select = 'mixed\_stage1',  
 valid = 'valid',  
 grouping = c('block\_a', 'block\_b'),  
 position = c(block\_a = 'position\_stage1\_blocka',  
 block\_b = 'position\_stage1\_blockb'),  
 mvs = mvs,  
 labels\_mvs = labels\_mvs,  
 suf\_item\_names = TRUE,  
 plots = TRUE  
)  
  
## Abschnitt 2  
  
… (analog zu Abschnitt 1)

Beachte bitte, dass bei beiden Lösungen die Summe aller fehlender Werte („ALL“) nie diejenigen fehlenden Wertebeinhaltet, die durch **den Abbruch des Tests vor dem letzten Abschnitt** zustande kommen.

# IRT-Analyse

## Allgemeine Informationen

Im Rahmen der IRT-Analysen werden die Fähigkeitsparameter der Personen sowie die Schwierigkeits- und ggf. Diskriminierungsparameter der Items geschätzt. Je nach vorliegendem Datenformat werden eine 1PL- und 2PL-Analyse (nur dichotome Items) und ggf. zusätzlich eine PCM- und GPCM-Analyse (auch polytome Items) durchgeführt und die Ergebnisse per Modellfit miteinander verglichen.

Die Analysen sollten wieder nur für alle validen Fälle (Funktionsargument valid) durchgeführt werden. Die 1PL- und 2PL-Analysen müssen immer durchgeführt werden, für sie werden alle dichotom gescorten Items verwendet (select='dich'). Wenn der Test auch polytome Items beinhaltet, werden zusätzlich noch PCM-und GPCM-Analysen mit allen in der Variable mixed definierten Items durchgeführt. Sollte der Itemsatz im Laufe der Skalierungen verändert werden, da z.B. Items ausgeschlossen werden, müssen die IRT-Analysen mit dem veränderten Itemsatz erneut durchgeführt werden (unter select werden dann entsprechend die neuen Analysevariablen definiert, z.B. dich\_new & mixed\_new).

Liegt eine Gruppierung (z.B. Testhefte) vor, werden die Analysen in den meisten Fällen sowohl für die ganze Stichprobe, als auch für jede Gruppe einzeln durchgeführt. Wenn die Gruppen sich zudem gemeinsame Items teilen, werden zusätzlich noch IRT-Analysen nur mit diesen Items durchgeführt. So kann man insgesamt auf sehr viele IRT-Analysen kommen (dichotome und polytome IRT-Analysen für jeweils die ganze Stichprobe und jede Gruppe). In diesem Fall hilft die Funktion grouped\_irt\_analysis() weiter, mit der man IRT-Analysen für mehrere Gruppen mit nur einem Befehl durchführen kann. Beachte, dass die Datensätze resp und vars für die Analysen die entsprechenden Gruppenvariablen und gruppenspezifische Analysevariablen beinhalten müssen (s. Abschnitt 4.3.2 *Gruppenvariablen*).

Falls nicht jeder Person alle Items administriert wurden, sollte man bei beiden übergreifenden Funktionen jeweils das Argument missing\_by\_design definieren, indem man ihm den user-defines missing value für design-bedingt fehlende Antworten übergibt. Das wird benötigt, um die Anzahl der Personen, denen ein Item administriert wurde, korrekt zu berechnen.

## Übergreifende Funktion irt\_analysis

Die übergreifende Funktion irt\_analysis() führt die notwendigen IRT-Analysen automatisiert durch, gibt die Ergebnisse in einigen Tabellen und als Text in der Konsole aus und speichert die (wichtigsten) Ergebnisse in Excel-Tabellen und als RDS-Format. Auch die ICC-Plots und Wright Maps können erstellt und gespeichert werden (wenn plots=TRUE). Die Funktion erkennt automatisch, ob im Itemsatz polytome Items vorliegen oder nicht und führt entsprechend eine 1PL- und 2PL-Analyse (nur dichotome Items) oder eine PCM- und GPCM-Analyse (auch polytome Items) durch. Das PCM und GPCM wird mit der ConQuest Parametrisierung “item + item\*step” durchgeführt.

Die Funktion führt die Analysen nur für einen Itemsatz (z.B. alle Items, ausgewählt durch das Funktionsargument select). Wenn man für mehrere verschiedene Itemsätze (z.B. leichtes und schweres Testheft und gemeinsame Items) IRT-Analysen durchführen muss, aber sich die wiederholte Ausführung der Funktion irt\_analysis() sparen möchte, kann die Funktion grouped\_irt\_analysis() benutzt werden (s. folgender Abschnitt).

# Rasch and 2PL  
NEPSroutines::irt\_analysis(  
 resp = resp,  
 vars = vars,  
 select = 'dich',  
 valid = 'valid',  
 missing\_by\_design = -54,  
 scoring = NULL,  
 plots = TRUE,  
 save = TRUE,  
 print = TRUE,  
 return = FALSE,  
 path\_results = ("Results"),  
 path\_table = ("Tables"),  
 path\_plots = ("Plots"),  
 overwrite = TRUE,  
 digits = 3,   
 name\_group = "all"  
)  
  
# PCM and GPCM  
NEPSroutines::irt\_analysis(  
 resp = resp,  
 vars = vars,  
 select = 'mixed',  
 valid = 'valid',  
 scoring = 'scoring',  
 missing\_by\_design = -54,  
 plots = TRUE,  
 save = TRUE,  
 print = TRUE,  
 return = FALSE,  
 path\_results = ("Results"),  
 path\_table = ("Tables"),  
 path\_plots = ("Plots"),  
 suf\_item\_names = TRUE,  
 overwrite = TRUE,  
 digits = 3,  
 name\_group = "all"  
)

Die Funktion beinhaltet folgende (optionale) besondere Funktionsargumente:

| **Argument** | **Art** | **Beschreibung** | **Default** |
| --- | --- | --- | --- |
| scoring | string | Name der numerischen Variable in vars, die die Scoring-Werte der Items beinhaltet. Vor allem bei PCM und GPCM-Analysen benötigt. | NULL |
| xsi\_fixed\_1p | named numeric vector | Elemente des Vektors sind die Itemschwierigkeiten für das 1PL bzw. PCM Modell (Output der Funktion TAM::tam.mml unter mod$xsi$xsi). Namen der Elemente sind die Itemnamen. | NULL |
| xsi\_fixed\_2p | named numeric vector | Elemente des Vektors sind die Itemschwierigkeiten für das 2PL bzw. GPCM Modell (Output der Funktion TAM::tam.mml.2pl unter mod$xsi$xsi). Namen der Elemente sind die Itemnamen. | NULL |
| control\_tam | list | Funktionsargument control für die Funktion TAM::tam.mml.mfr. | NULL |
| control\_wle | list | Liste mit den Funktionsargumenten Msteps und/oder convM für die Funktion TAM::tam.wle als Listenelemente. | NULL |
| pweights | numeric vector | Personengewichte, die als Funktionsargument pweights den TAM-Funktionen übergeben werden. | NULL |

## Übergreifende Funktion grouped\_irt\_analysis

Mit der Funktion grouped\_irt\_analysis() können die IRT-Analysen für mehrere Itemsätze (z.B. leichtes und schwieriges Testheft & gemeinsame Items) mit nur einem Befehl durchgeführt werden. Obwohl die Funktionsargumente prinzipiell die gleichen wie in irt\_analysis() sind, muss man einige der Argumente nun pro Gruppe spezifizieren (z.B. select). Dazu erstellt man jeweils eine Liste, die pro Gruppe ein Element enthält. Die Namen der Listenelemente entsprechen den Namen der Gruppenvariablen (genauso, wie sie in resp und vars definiert sind!, z.B. easy). Die Listenelemente selber enthalten dann jeweils das gruppenspezifische Funktionsargument (z.B. mixed\_easy). Im Fall des Funktionsarguments select **müssen** alle Gruppen enthalten sein. Bei den anderen gruppenspezifischen Funktionsargumenten (control\_tam, control\_wle, xsi\_fixed\_1p, xsi\_fixed\_2p) können auch nur für einzelne Gruppen Listenelemente spezifiziert werden oder auch das gesamte Argument leer gelassen werden (NULL). In der B129 gab es ein schwieriges und ein leichtes Testheft, die auch gemeinsame Items beinhalten. Für die IRT-Analysen ist es also wichtig, dass die Datensätze resp und vars jeweils die logischen Variablen easy, difficult und common, sowie vars noch zusätzlich die gruppenspezifischen Analysevariablen beinhaltet (s. Abschnitt 4.3.2 *Gruppenvariablen*). Die Funktion grouped\_irt\_analysis wurde folgendermaßen definiert:

# Rasch and 2PL  
NEPSroutines::grouped\_irt\_analysis(  
 resp = resp,  
 vars = vars,  
 valid = 'valid',  
 scoring = 'scoring',  
 missing\_by\_design = -54,  
 select = c(easy = 'dich\_easy',  
 difficult = 'dich\_diff',  
 common = 'dich\_common'),  
 overwrite = TRUE,  
 plots = TRUE,  
 return = FALSE,  
 path\_results = ("Results"),  
 path\_table = ("Tables"),  
 path\_plots = ("Plots")  
)  
  
# PCM and GPCM  
NEPSroutines::grouped\_irt\_analysis(  
 resp = resp,  
 vars = vars,  
 valid = 'valid',  
 scoring = 'scoring',  
 missing\_by\_design = -54,  
 select = c(easy = 'mixed\_easy',  
 difficult = 'mixed\_difficult',  
 common = 'mixed\_common'),  
 overwrite = TRUE,  
 plots = TRUE,  
 return = FALSE,  
 path\_results = ("Results"),  
 path\_table = ("Tables"),  
 path\_plots = ("Plots"),  
 suf\_item\_names = TRUE  
)

## Untergeordnete Funktionen

Neben der übergreifenden Funktion können noch die folgenden untergeordneten Funktionen verwendet werden:

| **Funktion** | **Input** | **Output** |
| --- | --- | --- |
| irt\_model | Dataframes resp und vars mit Informationen über Antworten, Items, Personenmerkmale und ggf. Gruppierung | Ergebnisse der IRT-Analyse (R-Objekt, ggf. RDS) |
| icc\_plots | Output der Funktion irt\_model | ICC-Plots für IRT Modelle (PNG) |
| wright\_map | Output der Funktion irt\_model | Wright-Maps für IRT Modelle (PNG) |
| irt\_summary | Output der Funktion irt\_model | Tabelle mit Ergebnissen der IRT-Analyse (R-Objekt, ggf. Excel) |
| irt\_model\_fit | Output der Funktion irt\_model | Tabelle mit Modellfit der 1PL und 2PL-Modelle bzw. PCM und GPCM (R-Objekt, ggf. Excel) |
| step\_analysis | Output der Funktion irt\_model (nur bei PCM Analysen) | Tabelle mit Ergebnissen der step-Analyse (R-Objekt, ggf. Excel) |
| print\_irt\_summary | Output der Funktionen irt\_summary, irt\_model und ggf. steps\_anlysis | Zusammenfassung der IRT- und step-Analyse und zusätzliche für den TR relevante Ergebnisse (in Konsole) |

Wie die einzelnen Funktionsargumente genutzt werden, kann man über die Hilfe-Funktion (z.B. ?irt\_summary) herausfinden.

# DIF-Analyse

## Allgemeine Information

Differential Item Functioning (DIF) beschreibt den Sachverhalt, dass ein Item *bei gleicher Fähigkeit* für verschiedene Subgruppen unterschiedlich schwer ist. Somit wird durch die DIF-Analyse sichergestellt, dass keine Items besondere Gruppen (z.B. Mädchen vs. Jungen oder Migranten vs. Nicht-Migranten) bevorzugt oder benachteiligt. Hier werden ein Haupteffekt-Modell und ein DIF-Modell gerechnet. Der Haupteffekt gibt die mittlere Differenz der Kompetenz zwischen zwei Subgruppen an und ist daher kein Indiz für DIF. Relevanter DIF wird dann angenommen, wenn er bei Items einen Wert von **>0.4** annimmt. Ist der Wert **>0.6**, ist der DIF problematisch, davor lediglich bedenklich. Um die Größe des DIF zusätzlich einschätzen zu können, wird ein Minimum Effects Hypothesis Test durchgeführt. Die DIF-Werte sollten in dessen Rahmen **nicht signifikant** sein. Für die Tests, die im Rahmen der NEPS eingesetzt werden, sollte kein DIF vorhanden sein (also alle Werte **<0.4** und **nicht signifikant**).

Die Analysen sollten wieder nur für alle validen Fälle (Funktionsargument valid) durchgeführt werden. Die DIF-Analysen basieren entweder auf dem 1PL-Modell oder dem PCM. Analog zu den IRT-Analysen sollte immer eine dichotome DIF-Analyse durchgeführt werden. Falls der Test auch polytome Items beinhaltet, wird zusätzlich eine polytome DIF-Analyse durchgeführt. Gruppierungen müssen bei den DIF-Analysen in den meisten Fällen nicht beachtet werden, da sie schon als DIF-Variable in die Analysen einfließen.

## Übergreifende Funktion

Um zu prüfen, ob DIF vorliegt, kann die Funktion dif\_analysis() verwendet werden. Die Funktion kann die DIF-Analysen für mehrere DIF-Variablen gleichzeitig durchführen (definiert im Funktionsargument dif\_vars). In diesem Fall kann im Funktionsargument select mit einem *Vektor* für jede DIF-Variable eine eigene select-Variable definiert werden (s. Beispiel). Wenn die für die Analysen benötigten Items aber zwischen den verschiedenen DIF-Variablen nicht variieren, reicht wie gewohnt ein einzelner *String* (z.B. select = "mixed"). Die benötigten Items können variieren, wenn für eine DIF-Variable nur bestimmte Items betrachtet werden können (z.B. nur beiden Testhefte gemeinsame Items bei der DIF-Variable difficulty).

Die Funktion untersucht einerseits, ob sich Itemschwierigkeiten signifikant zwischen den untersuchten DIF-Gruppen unterscheiden. Andererseits führt sie einen Modellvergleich zwischen dem DIF-Modell und dem Haupteffekt-Modell anhand der Informationskriterien AIC und BIC durch. Die Ergebnisse werden automatisch im RDS-Format und in Excel-Tabellen gespeichert. Die Funktion speichert insbesondere eine Tabelle, die direkt für den TR verwendet werden kann. In dieser Tabelle werden für jedes Item die unstandardisierten (vor der Klammer) und die standardisierten (in der Klammer) DIF-Effekte angegeben. Eine Zusammenfassung der Ergebnisse wird zudem in der Konsole ausgegeben.

# For Rasch  
NEPSroutines::dif\_analysis(  
 resp = resp,  
 vars = vars,  
 select = c('dich\_common', "dich", "dich"),  
 valid = 'valid',   
 dif\_vars = c('difficulty', 'sex', 'migration'),  
 overwrite = TRUE,  
 print = TRUE,  
 save = TRUE,  
 return = FALSE,  
 path\_table = here::here("Tables"),  
 path\_results = here::here("Results")  
)  
  
# For PCM  
NEPSroutines::dif\_analysis(  
 resp = resp,  
 vars = vars,  
 select = c('mixed\_common', "mixed", "mixed"),  
 valid = 'valid',   
 dif\_vars = c('difficulty', 'sex', 'migration'),  
 scoring = "scoring",   
 overwrite = TRUE,  
 print = TRUE,  
 save = TRUE,  
 return = FALSE,  
 path\_table = here::here("Tables"),  
 path\_results = here::here("Results"),  
 suf\_item\_names = TRUE  
)

Die Funktion beinhaltet folgende (teilweise optionale) besondere Funktionsargumente:

| **Argument** | **Art** | **Beschreibung** | **Default** |
| --- | --- | --- | --- |
| dif\_vars (nicht optional) | character vector | Vektor mit dem/den Namen der Variablen in resp, die im Rahmen der DIF-Analysen überprüft werden soll/en. |  |
| select | character vector | Vektor entweder mit einer select-Variablen für alle DIF-Variablen aus dif\_vars, oder mit genau einer select-Variablen für jede DIF-Variable aus dif\_vars. |  |
| scoring | string | Name der numerischen Variable in vars, die die Scoring-Werte der Items beinhaltet. | “scoring” |
| include\_mv | integer | Anzahl der mindestens benötigten fehlenden Werte, um fehlende Werte als zusätzliche Gruppe in die DIF-Analysen zu integrieren. | 200 |
| control | list | Funktionsargument control für die TAM-Funktionen. | NULL |
| pweights | numeric vector | Personengewichte für den aktuellen Messzeitpunkt, die als Funktionsargument pweights den TAM -Funktionen übergeben werden. | NULL |
| dif\_threshold | numeric | Grenzwert, ab dem DIF im Output als problematisch markiert wird. | 0.5 |

## Untergeordnete Funktionen

Neben der übergreifenden Funktion können noch die folgenden untergeordneten Funktionen verwendet werden. Beachte, dass einige Funktionen nur zur DIF-Analyse mit **einer** DIF-Variablen und andere Funktionen auch zur DIF-Analyse mit **mehreren** DIF-Variablen verwendet werden können. Das Funktionsargument für die DIF-Variable(n) heißt dann entsprechend dif\_var (bei einer DIF-Variablen) oder dif\_vars (bei mehreren DIF-Variablen). Das Funktionsargument select kann entsprechend nur als String, oder auch als Vektor definiert werden (also eine select-Variable pro DIF-Variable).

| **Funktion** | **Input** | **Output** |
| --- | --- | --- |
| dif\_model | Dataframes resp und vars mit Informationen über Antworten, Items und Personenmerkmale | Ergebnisse der DIF-Analyse für **eine** DIF-Variable (R-Objekt) |
| conduct\_dif\_analysis | Dataframes resp und vars mit Informationen über Antworten, Items und Personenmerkmale | Ergebnisse der DIF-Analyse für **mehrere** DIF-Variablen (R-Objekt, ggf. RDS) |
| dif\_summary | Output der Funktion dif\_model (bzw. das Element models$NameDIFVariable des Outputs der Funktion dif\_analysis) | Tabellen mit Ergebnissen der DIF-Analyse für **eine** DIF-Variable (R-Objekt, ggf. Excel) |
| summarize\_dif\_analysis | Output der Funktion conduct\_dif\_analysis (bzw. das Element models des Outputs der Funktion dif\_analysis) | Tabellen mit Ergebnissen der DIF-Analysen für **mehrere** DIF-Variablen (R-Objekt, ggf. Excel) |
| build\_dif\_tr\_tables | Output der Funktionen dif\_summaries | Tabelle mit Ergebnissen für den Technical Report (R-Objekt, ggf. Excel) |

Wie die einzelnen Funktionsargumente genutzt werden, kann man über die Hilfe-Funktion (z.B. ?dif\_summary) herausfinden.

# Distraktorenanalyse

## Allgemeine Information

In Aufgaben, in denen die getestete Person zwischen mehreren Antworten auswählen soll (Multiple-Choice), werden sog. Distraktoren eingesetzt. Hierbei handelt es sich um falsche Antworten, die jedoch wie richtige Antwortalternativen konstruiert sind. Eine Analyse der Distraktoren sollte im Rahmen der Testevaluation genutzt werden, um schlechte Distraktoren zu identifizieren. Dies ist wichtig, da schlechte Distraktoren Modellannahmen verletzen und zu Problemen in der Testinterpretation führen können. Im Rahmen der Distraktorenanalyse werden punkt-biserale Korrelationen zwischen den Distraktoren und dem Gesamtscore der untersuchten Person berechnet. Die Distraktorenanalyse sollte nur für alle validen Fälle (Funktionsargument valid) durchgeführt werden. Sie erfolgt meistens anhand der gesamten Stichprobe und nicht nach Gruppen getrennt.

## Übergreifende Funktion

Die Funktion dis\_analysis() berechnet die Korrelationen zwischen den korrekten bzw. falschen Itemantworten (Distraktoren) und dem Gesamtscore. Die Distraktoren werden für alle ungescorten Multiple-Choice Items analysiert (Funktionsargument select\_raw). Im Rahmen der Datenvorbereitung wurde dafür bereits eine Variable in vars erstellt, sie heißt in unserem Beispiel distan. Der Output der Funktion besteht zum einem aus den zusammengefassten Ergebnissen, in denen das Minimum und Maximum der Korrelationen der richtigen Antworten bzw. der Distraktoren mit dem Gesamtscore angegeben werden. Zum anderen wird eine Tabelle erstellt, in welcher die Anzahl der Antworten und die Korrelationen der Itemantworten mit dem Gesamtscore für die jeweiligen Aufgaben aufgeschlüsselt werden. Beides wird als Excel-Tabelle und im RDS-Format gespeichert. Die Zusammenfassung wird zudem in der Konsole ausgegeben.

NEPSroutines::dis\_analysis(  
 resp = resp,  
 vars = vars,  
 valid = "valid",  
 select\_raw = "distan",  
 print = TRUE,  
 save = FALSE,  
 overwrite = TRUE,  
 path\_table = here::here("Tables"),  
 path\_results = here::here("Results"),  
)

Die Funktion beinhaltet folgende (teilweise optionale) besondere Funktionsargumente:

| **Argument** | **Art** | **Beschreibung** | **Default** |
| --- | --- | --- | --- |
| select\_raw (nicht optional) | string | Name der Variablen in vars, die alle (Rohformat-)Items identifiziert, die für die Distraktorenanalyse verwendet werden sollen. |  |
| select\_score | string | Name der Variablen in vars, die alle Items identifizert, die zur Berechnung des Gesamtscores verwendet werden sollen. | “dich” |
| correct | string | Name der Variablen in vars, die die korrekten Antworten der Items enthält. | “correct\_response” |

## Untergeordnete Funktionen

Neben der übergreifenden Funktion können noch die folgenden untergeordneten Funktionen verwendet werden:

| **Funktion** | **Input** | **Output** |
| --- | --- | --- |
| conduct\_dis\_analysis | Dataframes resp und vars mit Informationen über Antworten, Items und Personenmerkmale | Ergebnisse der Distraktorenanalyse (R-Objekt, ggf. RDS) |
| dis\_summary | Output der Funktion conduct\_dis\_analysis (bzw. das Element analysis des Outputs der Funktion dis\_analysis) | Tabellen mit Ergebnissen der Distraktorenanalyse (R-Objekt, ggf. Excel) |
| print\_dis\_summary | Output der Funktion dis\_summary (bzw. das Element summary des Outputs der Funktion dis\_analysis) | Zusammenfassung der Distraktorenanalyse (in Konsole) |

Wie die einzelnen Funktionsargumente genutzt werden, kann man über die Hilfe-Funktion (z.B. ?dis\_summary) herausfinden.

# Dimensionalitätsanalyse

## Allgemeine Information

Die Dimensionalität eines Tests gibt an, wie viele latente Dimensionen für die Beantwortung des Tests benötigt werden. Sie wirkt sich auf die verwendeten IRT-Modelle aus. So ist beispielsweise eine Voraussetzung des Rasch-Models, dass Eindimensionalität gegeben ist.

Um diese Voraussetzung zu überprüfen, werden sowohl ein eindimensionales als auch mehrdimensionale Modelle berechnet und miteinander verglichen. Als Dimensionalitätsvariablen kommen Itemeigenschaften infrage, die die Items in Subgruppen aufteilen lassen. Das könnten z.B. übergreifende Aufgaben mit mehreren Items sein, der Aufgabentyp oder bei Lesetests der Texttyp. Die Prüfung basiert einerseits auf der Analyse von Varianzen und Korrelationen der Dimensionen des angenommenen mehrdimensionalen Modells. Für den Vergleich der Modelle werden erneut die Informationsmaße AIC und BIC verwendet.

Die Analysen laufen sehr lange in R (mehrere Stunden), daher ist es sinnvoll, diese an einem besonders leistungsstarken Computer durchlaufen zu lassen. Gerade die mehrdimensionalen Analysen sind sehr rechenaufwändig, weshalb hier statt Quadratur-Verfahren auf Monte-Carlo-Verfahren zurückgegriffen wird. Zum Probedurchlauf der Syntax kann zudem zunächst snodes = 10 verwendet werden, aber für die endgültige Schätzung wieder eine höhere Anzahl wählen (Default snodes = 5000).

Die Dimensionalitätsanalyse sollte nur für alle validen Fälle (Funktionsargument valid) durchgeführt werden. Sie erfolgt meistens anhand der gesamten Stichprobe und nicht nach Gruppen getrennt.

## Übergreifende Funktion

Im NEPSroutines Paket wird mit Hilfe der Funktion dim\_analysis() die Dimensionalität eines Tests geprüft. Die Funktion kann dabei für mehrere Dimensionalitätsvariablen gleichzeitig (Funktionsargument dim) Modelle berechnen und miteinander vergleichen. Sie führt die Analysen durch und gibt die benötigten Ergebnisse in Form einer Tabelle und weiterer Informationen in der Konsole aus. Zudem speichert sie die Ergebnisse als Excel-Tabelle und im RDS-Format. Im Beispiel wird ein mehrdimensionales Modell für den Aufgabentyp und ein mehrdimensionales Modell für den Texttyp geschätzt.

NEPSroutines::dim\_analysis(  
 resp = resp,  
 vars = vars,  
 select = 'mixed',  
 valid = 'valid',  
 dim = c("kog\_type", "text\_type"),  
 scoring = "scoring",  
 irtmodel = "PCM2",  
 verbose = TRUE,  
 return = FALSE,  
 save = TRUE,  
 print = TRUE,  
 path\_table = here::here("Tables"),  
 path\_results = here::here("Results"),  
 overwrite = TRUE  
)

Die Funktion beinhaltet folgende (teilweise optionale) besondere Funktionsargumente:

| **Argument** | **Art** | **Beschreibung** | **Default** |
| --- | --- | --- | --- |
| dim (nicht optional) | character vector | Enthält die Namen aller potentiellen Dimensionalitätsvariablen in vars. Die Dimensionalitätsvariablen selber müssen numerisch mit Integern von 1 bis zur Anzahl der Dimensionen kodiert werden! |  |
| scoring | string | Name der numerischen Variable in vars, die die Scoring-Werte der Items beinhaltet. | “scoring” |
| irtmodel (nicht optional) | string | Name des zugrundeliegenden Modells. Hierfür sind die Argumente “1PL”, “2PL”, “PCM2” und “GPCM” verwendbar. |  |
| maxiter | integer | Funktionsargument maxiter (maximale Anzahl an Iterationen) für die TAM-Funktionen. | 10000 |
| snodes | integer | Funktionsargument snodes (Anzahl der snodes) für die TAM-Funktionen. | 5000 |

## Untergeordnete Funktionen

Neben der übergreifenden Funktion können noch die folgenden untergeordneten Funktionen verwendet werden:

| **Funktion** | **Input** | **Output** |
| --- | --- | --- |
| conduct\_dim\_analysis | Dataframes mit Informationen über Items und Personenmerkmale, sowie der Spezifizierung der untersuchten Items. Zudem muss das IRT-Modell wie auch die scoring-Informationen, welche im Rahmen der Vorbereitung der Daten in vars erstellt wurde, angegeben werden | Ergebnisse der Dimensionsanalyse (R-Objekt, ggf. RDS) |
| dim\_summary | Output der Funktion conduct\_dim\_analysis (bzw. das Element analysis des Outputs der Funktion dim\_analysis) | Tabellen mit Ergebnissen der Dimensionsanalyse (R-Objekt, ggf. Excel) |
| print\_dim\_summary | Output der Funktion dim\_summary (bzw. das Element summary des Outputs der Funktion dim\_analysis) | Zusammenfassung der Dimensionsanalyse (in Konsole) |

Wie die einzelnen Funktionsargumente genutzt werden, kann man über die Hilfe-Funktion (z.B. ?dim\_summary) herausfinden.

# Linking

In dieser Version des NEPSroutines Pakets ist das Linking noch nicht implementiert.

# Scores berechnen

Am Schluss der Skalierung werden die **querschnittlichen** und **längsschnittlichen WLEs** geschätzt und für den SUF aufbereitet. Die WLEs werden hierbei prinzipiell wie in der Rasch- bzw. PCM-Analyse geschätzt. Auch wenn es verschiedene Testhefte gibt, werden alle WLEs auf Basis eines gemeinsam skalierten Modells (also nicht getrennt nach Testheften) berechnet. Liegt ein Rotationsdesign vor, so wird im Querschnitt dafür korrigiert, um unverzerrte Fähigkeitsschätzungen zu erhalten. Dafür muss das Argument facet entsprechend spezifiziert werden.

Für die Schätzung der **querschnittlichen WLEs** können die ***fixierten Itemparameter*** verwendet werden. Diese Itemparameter entsprechen den geschätzten Itemparametern des korrespondierenden Modells und sind im Objekt, das nach dem Ausführen der Funktion irt\_analysis erstellt wird, verfügbar (zum Beispiel unter irt\_poly\_all$model.pcm$mod$xsi$xsi). Dieses Objektelement soll im Argument xsi\_fixed angegeben werden. Beachten Sie, dass die fixierten Parameter als named numeric vector definiert werden sollten. Sie können dies beispielsweise wie folgt tun: xsi\_fixed = row.names(irt\_poly\_all$model.pcm$mod$xsi). Wenn Sie das Argument xsi\_fixed nicht spezifizieren, wird die Funktion create\_scores() automatisch die Itemparameter zuerst schätzen und sie dann für die Schätzung von querschnittlichen WLEs verwenden.

Für den Fall, dass mehr als ein Messzeitpunkt vorliegt, müssen darüber hinaus ***verlinkte Itemparameter*** genutzt werden. Das Linking ist in dieser Version des NEPSroutines Pakets jedoch leider noch nicht implementiert. In diesem Fall müssen die Scores also ohne das Paket berechnet werden.

Die Funktion create\_scores() gibt eine Tabelle mit den neu berechneten WLEs (Variable mit der Endung \_sc1 und die korrespondierenden SEs mit der Endung \_sc2) aus. Wenn man diese WLEs später in den SUF integrieren möchte (was normalerweise der Fall ist), sollte diese Tabelle in R als Objekt zwischengespeichert werden. Per Default werden die berechneten Scores im RDS-Format gespeichert. Zusätzlich werden die Itemparameter, die zur Schätzung der WLEs verwendet wurden, im XLSX-Format sowie das entsprechende TAM-Modell im RDS-Format abgespeichert.

Das Paket erlaubt außerdem, zusätzlich zu den WLEs die Summenscores und Metakognitionsscores (Variablen mit den Endungen \_sc5 und \_sc6) zu berechnen. Zur Berechnung der **Summenscores** (Variable mit der Endung \_sc3) werden alle im Test verwendeten, originalen Items benötigt. Im Rahmen der Datenvorbereitung (s. Abschnitt 4.3.3 *Variablen für die Scores und den SUF*) sollte daher im Datensatz vars bereits eine Variable angelegt worden sein, die alle dichotomen und polytomen Items identifiziert, die im Test verwendet wurden und die anschließend in den SUF aufgenommen werden müssen. Es handelt sich dabei um gescorte Items ohne kollabierte Kategorien. Im Beispiel heißt die Variable suf. Zur Berechnung der Summenscores muss das Argument sum\_score auf TRUE gesetzt sowie die Argumente sum\_select und num\_cat spezifiziert werden. Die berechneten Summenscores werden als Variable mit der Endung \_sc3 gespeichert.

Um die **Metakognitionsscores** berechnen zu lassen, muss das Argument metap auf TRUE gesetzt und die Argumente meta\_variable, num\_cat, meta\_score\_name und meta\_select spezifiziert werden. Fehlt eines der ersten beiden Argumente, gibt die Funktion eine Fehlermeldung aus. Fehlt meta\_score\_name oder meta\_select, werden stattdessen die Strings aus score\_name und select verwendet. Die berechneten Metakognitionsscores werden als Variablen mit den Endungen \_sc5 und \_sc6 gespeichert. Beachte, dass die Metakognitionsscores im Gegensatz zu den WLEs und Summenscores getrennt nach Testheften berechnet werden. Daher empfiehlt es sich, die Metakognitionsscores getrennt von den anderen Scores zu berechnen, um die Funktion create\_scores für beide Testhefte getrennt spezifizieren zu können (und bei select dann entsprechend z.B. suf\_easy oder suf\_difficult auswählen zu können). Wenn die im Argument meta\_variable angegebene Variable unübliche Werte für die Einschätzung der Metakognition enthält (z.B. 5-10, 5/10), dann werden diese Werte bei der Ausführung der Funktion create\_scores in NAs umgewandelt. Eine Warnmeldung wird dabei in der Konsole angezeigt.

scores <- NEPSroutines::create\_scores(  
 resp = resp,  
 vars = vars,  
 scoring = 'scoring',  
 score\_name = 'reg7',  
 facet = NULL,  
 select = 'mixed',  
 valid = 'valid',  
 wle = TRUE,  
 sum\_score = FALSE,  
 sum\_select = 'suf',  
 num\_cat = 'num\_cat',  
 metap = FALSE,  
 return = TRUE  
)

Die Funktion beinhaltet folgende (optionale) besondere Funktionsargumente:

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Art** | **Beschreibung** | **Default** |
| scoring | string | Name der numerischen Variable in vars, die die Scoring-Werte der Items beinhaltet. Nur bei PCM-Analysen benötigt. | NULL |
| score\_name | string | Name der WLEs *ohne* Suffix, z.B. *reg7* anstatt *reg7\_sc1*. | „score“ |
| xsi\_fixed | named numerical vector | Elemente des Vektors sind die Itemschwierigkeiten (Output der Funktion TAM::tam.mml unter mod$xsi$xsi). Namen der Elemente sind die Itemnamen. | NULL |
| facet | string | Name der Variablen in resp, die die Testrotation angibt. | NULL |
| wle | logical | Ob WLEs berechnet werden sollen. | TRUE |
| sum\_score | logical | Ob Summenscores berechnet werden sollen. | FALSE |
| sum\_select | string | Name der Variable in vars, die alle zur Berechnung der Summenscores benötigten Items identifiziert. Falls leer, werden stattdessen die Items aus select verwendet. | NULL |
| num\_cat | string | Name der numerischen Variable in vars, die die maximale Anzahl von Antwortmöglichkeiten für jedes Item enthält. Muss angegeben werden, wenn Metakognitions- oder Summenscores berechnet werden sollen. | „num\_cat“ |
| metap | logical | Ob der Metakognitionsscore berechnet werden soll. | FALSE |
| meta\_variable | string | Name der Metakognitionsvariable in resp. Muss angegeben werden, wenn Metakognitionsscore berechnet wird. | NULL |
| meta\_score\_name | string | Name des Metakognitionsscores *ohne* Suffix, z.B. *mpg7* anstatt *mpg7\_sc5*. Falls identisch mit score\_name, kann das Argument weggelassen werden. | score |
| meta\_select | string | Name der Variable in vars, die alle zur Berechnung des Metakognitionsscores benötigten Items identifiziert. Falls leer, werden stattdessen die Items aus select verwendet. | NULL |
| control\_tam | list | Funktionsargument control für die Funktion TAM::tam.mml.mfr. | NULL |
| control\_wle | list | Liste mit den Fuktionsargumenten Msteps und/oder convM für die Funktion TAM::tam.wle als Listenelemente. | NULL |
| pweights | numeric vector | Personengewichte für die aktuellen Messzeitpunkte, die als Funktionsargument pweights den TAM-Funktionen übergeben werden. | NULL |
| poly2dich | logical | Ob nur korrekt bewertete dichotome oder auch vollständig korrekt bewertete polytome Items gezählt werden. Per Default werden polytome Items als dichotome gescort. | TRUE |

# SUF erstellen

## Übergreifende Funktion

Mit der Funktion create\_suf() wird der SUF mit den neu berechneten WLEs erstellt. Beachte, dass alle dichotomen und polytomen Items, die im Test verwendet wurden, in den SUF aufgenommen werden müssen. Es handelt sich dabei um die gescorten Items. Polytome Variablen mit kollabierten Kategorien sollen nicht in den SUF aufgenommen werden. Im Rahmen der Datenvorbereitung sollte daher schon eine logische Variable erstellt worden sein, die die entsprechenden Items identifiziert (s. Abschnitt 4.3.3 *Variablen für die Scores und den SUF*). Im Beispiel heißt die Variable suf.

NEPSroutines::create\_suf(  
 resp = resp,  
 vars = vars,  
 select = 'suf',  
 competence = 'Reading',  
 scores = scores,  
 score\_name = 'reg7',  
 path = here::here("suf")  
)

Die Funktion beinhaltet folgende (teilweise optionale) besondere Funktionsargumente:

| **Argument** | **Art** | **Beschreibung** | **Default** |
| --- | --- | --- | --- |
| competence (nicht optional) | string | Name der getesteten Kompetenz für die Bezeichnung der Daten. |  |
| scores | data.frame | Dataframe mit ID\_t, den WLEs und ggf. Summenscores und Metakognitionsscores als Variablen und den Personen als Reihen (Output der Funktion create\_scores). | NULL |
| score\_name | string | Name der WLEs (*ohne* Suffix, z.B. *reg7* anstatt *reg7\_sc1*). | NULL |
| path | character vector | Name des Ordners, in dem die SUF-Datei gespeichert werden soll. | “suf” |
| items\_labels | named character vector | Elemente sind die Labels, Namen der Elemente die entsprechenden Itemnamen. | NULL |

## Untergeordnete Funktionen

Neben der übergeordneten Funktion create\_suf() kann auch die untergeordnete Funktion set\_lables() verwendet werden. Mit ihrer Hilfe können die Labels der Items im SUF geändert werden. Sie gibt einen Datensatz aus, der den SUF mit den geänderten Labels beinhaltet. Wie die einzelnen Funktionsargumente genutzt werden, ist über die Hilfe-Funktion (?set\_labels) einzusehen.

# Literatur

* Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee’s ability. Statistical theories of mental test scores.
* Fischer, L., Rohm, T., Gnambs, T., & Carstensen, C. H. (2016). Linking the data of the competence tests (NEPS Survey Paper No. 1). Bamberg: Leibniz Institute for Educational Trajectories, National Educational Panel Study.
* Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.
* Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *ETS Research Report Series, 1992*(1).
* Pohl, S. & Carstensen, C. H. (2012): NEPS Technical Report: Scaling the Data of the Competence Tests (NEPS Working Paper No. 14). Bamberg: Otto-Friedrich-Universität, Nationales Bildungspanel.
* Rasch, G. (1960). Studies in mathematical psychology: I. Probabilistic models for some intelligence and attainment tests.