Package 'PROscorerTools'

May 15, 2017

Type Package

Title Tools to Score Patient-Reported Outcome (PRO) and Other Psychometric Measures

Version 0.0.1

Description Provides a reliable and flexible toolbox to score patient-reported outcome (PRO), Quality of Life (QOL), and other psychometric measures. The guiding philosophy is that scoring errors can be eliminated by using a limited number of well-tested, well-behaved functions to score PRO-like measures. The workhorse of the package is the 'scoreScale' function, which can be used to score most single-scale measures. It can reverse code items that need to be reversed before scoring and pro-rate scores for missing item data. Currently, three different types of scores can be output: summed item scores, mean item scores, and scores scaled to range from 0 to 100. The 'PROscorerTools' functions can be used to write new functions that score more complex measures. In fact, 'PROscorerTools' functions are the building blocks of the scoring functions in the 'PROscorer' package (which is a repository of functions that score specific commonly-used instruments). Users are encouraged to use 'PROscorerTools' to write scoring functions for their favorite PRO-like instruments, and to submit these functions for inclusion in 'PROscorer' (a tutorial vignette will be added soon). The long-term vision for the 'PROscorerTools' and 'PROscorer' packages is to provide an easy-to-use system to facilitate the incorporation of PRO measures into research studies in a scientifically rigorous and reproducible manner. These packages and their vignettes are intended to help establish and promote ``best practices'' for scoring and describing PRO-like measures in research.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

URL http://github.com/raybaser/PROscorerTools

BugReports http://github.com/raybaser/PROscorerTools/issues

RoxygenNote 6.0.1

Suggests testthat, knitr, rmarkdown, covr

VignetteBuilder knitr

NeedsCompilation no

Author Ray Baser [aut, cre]

Maintainer Ray Baser <ray.stats@gmail.com>

Repository CRAN

Date/Publication 2017-05-15 05:25:34 UTC

R topics documented:

`chk_nitems` *Checks the number and values of items passed to custom scoring functions*

Description

These functions are designed to used within custom scoring functions to help check the arguments passed to those functions. Typically, these argument checkers will be used within the body of a custom scoring function before calling the `scoreScale` function to handle the bulk of the work. See Details.

- `chk_nitems` checks if `dfItems` contains the correct number of items (`nitems`), and `chkstop_nitems` returns an error message if this condition is not met.
- `chk_values` checks if all of the item values in `dfItems` are in the set of possible values given to the `values` argument, and `chkstop_values` returns an error message if this condition is not met.

Usage

```
chk_nitems(dfItems, nitems)

chkstop_nitems(dfItems, nitems)

chk_values(dfItems, values)

chkstop_values(dfItems, values)
```

Arguments

dfItems	A data frame with only the items to be scored.
nitems	The number of items on the scale to be scored.
values	A vector of all of the possible values that the items can take.

Details

Functions with prefix chk_ simply check whether their argument values meet a condition and return TRUE or FALSE. Functions with the prefix chkstop_ check the arguments and, if FALSE, stop execution and display an error message to help the user pinpoint the problem.

The scoreScale function is a general, all-purpose tool that can be used to score a scale regardless of the number or values of items on the scale. Because of this, however, it does not check that the user has given it the correct number of items, and it does not check that those item values are all within the range possible for that scale. Therefore, whenever scoreScale is used to write a function to score a specific instrument (presumably with a known number of items and item values), the programmer should run some additional checks on the arguments that are not already built-in to scoreScale.

Value

Functions with prefix chk_ return TRUE if the arguments **pass** the argument checks, or FALSE if the arguments **fail** the checks. Functions with the prefix chkstop_ print an error message and stop the execution of the function in which they are embedded.

Note

Use with caution! These functions work, but they might be deprecated in future updates of the package. I am hoping to come up with a more streamlined, user-friendly system for checking arguments and input values. Until then, these functions get the job done, but not as gracefully as I would like.

Examples

```
itemBad <- c(0, 1, 2, 3, 10)
itemGood <- c(0, 1, 2, 3, 0)
dfBad <- data.frame(itemBad, itemGood)
dfGood <- data.frame(itemGood, itemGood)
chk_nitems(dfBad, 1)
chk_nitems(dfGood, 2)
```

```
chk_values(dfBad, 0:3)
chk_values(dfGood, 0:3)
```

`get_dfItems` *Get a data frame with only items from user input*

Description

Given a data frame and an item index, returns a data frame containing only the items. These functions are used internally by other PROscorerTools functions, particularly `scoreScale`. Their job is to return a data frame containing only the items to be scored. These functions are also used in the scoring functions in the **PROscorer** package to help process the user's input. These functions will be of interest mainly to developers wishing to contribute to the **PROscorer** package.

Usage

```
get_dfItems(df, items)

get_dfItemsrev(df, dfItems, revitems, minmax)
```

Arguments

`df`	A data frame given as the argument to `scoreScale`
`items`	Item index, given as argument to `scoreScale`
`dfItems`	A data frame with only items, created and used by `scoreScale` as an interim step in scoring a scale
`revitems`	Items to reverse, given as argument to `scoreScale`
`minmax`	Minimum and maximum possible item values, given as argument to `scoreScale`

Value

These functions return a data frame containing only the items to be scored. In the case of `get_dfItemsrev`, the specified items will be reverse scored in the returned data frame.

makeFakeData *Make a data frame of fake item data*

Description

makeFakeData creates a data frame containing fake item data to facilitate the writing and testing of new scoring functions. It is also used to create data for examples of scoring function usage.

Usage

```
makeFakeData(n = 20, nitems = 9, values = 0:4, propmiss = 0.2,
  prefix = "q", id = FALSE)
```

Arguments

n	The number of respondents (rows) in the fake data. The default is 20.
nitems	The number of items in the fake data. The default is 9.
values	A vector of all possible values the items can take. The default is 0:4, or equivalently c(0, 1, 2, 3, 4).
propmiss	The proportion of responses that will be randomly assigned to be missing. The default is .20.
prefix	A quoted character that will be used to prefix the item numbers. The default is "q".
id	Logical, if TRUE the first variable in the data frame will be a unique row "ID". The default is FALSE, and the "ID" variable is omitted.

Details

The item responses in the first row are all the lowest possible value and never NA, and the responses on the second row are all the highest possible value and never NA. This makes it easier to check if the scoring function is at least getting the scores correct for subjects with no missing values. It also makes it easier in some cases to check that the scoring function is properly reversing the items according to the itemsrev argument of the scoring function.

Although the resulting data frame can be customized using the arguments, the default values are sufficient for most generic testing purposes (see example).

Value

A data frame with n rows, nitems items, and possibly with some missing values randomly inserted.

Examples

```
makeFakeData()
```

makeItemNames *Quickly create a vector of sequentially numbered item names*

Description

Takes a prefix (e.g., "Q") and the number of items you want (e.g., 3), and returns a vector of item names (e.g., c("Q1", "Q2", "Q3")).

Usage

```
makeItemNames(prefix, nitems)
```

Arguments

prefix	A quoted prefix that will precede the number in the item name (e.g., the "Q" in "Q1").
nitems	The number of items

Value

A character vector of sequentially numbered item names.

Examples

```
makeItemNames("q", 3)
itemNames <- makeItemNames("item", 7)
itemNames
```

missTally *Determine the number (or proportion) of missing (or non-missing) items for each respondent*

Description

This is a handy helper function that can be used in PRO scoring functions to determine the number (or proportion) of item responses that are missing (or valid) for each row in a data frame of items. This is used by scoreScale to help determine if a respondent has answered enough items to be assigned a prorated scale score.

Usage

```
missTally(dfItems, what = c("pmiss", "nmiss", "pvalid", "nvalid"))
```

Arguments

dfItems	A data frame containing only the items of interest.
what	One of four quoted names indicating the value you want for each respondent (row) in dfItems: (1) "pmiss" (the default), for the proportion of items that are missing; (2) "nmiss", for the number of items that are missing; (3) "pvalid", for the proportion of items that are valid, non-missing; and (4) "nvalid" for the number of items that are valid, non-missing.

Value

A vector of length nrow(dfItems) that contains the quantity requested in what for each row of dfItems.

Examples

```
set.seed(8675309)
#  Make data frame with 10 respondents, 10 items, and approx 30% missing data
(myItems <- makeFakeData(n = 10, nitems = 10, propmiss = .30))
#  The default is to return "pmiss", the proportion missing for each row.
missTally(myItems)
missTally(myItems, "pvalid")
missTally(myItems, "nmiss")
missTally(myItems, "nvalid")
```

---

msgWrap *msgWrap*

Description

Helps format line-wrapping of long error and warning messages

Usage

```
msgWrap(msg)
```

Arguments

msg	A quoted message.

Details

This is just a shorter version of the following that makes my function code easier to read: paste(strwrap(msg, exdent = 2,
It seems to work fine when embedded in warning or stop, but may give unexpected output if called alone.

Value

It returns msg formatted wrapped nicely for the console.

Examples

```
txt <- "If you use 'itemsrev' to indicate items that
        must be reverse-coded before scoring,
        you must provide a valid numeric range to 'minmax'.
        For example, if your lowest possible item response
        is 0 and your highest possible response is 4,
        you would use 'minmax = c(0, 4)'."

warning(msgWrap(msg = txt))
```

PROscorerTools *PROscorerTools*

Description

Tools to score Patient-Reported Outcome (PRO), Quality of Life (QOL), and other psychometric measures and questionnaire-based instruments.

Details

Provides a set of reliable and flexible tools to score PRO, QOL, and other psychometric and psychological measures. Additionally, **PROscorerTools** provides the infrastructure for the scoring functions in the **PROscorer** package.

The `scoreScale` function is the workhorse of **PROscorerTools**, and it can be used to score most single-scale measures. For example, it can reverse code some or all items before scoring, and it can generate different types of scores (sum scores, mean scores, or scores scaled to range from 0 to 100). It and the other **PROscorerTools** functions can be used together to flexibly write new functions that score more complex, multi-scale measures.

The `scoreScale` function itself is composed of other **PROscorerTools** helper functions. This is an intentional feature of the **PROscorerTools** and **PROscorer** system. It represents the central design philosophy that all scoring functions should be modularly composed of a small number of well-tested, reliable helper functions. This important feature minimizes the possibility of scoring errors and other unexpected behaviors. This starts with the `scoreScale` function, and the benefits extend to the **PROscorer** functions and any other scoring function that uses `scoreScale` as its backbone.

Scoring procedures represent a major source of error in research studies that rely upon PRO and similar measures. These errors typically go unnoticed, hidden, and/or ignored, eroding the scientific integrity of the research and hindering progress in the numerous scientific fields that conduct studies that use these measures. A seemingly minor scoring error can compromise measurement validity and reliability, as well as make research results difficult to reproduce and unlikely to replicate. The ultimate goal of the **PROscorerTools** and **PROscorer** packages is to eliminate these serious deficiencies in PRO-based research by providing a small set of gold-standard scoring tools for PRO-like measures commonly used in research.

Overview of Functions

Main scoring workhorse:

- scoreScale: Scores a single scale score from a set of items. Has flexible arguments allowing almost any type of single scale to be scored. Can be used as the primary building block of more complex scoring functions. For example, if you are writing a new function to score an instrument with 4 subscales, you can call scoreScale 4 times from within your function, once for each subscale.

Functions useful within another scoring function, or on their own:

- missTally: Count the number or proportion of items missing (or non-missing) for each subject.
- rerange: Linearly rescale a variable to have new min and max values of the user's choosing.
- rerange100: Like rerange, but the variable is rescaled to range from 0 to 100.
- revcode: Reverse code the values of an item or score.
- makeItemNames: Easily make a vector of item names, e.g., `c("Q1", "Q2", "Q3")`.
- makeFakeData: Quick and dirty way to generate fake item data to test scoring functions.

Functions mainly useful only within another scoring function:

These functions are used to check the arguments supplied to custom-written scoring functions, and to perform some minimal processing of function input. **AT THIS TIME, USE WITH CAUTION, IF AT ALL.** Most of these (if not all) will likely undergo substantial changes in a near-future version of the package, and/or be deprecated in favor of a more streamlined system (e.g., using the **assertive** package).

- msgWrap: Used inside of `paste` to help line-wrap long error and warning messages.
- chk_nitems: Checks if a data frame has the correct number of items.
- chkstop_nitems: Checks if a data frame has the correct number of items, and gives an error message if it does not.
- chk_values: Checks if all item values in a data frame are in the expected set of possible values.
- chkstop_values: Checks if all item values in a data frame are in the expected set of possible values, and gives an error message if this is not true.
- get_dfItems: Given a data frame and an item index, returns a data frame containing only the items.
- get_dfItemsrev: Like get_dfItems, but will also reverse code some or all of the items.

Internal Functions Used by scoreScale

These are internal functions used to make the scoreScale function more modular, and are used exclusively to check the arguments to that function. They will likely be of little use to others, and will probably change in a near-future version of the package, or be deprecated altogether in favor of a more streamlined system (e.g., using the **assertive** package). They are documented here only for development purposes. **NOTE**: The interface and functionality of the scoreScale function will remain stable, even if/when these functions change.

- chkstop_df:
- chkstop_okmiss:
- chkstop_type:
- chkstop_revitems:
- chk_imin:
- chk_imax:
- chkstop_minmax:

---

rerange *Change the range of an item or score*

Description

- rerange linearly rescales a numeric variable to have new minimum and maximum values of the user's choosing.
- rerange100 is a simplified version of rerange that rescales a variable to range from 0 to 100.

Usage

```
rerange(score, old = NULL, new = c(0, 100), rev = FALSE)

rerange100(score, mn, mx)
```

Arguments

score	The variable to be re-ranged.
old	A numeric vector of length 2 indicating the old range (e.g., c(min, max) score. This is a required argument.
new	A numeric vector of length 2 indicating the new range you want for score. The default value is c(0, 100).
rev	Logical, if TRUE score will will not only be re-ranged, but it will also be reversed (see Details for more information). The default is FALSE.
mn	The minimum possible value that score can take.
mx	The maximum possible value that score can take.

Details

The rerange function can re-range and reverse code a variable all at once. If rev = TRUE, score will be reversed using revcode after it is re-ranged. The same could be accomplished by keeping rev = FALSE and reversing the order of the range given to new. For example, the following two calls to rerange will return the same values:

- rerange(score, old = c(0, 10), new = c(0, 100), rev = TRUE)
- rerange(score, old = c(0, 10), new = c(100, 0), rev = FALSE)

The rerange100 function is a short-cut for rerange with the arguments set to the values typically used when scoring a PRO measure. Specifically, rerange100 is defined as:

- rerange(score, old = c(mn, mx), new = c(0, 100), rev = FALSE)

These functions can produce verbose warning messages. If you are using this function within another function, you can suppress these messages by wrapping your call to rerange in suppressWarnings().

Value

A re-ranged vector.

A version of score that is rescaled to range from 0 to 100.

Examples

```
qol_score <- c(0:4)

# Default is to rerange to c(0, 100)
rerange(qol_score, old = c(0, 4))
# Below gives same result as above
rerange100(qol_score, 0, 4)

# These two lines are different ways to rerange and reverse code at same time
rerange(qol_score, old = c(0, 4), new = c(0, 100), rev = TRUE)
rerange(qol_score, old = c(0, 4), new = c(100, 0))
```

---

revcode *Reverse code an item or score.*

Description

Given an item (or score) and the minimum and maximum possible values that the item can take, this helper function reverse codes the item. For example, it turns c(0, 1, 2, 3, 4) into c(4, 3, 2, 1, 0).

Usage

```
revcode(x, mn, mx)
```

Arguments

x	A single item (or score) to reverse code.
mn	The minimum possible value that x can take.
mx	The maximum possible value that x can take.

Details

The user must supply the *theoretically possible* minimum and maximum values to this function (using mn and mx, respectively). Some similar functions do not require users to provide the minimum and maximum values. Instead, those functions calculate the minimum and maximum values from the data. However, in cases where not all of the possible item values are contained in the data, this would incorrectly reverse score the items. In the interest of scoring accuracy, these arguments are required for revcode.

Value

A vector the same length as x, but with values reverse coded.

Examples

```
item1 <- c(0, 1, 2, 3, 4)
revcode(item1, 0, 4)
item2 <- c(0, 1, 2, 3, 0)
revcode(item2, 0, 4)
```

---

scoreScale *Flexible function to score a single PRO or other psychometric scale*

Description

scoreScale is a flexible function that can be used to calculate a single scale score from a set of items.

Usage

```
scoreScale(df, items = NULL, revitems = FALSE, minmax = NULL,
  okmiss = 0.5, type = c("pomp", "100", "sum", "mean"),
  scalename = "scoredScale", keepNvalid = FALSE)
```

Arguments

df	A data frame containing the items you wish to score. It can contain only the items, or the items plus other non-scored variables. If it contains non-scored variables, then you must use the items argument to let the function know how to find your items in df.
items	(optional) A character vector with the item names, or a numeric vector indicating the column numbers of the items in df. If items is omitted, then scoreScale will assume that df contains only the items to be scored and no non-scored variables.
revitems	(optional) either TRUE, FALSE, or a vector indicating which items in df should be reverse coded before scoring. If omitted or FALSE (the default), no items are reverse coded. If TRUE, all items are reverse coded before scoring. If only some of the items should be reverse coded, provide either a character vector

with names of the items or a numeric vector with column numbers of the items in `df` that should be reverse coded before scoring. If this argument is anything but `FALSE`, then the `minmax` argument is required.

`minmax` (optional) A vector of 2 integers of the format `c(itemMin, itemMax)`, indicating the minimum and maximum possible item responses, e.g., `c(0, 4)`. This argument is required if `type` equals `"pomp"` (the default `type`) or `"100"`. This is also required only `revitems` is used and not set to `FALSE`. This function assumes that all items have the same response range. If this is not the case, then manually reverse code your items in `df` before using this function, and omit the `revitems` and `minmax` arguments.

`okmiss` The maximum proportion of items that a respondent is allowed to have missing and still have their non-missing items scored (and prorated). If the proportion of missing items for a respondent is greater than `okmiss`, then the respondent will be assigned a value of `NA` for their scale score. The default value is `0.50`.

`type` The type of score that `scoreScale` should produce. Must be one of either `"sum"` (for the sum of the item scores), `"mean"` (for the mean of the item scores), `"100"` (for the score transformed to range from 0 to 100), or `"pomp"` (for a score representing the "Percent Of the Maximum Possible", which is exactly the same as `"100"` but with a better name). The default is `"pomp"`.

`scalename` The quoted variable name you want the function to give your scored scale. If this argument is omitted, the scale will be named `"scoredScale"` by default.

`keepNvalid` Logical value indicating whether a variable containing the number of valid, non-missing items for each respondent should be returned in a data frame with the scale score. The default is `FALSE`. Set to `TRUE` to return this variable, which will be named `"scalename_N"` (with whatever name you gave to the `scalename` argument). Most users should probably omit this argument entirely. This argument might be removed from future versions of the package, so please let me know if you think this argument useful and would rather it remain a part of the function.

Details

The `scoreScale` function is the workhorse of the **PROscorerTools** package, and it is intended to be the building block of other, more complex scoring functions tailored to specific PRO measures. It can handle items that need to be reverse coded before scoring, and it has options for handling missing item responses. It can use three different methods to score the items: (1) sum scoring (the sum of the item scores), mean scoring (the mean of the item scores), and 0-100 scoring (like sum or mean scoring, except that the scores are rescaled to range from 0 to 100). This latter method is also called "POMP" scoring (Percent Of the Maximum Possible), and is the default scoring method of `scoreScale` since it has numerous advantages over other scoring methods (see References).

This function assumes that all items have the same numeric response range. It can still be used to score scales comprised of items with different response ranges with two caveats:

- First, if your items have different ranges of possible response values AND some need to be reverse coded before scoring, you should not use this function's `revitems` plus `minmax` arguments to reverse your items. Instead, you should manually reverse code your items (see `revcode`) before using `scoreScale`, and omit the `revitems` and `minmax` arguments.

- Second, depending on how the different item response options are numerically coded, some items might contribute more/less to the scale score than others. For example, consider a questionnaire where the first item has responses coded as "0 = No, 1 = Yes" and the rest of the items are coded as "0 = Never, 1 = Sometimes, 2 = Always". The first item will contribute relatively less weight to the scale score than the other items because its maximum value is only 1, compared to 2 for the other items. This state of affairs is not ideal, and you might want to reconsider including items with different response ranges in one scale score (if you have that option).

Value

A data frame with a variable containing the scale score. Optionally, the data frame can additionally have a variable containing the number of valid item responses for each respondent.

Further Explanation of Arguments

The `scoreScale` function technically has only 1 required argument, `df`. If none of your items need to be reverse coded before scoring, your items are in a data frame named `myData`, and `myData` contains ONLY the items to be scored and no non-scored variables, then `scoreScale(myData)` is sufficient to score your items.

In most real-world situations, however, you will likely have a data frame containing a mix of items and other variables. In this case, you should additionally use the `items` argument to indicate which variables in your data frame are the items to be scored. For example, assume that `myData` contains an ID variable named "ID", followed by three items named "Q1", "Q2", and "Q3", none of which need to be reverse coded. You can score the scale by providing the `items` argument with either **(1)** a numeric vector with the column indexes of the items, like `scoreScale(myData, items = 2:4)` or `scoreScale(myData, items = c(2, 3, 4)`, or **(2)** a character vector with the item names, like `scoreScale(myData, items = c("Q1", "Q2", "Q3")`.

References

Cohen, P, Cohen, J, Aiken, LS, & West, SG (1999). The problem of units and the circumstance for POMP. *Multivariate Behavioral Research*, 34(3), 315-346.

Examples

```
# Make a data frame using default settings of makeFakeData() function
# (20 respondents, 9 items with values 0 to 4, and about 20% missing)
dat <- makeFakeData()

# First "sum" score the items, then "mean" score them
scoreScale(dat, type = "sum")
scoreScale(dat, type = "mean")

# Must use "minmax" argument if the "type" argument is "100"
scoreScale(dat, type = "100", minmax = c(0, 4))
# If you omit "type", the default is "pomp" (which is identical to "100")
scoreScale(dat, minmax = c(0, 4))

# "minmax" is also required if any items need to be reverse coded for scoring
```

```
#  Below, the first two items are reverse coded before scoring
scoreScale(dat, type = "sum", revitems = c("q1", "q2"), minmax = c(0, 4))
```

Index