Package: questionnaires (via r-universe)

November 3, 2024

Title Package with functions to calculate components and sums for LCBC questionnaires

Version 0.0.3

Description Creates summaries and factorials of answers to questionnaires.

License MIT + file LICENSE

Encoding UTF-8

Imports dplyr (>= 1.0.0), tidyr, lubridate, lfactors, cli

Suggests testthat (>= 2.1.0), here, devtools, knitr, rmarkdown, covr, katex

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

URL https://github.com/LCBC-UiO/questionnaires

BugReports https://github.com/LCBC-UiO/questionnaires/issues

Repository https://lcbc-uio.r-universe.dev

RemoteUrl https://github.com/LCBC-UiO/questionnaires

RemoteRef HEAD

RemoteSha 2e70ecae9d4bc3a9e702686fa810c66267c5cd0c

Contents

`bdi_compute` *Calculate BDI scores*

Description

Beck Depression Inventory-II (BDI-II) is one of the most widely used instruments for measuring the severity of self-reported depression in adolescents and adults. As a general rule, BDI-II is administrated in LCBC to adults with an upper cut off around 60 years, while depression in older adults is assessed with the Geriatric Depression Scale (GDS). However, please consult the instructions for each project, as this guideline has been implemented at different time points across the projects.

The questionnaire consists of 21 statements, each reflecting a depression symptom or attitude which could be rated from 0 to 3 in terms of intensity. The answers should be based the participant´s feelings throughout the last week, including the day of filling out the form. The sum of the scores of the items (0-3) yields one total score, with a possible range between 0 and 62.

Interpretation of the scores (Total scores):

Score	Category
0-10	These ups and downs are considered to be normal
11-16	Mild mood disturbance
17 – 20	Borderline clinical disturbance
21 – 30	Moderate depression
31 – 40	Severe depression
Above 40	Extreme depression

If a participant scores `>= 17`, we should consider contacting the participant to follow up on this and offer making a note for the participant's doctor describing the scores.

Data requirements:

Column names:
By default, the functions assume that columns have names in the manner of `bdi_XX` where `XX` is a zero-padded (i.e. zero in front of numbers below 9, eg. `09`) question number of the inventory. You may have column names in another format, but in that case you will need to supply to the functions the names of those columns using tidy-selectors (see the tidyverse packages for this). The columns should adhere to some naming logic that is easy to specify.

Data values:
The values in the columns should be the item number of the question that was answered (i.e. `0`, `1`, `2`, or `3`). The inventory allows subjects to respond to several options per question, in the case of this, the mean of the responded alternatives should be applied.

References:

Aaron T.Beck, Robert A.Steer, Margery G.Carbin (1988) Psychometric properties of the Beck Depression Inventory: Twenty-five years of evaluation, *Clinical Psychology Review, Volume* 8, Issue 1, Pages 77-100, doi: 10.1016/0272-7358(88)90050-5

Robert A.Steer, David J.Rissmiller, Aaron T.Beck (2000) Use of the Beck Depression Inventory-II with depressed geriatric inpatients *Behaviour Research and Therapy* Volume 38, Issue 3,Pages 311-318, doi: https://doi.org/10.1016/S0005-7967(99)00068-6

Groth-Marnat G. (1990). The handbook of psychological assessment (2nd ed.). New York: John Wiley & Sons.

Usage

```
bdi_compute(
  data,
  cols = matches("bdi_[0-9][0-9]$"),
  max_missing = 0,
  prefix = "bdi_",
  keep_all = TRUE
)

bdi_compute_sum(data, cols = matches("bdi_[0-9][0-9]$"), max_missing = 0)

bdi_factorise(bdi_sum)
```

Arguments

`data`	Data containing BDI data
`cols`	Columns that contain BDI data
`max_missing`	Maximum number of components allowed to be missing. Defaults to "0", and will return `NA` if missing any question. If set to `NULL` any missing component counts as 0, meaning if all BDI components are missing, the sum is still 0, not `NA`.
`prefix`	string to prefix column names of computed values
`keep_all`	logical, append to data.frame
`bdi_sum`	Sum of BDI questions, as summed by bdi_compute_sum

Value

data.frame

Functions

- `bdi_compute_sum()`: Compute the BDI sum based on a data.frame containing the BDI data. Returns a numeric vector.
- `bdi_factorise()`: Create a factor based on the BDI sum, with the cut-off points as described in original paper.

Examples

```
# Example of treatment of missing values
library(dplyr)
library(questionnaires)
data <- tibble(
bdi_01 = c(1, NA_real_, NA_real_, 2, 1),
bdi_02 = c(1, 1, NA_real_, 2, NA_real_)
)

# Row with all components missing, gets sum 0
  bind_cols(data,
```

```
      bdi_sum = bdi_compute_sum(data))

# Do not allow any missing values
  bind_cols(data,
      bdi_sum = bdi_compute_sum(data, max_missing = 0))

# Allow one missing value
  bind_cols(data,
      bdi_sum = bdi_compute_sum(data, max_missing = 2))
```

`bdi_restructure` *Restructure BDI questions from wide format*

Description

If data come from Nettskjema, the structure is in wide format, with each question option as columns, creating 21*4 columns of data. This function allows you to gather and create single columns for questions.

Usage

```
bdi_restructure(data, cols = matches("[0-9]_[0-9]"), sep = "_")
```

Arguments

`data`	Data containing BDI data
`cols`	Columns that contain BDI data
`sep`	separator to use for the column names

Details

The columns must adhere to some specific logic to work. It is recommended that the column names are in the format bdi_01_0 bdi_01_1 bdi_01_2 bdi_01_3, where the first two numbers are the question number, and the last number is the option number.

Value

data frame

Examples

```
dat <- data.frame(
   ID = 1:4,
   bdi_01_0 = c(NA,1, NA, NA),
   bdi_01_1 = c(1, NA, 1, NA),
   bdi_01_2 = c(NA, NA, 1, NA),
   bdi_01_3 = c(NA, NA, NA, NA),
   bdi_02_0 = c(1, NA, NA, NA),
```

```
    bdi_02_1 = c(NA,NA, NA, NA),
    bdi_02_2 = c(NA,1, NA, NA),
    bdi_02_3 = c(NA, NA, NA, 1)
)
bdi_restructure(dat)
```

bfi *Big 5 Inventory*

Description

The BFI-2 is a measure of the Big Five personality domains (which we label Extraversion, Agreeableness, Conscientiousness, Negative Emotionality, and Open-Mindedness) and 15 more-specific facet traits. The Big Five personality traits was the model to comprehend the relationship between personality and academic behaviors. This model was defined by several independent sets of researchers who used factor analysis of verbal descriptors of human behavior. These researchers began by studying relationships between a large number of verbal descriptors related to personality traits. They reduced the lists of these descriptors by 5–10 fold and then used factor analysis to group the remaining traits (using data mostly based upon people's estimations, in self-report questionnaire and peer ratings) in order to find the underlying factors of personality

Item numbers for the BFI-2 domain and facet scales are listed below. Reverse-keyed items are denoted by "R." For more information about the BFI-2, visit the Colby Personality Lab website (http://www.colby.edu/psych/personality-lab/).

Domain Scales:

Extraversion: 1, 6, 11R, 16R, 21, 26R, 31R, 36R, 41, 46, 51R, 56 **Agreeableness:** 2, 7, 12R, 17R, 22R, 27, 32, 37R, 42R, 47R, 52, 57 **Conscientiousness:** 3R, 8R, 13, 18, 23R, 28R, 33, 38, 43, 48R, 53, 58R **Negative Emotionality:** 4R, 9R, 14, 19, 24R, 29R, 34, 39, 44R, 49R, 54, 59 **Open-Mindedness:** 5R, 10, 15, 20, 25R, 30R, 35, 40, 45R, 50R, 55R, 60

Facet Scales:

Sociability: 1, 16R, 31R, 46 **Assertiveness:** 6, 21, 36R, 51R **Energy Level:** 11R, 26R, 41, 56 **Compassion:** 2, 17R, 32, 47R **Respectfulness:** 7, 22R, 37R, 52 **Trust:** 12R, 27, 42R, 57 **Organization:** 3R, 18, 33, 48R **Productiveness:** 8R, 23R, 38, 53 **Responsibility:** 13, 28R, 43, 58R **Anxiety:** 4R, 19, 34, 49R **Depression:** 9R, 24R, 39, 54 **Emotional Volatility:** 14, 29R, 44R, 59 **Intellectual Curiosity:** 10, 25R, 40, 55R **Aesthetic Sensitivity:** 5R, 20, 35, 50R **Creative Imagination:** 15, 30R, 45R, 60

Relational table:

Domain	Factor-pure facet	Complementary facets
E	Sociability	Assertiveness, Energy Level
A	Compassion	Respectfulness, Trust
C	Organization	Productiveness, Responsibility
N	Anxiety	Depression, Emotional Volatility
O	Aesthetic Sensitivity	Intellectual Curiosity, Creative Imagination

Data requirements:

Column names:

The package functions expect the data to be named in a specific way, and to not contain data other than the BFI-2 data. Column names should be zero-leading two digits to indicate the question number, and they should end with these two digits. If this system is followed, then all functions work out of the box.

Examples that work:

- `bfi_01` `bfi_02` ... `bfi_59` `bfi_60`
- `big_five_01` `big_five_02` ... `big_five_59` `bbig_five_60`

Examples that won't work

- `bfi_1` `bfi_2` ... `bfi_59` `bfi_60`
- `big_five_01_trust` `big_five_02_change` ... `big_five_59_test` `bbig_five_60_lat`

Data values:

The data should be coded with the original scoring system 1-5. The data should **not** have implemented necessary reversal of answers for any of the questions, the functions will take care of this.

References:

Soto, C. J., & John, O. P. (2017). The next Big Five Inventory (BFI-2): Developing and assessing a hierarchical model with 15 facets to enhance bandwidth, fidelity, and predictive power. *Journal of Personality and Social Psychology*, **113**(1), 117–143. https://doi.org/10.1037/pspp0000096

Usage

```
bfi_compute(
  data,
  type = c("domains", "facets"),
  keep_all = FALSE,
  prefix = "bfi_"
)

bfi_compute_domains(
  data,
  domains = c("extraversion", "agreeableness", "conscientiousness",
    "negative emotionality", "open-mindedness"),
  keep_all = FALSE,
  prefix = "domain_"
)

bfi_compute_facets(
  data,
 facets = c("sociability", "assertiveness", "energy", "compassion", "respectful",
   "trust", "organization", "productive", "responsibility", "anxiety", "depression",
    "emotional volatility", "intellectual curiosity", "aesthetic sensebility",
    "creative imagination"),
  keep_all = FALSE,
  prefix = "facet_"
)
```

Arguments

data	data.frame containing bfi data
type	Choose domains or facets. Default is both
keep_all	logical, append to data.frame
prefix	string to prefix column names of computed values
domains	string vector of domains to compute
facets	string vector of facets to compute

Value

data.frame with calculated scores

Functions

- `bfi_compute_domains()`: Compute BFI-2 domains and return in a data.frame
- `bfi_compute_facets()`: Compute BFI-2 domains and return in a data.frame

Examples

```
library(dplyr)
# Making some test data
test_data <- tibble(
  id = rep(1:10, each = 60),
  name = rep(sprintf("bfi_%02d", 1:60), 10),
  value = lapply(1:10, function(x){
    sample(1:5, size = 60, replace = TRUE)
  }) %>% unlist()
) %>%
  tidyr::pivot_wider()

bfi_compute(test_data)
bfi_compute(test_data, prefix = "bfi_")
```

---

bfi_domain *BFI-2 Domain computations*

Description

Calculate the domains of the BFI-2

Usage

```
bfi_domain_extravers(
  data,
  cols = matches("01$|06$|11$|16$|21$|26$|31$|36$|41$|46$|51$|56$"),
  reverse = TRUE,
  ...
)

bfi_domain_agreeable(
  data,
  cols = matches("02$|07$|12$|17$|22$|27$|32$|37$|42$|47$|52$|57$"),
  reverse = TRUE,
  ...
)

bfi_domain_conscient(
  data,
  cols = matches("03$|08$|13$|18$|23$|28$|33$|38$|43$|48$|53$|58$"),
  reverse = TRUE,
  ...
)

bfi_domain_negemotion(
  data,
  cols = matches("04$|09$|14$|19$|24$|29$|34$|39$|44$|49$|54$|59$"),
  reverse = TRUE,
  ...
)

bfi_domain_openminded(
  data,
  cols = matches("05$|10$|15$|20$|25$|30$|35$|40$|45$|50$|55$|60$"),
  reverse = TRUE,
  ...
)
```

Arguments

data	data.frame with BFI data in
cols	tidyselector(s) of which columns data are in
reverse	logical. If reversal is needed (default) or not
...	other arguments to bfi_reversal

Value

a vector with computed domain values.

Functions

- `bfi_domain_extravers()`: Calculate the extraversion domain.
- `bfi_domain_agreeable()`: Calculate the agreeableness domain.
- `bfi_domain_conscient()`: Calculate the conscientiousness domain.
- `bfi_domain_negemotion()`: Calculate the negative emotionality domain.
- `bfi_domain_openminded()`: Calculate the open-minded domain.

Examples

```
library(dplyr)
# Making some test data
test_data <- dplyr::tibble(
  id = rep(1:10, each = 60),
  name = rep(sprintf("bfi_%02d", 1:60), 10),
  value = lapply(1:10, function(x){
    sample(1:5, size = 60, replace = TRUE)
  }) %>% unlist()
) %>%
  tidyr::pivot_wider()

bfi_domain_extravers(test_data)
bfi_domain_conscient(test_data)
```

---

bfi_facet *BFI-2 Facet computations*

Description

Calculate the facets of the BFI-2

Usage

```
bfi_facet_sociability(
  data,
  cols = matches("01$|16$|31$|46$"),
  reverse = TRUE,
  ...
)

bfi_facet_assertive(
  data,
  cols = matches("06$|21$|36$|51$"),
  reverse = TRUE,
  ...
)
```

```
bfi_facet_energy(data, cols = matches("11$|26$|41$|56$"), reverse = TRUE, ...)

bfi_facet_compassion(
  data,
  cols = matches("02$|17$|32$|47$"),
  reverse = TRUE,
  ...
)

bfi_facet_respectful(
  data,
  cols = matches("07$|22$|37$|52$"),
  reverse = TRUE,
  ...
)

bfi_facet_trust(data, cols = matches("12$|27$|42$|57$"), reverse = TRUE, ...)

bfi_facet_organization(
  data,
  cols = matches("03$|18$|33$|48$"),
  reverse = TRUE,
  ...
)

bfi_facet_productive(
  data,
  cols = matches("08$|23$|38$|53$"),
  reverse = TRUE,
  ...
)

bfi_facet_responsibility(
  data,
  cols = matches("13$|28$|43$|58$"),
  reverse = TRUE,
  ...
)

bfi_facet_anxiety(data, cols = matches("04$|19$|34$|49$"), reverse = TRUE, ...)

bfi_facet_depression(
  data,
  cols = matches("09$|24$|39$|54$"),
  reverse = TRUE,
  ...
)
```

```
bfi_facet_emovolatility(
  data,
  cols = matches("14$|29$|44$|59$"),
  reverse = TRUE,
  ...
)

bfi_facet_intcuriosity(
  data,
  cols = matches("10$|25$|40$|55$"),
  reverse = TRUE,
  ...
)

bfi_facet_aestheticsens(
  data,
  cols = matches("05$|20$|35$|50$"),
  reverse = TRUE,
  ...
)

bfi_facet_imagination(
  data,
  cols = matches("15$|30$|45$|60$"),
  reverse = TRUE,
  ...
)
```

Arguments

`data`	data.frame with BFI data in
`cols`	tidyselector(s) of which columns data are in
`reverse`	logical. If reversal is needed (default) or not
`...`	other arguments to `bfi_reversal`

Value

a vector with computed domain values.

Functions

- `bfi_facet_sociability()`: Calculate the sociability facet.
- `bfi_facet_assertive()`: Calculate the assertive facet.
- `bfi_facet_energy()`: Calculate the energy level facet.
- `bfi_facet_compassion()`: Calculate the compassion facet.
- `bfi_facet_respectful()`: Calculate the respectfulness facet.
- `bfi_facet_trust()`: Calculate the trust facet.

- bfi_facet_organization(): Calculate the organization facet.
- bfi_facet_productive(): Calculate the productive facet.
- bfi_facet_responsibility(): Calculate the responsibility facet.
- bfi_facet_anxiety(): Calculate the anxiety facet.
- bfi_facet_depression(): Calculate the depression facet.
- bfi_facet_emovolatility(): Calculate the emotional volatiliity facet.
- bfi_facet_intcuriosity(): Calculate the intellectual curiosity facet.
- bfi_facet_aestheticsens(): Calculate the aesthetic sensibility facet.
- bfi_facet_imagination(): Calculate the creative imagination facet.

Examples

```
library(dplyr)
# Making some test data
test_data <- tibble(
  id = rep(1:10, each = 60),
  name = rep(sprintf("bfi_%02d", 1:60), 10),
  value = lapply(1:10, function(x){
    sample(1:5, size = 60, replace = TRUE)
  }) %>% unlist()
) %>%
  tidyr::pivot_wider()

bfi_facet_sociability(test_data)
bfi_facet_assertive(test_data)
```

---

bfi_reversal *Big-5 Item reversals*

Description

Big-5 Item reversals

Usage

```
bfi_reversal(data, ...)
```

Arguments

data	Data with big-5 columns
...	Column selection to reverse

Value

data.frame with specified columns reversed

Examples

```
data <- dplyr::tibble(
 col_01 = c(1:5, 3, 5, 4),
 col_02 = c(1:5, 3, 5, 4)
)
bfi_reversal(data, col_01)
```

---

edu_compile *Compile education across sources*

Description

Compiles education from participant, mother or father depending on source availability. Made for ease of testing and reporting education SES of family

Usage

```
edu_compile(data, participant, mother, father)
```

Arguments

`data`	MOAS-like data.frame
`participant`	unquoted column of 4 category education for participant
`mother`	unquoted column of 4 category education for participant's mother
`father`	unquoted column of 4 category education for participant's father

Value

dataframe with three new columns

See Also

Other edu_functions: `edu_compute()`, `edu_factorise()`, `edu_levels2name()`, `edu_levels()`, `edu_reduce()`, `edu_to_years()`

Examples

```
edu <- data.frame(
    edu4 = c("3", "High school", 1, NA,
         "University/University college (> 4 years)", NA,
          "University/University college (< 4 years)"),
    edu9 = c(7,7,8,NA,"Primary school (6 years)",5, 9),
    edu_years = c(NA, 12, 9, NA, 19, 19, NA),
    mother = c("3", "High school", 1, NA,
               "University/University college (> 4 years)",
               "University/University college (> 4 years)",
               "University/University college (< 4 years)"),
    father = c(7,7,8,4,"Primary school (6 years)",5, 10),
```

```
    stringsAsFactors = FALSE
    )

library(dplyr)
edu %>%
    mutate(
      mother = ifelse(mother == "3", NA, mother),
      mother = edu4_factorise(mother),
      father = edu9_reduce(edu9_factorise(father))
    ) %>%
    edu_compile(
      participant = edu4,
      mother = mother,
      father = father
      )
```

---

edu_compute *Fill inn Education in MOAS*

Description

Using existing data in the MOAS, fills in gaps, converts from on type of coding to another etc.

Usage

```
edu_compute(
  data,
  edu4 = edu_coded4,
  edu9 = edu_coded10,
  edu_years = edu_years,
  prefix = "edu_",
  keep_all = TRUE
)
```

Arguments

data	MOAS-like data
edu4	unquoted column containing Education coded in 4 categories
edu9	unquoted column containing Education coded in 4 categories
edu_years	unquoted column containing Education in years to highest completed
prefix	string to prefix column names of computed values
keep_all	logical, append to data.frame

Value

a data.frame

See Also

Other edu_functions: edu_compile(), edu_factorise(), edu_levels2name(), edu_levels(), edu_reduce(), edu_to_years()

Examples

```
edu <- data.frame(
    edu4 = c("3", "High school", 1, NA,
         "University/University college (> 4 years)", NA,
          "University/University college (< 4 years)"),
    edu9 = c(7,7,8,NA,"Primary school (6 years)",5, 9),
    edu_years = c(NA, 12, 9, NA, 19, 19, NA),
    mother = c("3", "High school", 1, NA,
               "University/University college (> 4 years)",
               "University/University college (> 4 years)",
               "University/University college (< 4 years)"),
    father = c(7,7,8,4,"Primary school (6 years)",5, 10),
    stringsAsFactors = FALSE
    )

 edu_compute(edu,
             edu4 = edu4,
             edu9 = edu9,
             edu_years = edu_years)
```

---

edu_factorise *Create factor from education vector*

Description

Will convert even a mixed character vector (combining numbers and text) of education levels 10 and 4 to a factor.

Usage

```
edu_factorise(x, levels)

edu4_factorise(x)

edu9_factorise(x)
```

Arguments

x	character vector
levels	levels returned from the edu_levels() function

Details

Specialized returns

- edu_factorise - with option to choose number of levels
- edu4_factorise - directly transform vector coded in 4-level scheme
- edu9_factorise - directly transform vector coded in 9-levels scheme

Value

factor

See Also

Other edu_functions: edu_compile(), edu_compute(), edu_levels2name(), edu_levels(), edu_reduce(), edu_to_years()

Examples

```
edu9 <- c("7", "7", "8", NA, "Primary school (6 years)", "5", "9")
edu_factorise(edu9, 9)
edu9_factorise(edu9)
```

---

edu_levels   *Get education levels scheme*

Description

Keeping track of the different educational coding schemes at LCBC can be tricky. This formula contains the two current types of coding schemas employed by LCBC.

Usage

```
edu_levels(levels = 4)

edu4_levels()

edu9_levels()
```

Arguments

levels	how many levels to return (either 4 or 9)

Details

Specialized returns

- edu_levels - returns named numeric vector for levels specified
- edu4_levels - returns named numeric vector for 4-levels scheme
- edu9_levels - returns named numeric vector for 9-levels scheme

Value

named numeric vector

See Also

Other edu_functions: edu_compile(), edu_compute(), edu_factorise(), edu_levels2name(), edu_reduce(), edu_to_years()

Examples

```
edu_levels(4)
edu_levels(9)

edu4_levels()
edu9_levels()
```

---

edu_levels2name *Alter levels to name*

Description

Change educational coded levels to names of the levels

Usage

```
edu_levels2name(x, levels)

edu4_levels2name(x)

edu9_levels2name(x)
```

Arguments

x	vector containing levels
levels	numeric of number of levels (4 or 10)

Details

Specialized returns

- edu_levels2name - transforms levels to names for levels specified
- edu4_levels2name - transforms levels to names for 4-levels scheme
- edu9_levels2name - transforms levels to names for 9-levels scheme

Value

character vector

See Also

Other edu_functions: edu_compile(), edu_compute(), edu_factorise(), edu_levels(), edu_reduce(), edu_to_years()

Examples

```
edu4 <- c(9, 9, 16, 19)
edu_levels2name(edu4, 4)
# does the same as
edu4_levels2name(edu4)

edu9 <- c(0, 6, 21, 16)
edu_levels2name(edu9, 9)
# does the same as
edu9_levels2name(edu9)
```

---

edu_map *Create a mapped table for conversion*

Description

Converting from a high-level educational coding to a lower level one is cumbersome. This function bases it self in any coding scheme specified in edu_levels and tries creating a conversion table between two specified schemas.

Usage

```
edu_map(from = 9, to = 4)

edu_map_chr(from = 9, to = 4)

edu_map_num(from = 9, to = 4)
```

Arguments

from	schema levels to convert from
to	shcema levels to convtert to

Details

Specialized returns

- edu_map - returns a data.frame of two named vectors
- edu_map_chr - returns a data.frame with two character vectors
- edu_map_num - returns a data.frame with two numeric vectors

edu_recode *Recode new 9 levels into old*

Description

New nettskjema data requires codebook to not have special characters, and as such the old and new coding scheme does not fit. This function turns new coding scheme into the old, wanted one

Usage

```
edu_recode(x, names = TRUE)
```

Arguments

x	character vector of old scheme
names	logical. toggle return of names rather than numbers

Value

character

Examples

```
eds <- c(NA, "UnderGrad_BA", "HighSchool_Initial", "PostGrad_MA",
   "PostGrad_PhD", "HighSchool", "Junior-HighSchool", "HighSchool_addition")
edu_recode(eds)

eds <- c(1,5,8,2,6,9,1,10)
edu_recode(eds, names = FALSE)
```

edu_reduce *Reduce education categories*

Description

These functions will aid in converting one education scheme into another. While you may attempt to go from a low level to a high (from 4 to 9), there is no way to actually do that in a consistent way that will correctly reflect the underlying data.

Usage

```
edu_reduce(x, from, to)

edu9_reduce(x, to = 4)
```

Arguments

x	character vector
from	factor level to transform from
to	factor level to transform to

Details

Always go from a higher level scheme to a lower one (currently from 9 to 4 only)

Specialized returns

- edu_reduce - reduce with own to and from specification
- edu9_reduce - directly reduce from 9 to 4

Value

factor

See Also

Other edu_functions: `edu_compile()`, `edu_compute()`, `edu_factorise()`, `edu_levels2name()`, `edu_levels()`, `edu_to_years()`

Examples

```
edu9 <- c("7", "7", "8", NA, "Primary school (6 years)", "5", "9")
edu_reduce(edu9, 9, 4)
edu9_reduce(edu9)
```

---

edu_to_years *Turn education data to years*

Description

Turn education data to years

Usage

```
edu_to_years(x, levels)

edu4_to_years(x)

edu9_to_years(x)
```

Arguments

x	character vector
levels	levels returned from the edu_levels() function

Details

Specialized returns

- edu_to_years - Alter education to years specifying number of levels
- edu4_to_years - directly alter 4-level coded education to years
- edu9_to_years - directly alter 9-level coded education to years

Value

vector of integers

See Also

Other edu_functions: `edu_compile()`, `edu_compute()`, `edu_factorise()`, `edu_levels2name()`, `edu_levels()`, `edu_reduce()`

Examples

```
edu4 <- c("3", "High school", "1", NA,
          "University/University college (> 4 years)",
           NA, "University/University college (< 4 years)")

edu_to_years(edu4, 4)
edu4_to_years(edu4)

edu9 <- c("7", "7", "8", NA, "Primary school (6 years)", "5", "9")
edu_to_years(edu9, 9)
edu9_to_years(edu9)
```

---

ehi_change — *Create vector with only correct values*

Description

Since the coding we have often uses negative numbers to indicate left-hand preferences, a specialized function is here to return a vector with only the values asked for.

Usage

```
ehi_change(x, direction = 1)
```

Arguments

x	numeric vector
direction	either 1 for positive, -1 for negative

Details

If direction is set to 1, returns only positive numbers, negative and 0 returns as NA. If direction is set to -1, returns only negative numbers, positive and 0 returns as NA.

Value

numeric vector

`ehi_compute` *Edinburgh handedness inventory*

Description

Compute all variables of ehi, using other functions in this package. Will return the given data.frame with three additional columns, the laterality quotient (LQ), the laterality factor (Coded), and the nominal laterality code (Nominal).

Usage

```
ehi_compute(
  data,
  cols = matches("^ehi_[0-9][0-9]$"),
  writing = ehi_01,
  ...,
  keep_all = TRUE,
  prefix = "ehi_"
)
```

Arguments

`data`	data.frame containing ehi data
`cols`	tidyselected columns of all ehi data
`writing`	numeric vector of writing preference (-2,-1,0,1,2)
`...`	additional arguments to ehi_factorise_lqa
`keep_all`	logical, append to data.frame
`prefix`	string to prefix column names of computed values

Details

Background:

The Edinburgh Handedness Inventory is a measurement scale used to assess the dominance of a person's right or left hand in everyday activities, sometimes referred to as laterality. The inventory can be used by an observer assessing the person, or by a person self-reporting hand use. The latter method tends to be less reliable due to a person over-attributing tasks to the dominant hand.

Scoring:

The EHI has several measures that can help assess a person's laterality.

answer	value	nominal	lq	lq_cat	lqa_cat
Left dominance	-2	left	-100	left	left
Left preference	-1	left	-40	left	ambidexter
No preference	0	ambidexter	0	right	ambidexter
Right preference	1	right	40	right	ambidexter
Right dominance	2	right	100	right	right

Nominal:

The easiest measure from the EHI is the nominal laterality value, which is just the answer to the first question on hand preference when writing. This simple index just treat negative answers as "left" dominance, positive number as "right" dominance, and a 0 as ambidextrous. **Note:** The original paper by Oldfield (1971) does not explicitly state a category for "Ambidextrous". It is very rare that a person does not have a clear preference on writing hand, even if they *can* write with both hands. This category is only added in this package to handle the possible case of someone answering "No preference".

min	max	category
-2	-1	left
0	0	ambidexter
1	2	right

Laterality quotient (lq):

The total score of the EHI is more than just summing the values for each answer. The laterality quotient (LQ) uses the answers to all the questions. The LQ can take values from -100 to 100, and is calculated by taking the sum of all positive answers subtracting the sum of absolute values of the negative answers, divided by the sum of both, and multiplied by 100.

`katex::katex_html(equation)`

Laterality index:

The laterality index is based on the laterality quotient (above) and categorises answers into to categories, Left and Right. The Oldfield (1971) paper mentions "indeterminate handedness" a couple of times in the paper, but the case for "true" ambidextrous is not made, and as such the inventory does not have official categories for that. As the index is based on the quotient, that ranges from -100 to 100, getting a perfect `0` LQ is very unlikely, and as indicated in the paper, such score is assumed to belong to the Right hand part of the scale.

min	max	category
-100	-1	left
0	100	right

An alternate laterality index is also often employed, where scores between -40 and 40 are treated as ambidextrous.

Data requirements:

One row of data should refer to a single questionnaire answered, and as such, if a person has answered multiple times, these should appear on separate rows with columns identifying ID and time point per observation.

Column names:
For ease, we recommend naming the columns in a consistent way, so the functions in this package become easier to use. The LCBC database follows a naming scheme that prefixes all columns with `ehi_` and ends with a zero-padded double digit indicator of the question number.

Data values:
The cell values in the data should be coded from -2 through 0 to 2, and there should be a single value per question.

value	category
-2	Left hand dominance
-1	Left hand preference
0	No preference
1	Right hand preference
2	Right hand dominance

References:

Oldfield, RC (March 1971) *The assessment and analysis of handedness: The Edinburgh inventory*. Neuropsychologia. 9 (1): 97–113. doi:10.1016/0028-3932(71)90067-4

Verdino, M; Dingman, S (April 1998). *Two measures of laterality in handedness: the Edinburgh Handedness Inventory and the Purdue Pegboard test of manual dexterity*. Perceptual and Motor Skills. 86 (2): 476–8. doi:10.2466/pms.1998.86.2.476

Knecht, S; Dräger, B; Deppe, M; Bobe, L; Lohmann, H; Flöel, A; Ringelstein, E-B; Henningsen, H (December 2000). *Handedness and hemispheric language dominance in healthy humans*. Brain. 123 (12): 2512–8. doi:10.1093/brain/123.12.2512.

Value

data.frame

See Also

Other ehi_functions: `ehi_compute_lq()`, `ehi_factorise_lq()`, `ehi_factorise_nominal()`

`ehi_compute_lq` *Laterality Quotient*

Description

The laterality quotient is calculated using all the answers on the ehi, with the formula: (pos-neg)/(pos+neg)*100)

Background:

The Edinburgh Handedness Inventory is a measurement scale used to assess the dominance of a person's right or left hand in everyday activities, sometimes referred to as laterality. The inventory can be used by an observer assessing the person, or by a person self-reporting hand use. The latter method tends to be less reliable due to a person over-attributing tasks to the dominant hand.

Scoring:

The EHI has several measures that can help assess a person's laterality.

answer	value	nominal	lq	lq_cat	lqa_cat
Left dominance	-2	left	-100	left	left
Left preference	-1	left	-40	left	ambidexter
No preference	0	ambidexter	0	right	ambidexter
Right preference	1	right	40	right	ambidexter
Right dominance	2	right	100	right	right

Nominal:

The easiest measure from the EHI is the nominal laterality value, which is just the answer to the first question on hand preference when writing. This simple index just treat negative answers as "left" dominance, positive number as "right" dominance, and a 0 as ambidextrous. **Note:** The original paper by Oldfield (1971) does not explicitly state a category for "Ambidextrous". It is very rare that a person does not have a clear preference on writing hand, even if they *can* write with both hands. This category is only added in this package to handle the possible case of someone answering "No preference".

min	max	category
-2	-1	left
0	0	ambidexter
1	2	right

Laterality quotient (lq):

The total score of the EHI is more than just summing the values for each answer. The laterality quotient (LQ) uses the answers to all the questions. The LQ can take values from -100 to 100, and is calculated by taking the sum of all positive answers subtracting the sum of absolute values of the negative answers, divided by the sum of both, and multiplied by 100.

`katex::katex_html(equation)`

Laterality index:

The laterality index is based on the laterality quotient (above) and categorises answers into to categories, Left and Right. The Oldfield (1971) paper mentions "indeterminate handedness" a couple of times in the paper, but the case for "true" ambidextrous is not made, and as such the inventory does not have official categories for that. As the index is based on the quotient, that ranges from -100 to 100, getting a perfect 0 LQ is very unlikely, and as indicated in the paper, such score is assumed to belong to the Right hand part of the scale.

min	max	category
-100	-1	left
0	100	right

An alternate laterality index is also often employed, where scores between -40 and 40 are treated as ambidextrous.

Data requirements:

One row of data should refer to a single questionnaire answered, and as such, if a person has answered multiple times, these should appear on separate rows with columns identifying ID and time point per observation.

Column names:

For ease, we recommend naming the columns in a consistent way, so the functions in this package become easier to use. The LCBC database follows a naming scheme that prefixes all columns with `ehi_` and ends with a zero-padded double digit indicator of the question number.

Data values:

The cell values in the data should be coded from -2 through 0 to 2, and there should be a single value per question.

value	category
-2	Left hand dominance
-1	Left hand preference
0	No preference
1	Right hand preference
2	Right hand dominance

References:

Oldfield, RC (March 1971) *The assessment and analysis of handedness: The Edinburgh inventory*. Neuropsychologia. 9 (1): 97–113. doi:10.1016/0028-3932(71)90067-4

Verdino, M; Dingman, S (April 1998). *Two measures of laterality in handedness: the Edinburgh Handedness Inventory and the Purdue Pegboard test of manual dexterity*. Perceptual and Motor Skills. 86 (2): 476–8. doi:10.2466/pms.1998.86.2.476

Knecht, S; Dräger, B; Deppe, M; Bobe, L; Lohmann, H; Flöel, A; Ringelstein, E-B; Henningsen, H (December 2000). *Handedness and hemispheric language dominance in healthy humans*. Brain. 123 (12): 2512–8. doi:10.1093/brain/123.12.2512.

Usage

```
ehi_compute_lq(data, cols = matches("^ehi_[0-9][0-9]$"))
```

Arguments

`data`	data.frame containing ehi data
`cols`	tidyselected columns of all ehi data

Value

numeric

See Also

Other ehi_functions: ehi_compute(), ehi_factorise_lq(), ehi_factorise_nominal()

---

ehi_factorise_lq *Factorise laterality quotient*

Description

While the laterality quotient is nice to use if your sample and variance is large enough for analyses, in most cases you will need to report the categories of laterality your participants fall within. This function takes the laterality quotient as computed by ehi_compute_lq and creates a factor using common specifications.

Usage

```
ehi_factorise_lq(lq = ehi_lq)

ehi_factorise_lqa(
  lq,
  min = -70,
  max = 70,
  levels = c("left", "ambidexter", "right")
)
```

Arguments

`lq`	numeric vector calculated by ehi_compute_lq
`min`	minimum value for ambidexter specification (default = -70)
`max`	maximum value for ambidexter specification (default = 70)
`levels`	the levels for the lq component. Usually c("left", "ambidexter", "right").

Details

Background:

The Edinburgh Handedness Inventory is a measurement scale used to assess the dominance of a person's right or left hand in everyday activities, sometimes referred to as laterality. The inventory can be used by an observer assessing the person, or by a person self-reporting hand use. The latter method tends to be less reliable due to a person over-attributing tasks to the dominant hand.

Scoring:

The EHI has several measures that can help assess a person's laterality.

answer	value	nominal	lq	lq_cat	lqa_cat
Left dominance	-2	left	-100	left	left
Left preference	-1	left	-40	left	ambidexter
No preference	0	ambidexter	0	right	ambidexter
Right preference	1	right	40	right	ambidexter
Right dominance	2	right	100	right	right

Nominal:

The easiest measure from the EHI is the nominal laterality value, which is just the answer to the first question on hand preference when writing. This simple index just treat negative answers as "left" dominance, positive number as "right" dominance, and a 0 as ambidextrous. **Note:** The original paper by Oldfield (1971) does not explicitly state a category for "Ambidextrous". It is very rare that a person does not have a clear preference on writing hand, even if they *can* write with both hands. This category is only added in this package to handle the possible case of someone answering "No preference".

min	max	category
-2	-1	left
0	0	ambidexter
1	2	right

Laterality quotient (lq):

The total score of the EHI is more than just summing the values for each answer. The laterality quotient (LQ) uses the answers to all the questions. The LQ can take values from -100 to 100, and is calculated by taking the sum of all positive answers subtracting the sum of absolute values of the negative answers, divided by the sum of both, and multiplied by 100.

`katex::katex_html(equation)`

Laterality index:

The laterality index is based on the laterality quotient (above) and categorises answers into to categories, Left and Right. The Oldfield (1971) paper mentions "indeterminate handedness" a couple of times in the paper, but the case for "true" ambidextrous is not made, and as such the inventory does not have official categories for that. As the index is based on the quotient, that ranges from -100 to 100, getting a perfect `0` LQ is very unlikely, and as indicated in the paper, such score is assumed to belong to the Right hand part of the scale.

min	max	category
-100	-1	left
0	100	right

An alternate laterality index is also often employed, where scores between -40 and 40 are treated as ambidextrous.

Data requirements:

One row of data should refer to a single questionnaire answered, and as such, if a person has answered multiple times, these should appear on separate rows with columns identifying ID and time point per observation.

Column names:

For ease, we recommend naming the columns in a consistent way, so the functions in this package become easier to use. The LCBC database follows a naming scheme that prefixes all columns with `ehi_` and ends with a zero-padded double digit indicator of the question number.

Data values:

The cell values in the data should be coded from -2 through 0 to 2, and there should be a single value per question.

value	category
-2	Left hand dominance
-1	Left hand preference
0	No preference
1	Right hand preference
2	Right hand dominance

References:

Oldfield, RC (March 1971) *The assessment and analysis of handedness: The Edinburgh inventory*. Neuropsychologia. 9 (1): 97–113. doi:10.1016/0028-3932(71)90067-4

Verdino, M; Dingman, S (April 1998). *Two measures of laterality in handedness: the Edinburgh Handedness Inventory and the Purdue Pegboard test of manual dexterity*. Perceptual and Motor Skills. 86 (2): 476–8. doi:10.2466/pms.1998.86.2.476

Knecht, S; Dräger, B; Deppe, M; Bobe, L; Lohmann, H; Flöel, A; Ringelstein, E-B; Henningsen, H (December 2000). *Handedness and hemispheric language dominance in healthy humans*. Brain. 123 (12): 2512–8. doi:10.1093/brain/123.12.2512.

- ehi_factorise_lq - returns original two-factor specification
- ehi_factorise_lqa - returns commonly used three-factor specification

Value

factor

See Also

Other ehi_functions: `ehi_compute_lq()`, `ehi_compute()`, `ehi_factorise_nominal()`

Examples

```
LQ <- c(1, 40, 70, -20, 0, 100, -90)
ehi_factorise_lq(LQ)
ehi_factorise_lqa(LQ)
ehi_factorise_lqa(LQ, min = -40, max = 60)
```

ehi_factorise_nominal *Nominal laterality factor*

Description

Using the answers to the first question on writing from the Edinburgh handedness inventory, a nominal scale of three factors can be returned.

Usage

```
ehi_factorise_nominal(writing = ehi_01)
```

Arguments

`writing`	numeric vector of writing preference (-2,-1,0,1,2)

Value

factor

See Also

Other ehi_functions: `ehi_compute_lq()`, `ehi_compute()`, `ehi_factorise_lq()`

Examples

```
writing <- c(2, 2, -1, 0, 1, -2)
ehi_factorise_nominal(writing)
```

ehi_values *Sum ehi columns*

Description

Calculate the sum on non-NA values in all columns in the specified direction(1 == sum all positives, -1 sum absolutes values of negatives)

Usage

```
ehi_values(data, cols = matches("^ehi_[0-9][0-9]$"), direction = 1)
```

Arguments

`data`	data.frame containing ehi data
`cols`	tidy-selection of all ehi columns
`direction`	sum positive or negatives (1 for positive, -1 for negative)

Value

numeric vector

`gds_alter_values` *Change coding of GDS to correct numeric values*

Description

Necessary step for computing the total score

Usage

```
gds_alter_values(
  data,
  values = gds_values(),
  reverse = FALSE,
  cols = matches("01$|05$|07$|09$|15$|19$|21$|27$|29$|30$")
)
```

Arguments

`data`	data.frame with GDS data in it
`values`	named vector of 2 providing the coding for Yes and No answers c(Yes = 1, No = 2)
`reverse`	reverse logic
`cols`	GDS data columns

See Also

Other gds_functions: `gds_binary()`, `gds_compute_sum()`, `gds_values()`

`gds_binary` *Binarise GDS values*

Description

internal function to make all "yes" answers equal to 1, and all "no" to 0. This for convenience of calculations later.

Usage

```
gds_binary(x, values = gds_values())
```

Arguments

`x`	vector of yes and no coding
`values`	named vector of 2 providing the coding for Yes and No answers c(Yes = 1, No = 2)

Value

vector of 0's and 1's

See Also

Other gds_functions: `gds_alter_values()`, `gds_compute_sum()`, `gds_values()`

Examples

```
gds_binary(c(1,1,0,NA,1), gds_values(1,0))
gds_binary(c("y","y","n",NA,"y"), gds_values(yes = "y", no = "n"))
```

`gds_compute_sum` *Compute the GDS sum*

Description

The Geriatric Depression Scale (GDS) is an instrument designed specifically for rating depression in the elderly. It can be administrated to healthy, medically ill, and mild to moderately cognitively impaired older adults. As a general rule, GDS is administrated in LCBC to older adults with a lower cut off around 60 years. However, please consult the instructions for each project, as this guideline has been implemented at different time points across the projects.

The questionnaire consists of 30 questions tapping into a wide variety of topics relevant to depression, including cognitive complaints, motivation, thoughts about the past and the future, self-image, and mood itself. The answers should be based the participants' feelings throughout the last week.

Twenty of the questions indicate the presence of depression when answered positively, while the ten remaining indicate depression when answered negatively (see scoring instructions below). The questionnaire is scored accordingly, giving one point for each statement that affirms a depressive symptom. The sum of these scores yields one total score, with a possible range between 0 and 30. ##Scoring The GDS is quite straight forward in its format, a series of 30 questions that take a yes or no answer. This binary coding makes it quite easy to work with. Several of the questions, however, are formulated in such a way that they require a reversal of the coding before the total score can be summed. The questions which require reversal of coding are, **01, 05, 07, 09, 15, 19, 21, 27, 29, 30**, meaning answering "yes" to these should be altered to 0, and "no" altered to 1, before calculating the sum score. The total GDS score is after reversal, a simple addition of all the answers into a single score.

One point is given for any “No” answered to the following questions: 1, 5, 7, 9, 15, 19, 21, 27, 29 and 30

and one point is given for every “Yes” answered on the following questions: 2, 3, 4, 6, 8, 10, 11, 12, 13, 14, 16, 17, 18, 20, 22, 23, 24, 25, 26, 28

Depression categories:

There are 3 categories of severity for the GDS total score. Below or equal to 9 is "Normal", above 19 is "Severe depression", and the remaining fall within "Mild depression".

GDS score	Depression category
0-9	Normal
10-19	Mild depressive
20-30	Severe depressive

Data requirements:

Column naming:
The easiest is to have data coded as in the NOAS, as this will let you use default values for the arguments. The column names in the NOAS all start with `gds_` and then are followed by a two-digit numbering of the question:
gds_01, gds_02, gds_03, ... gds_28, gds_29, gds_30
If your data is coded differently, a consistent naming scheme should help you use the functions anyway.

Data values:
Each row of data should belong to a single answer to the entire questionnaire. Meaning if you have multiple answers to the questionnaire over time, these should be placed in another row, duplicating the participant ID, together with a column indicating the timepoint the data was collected in. Data values are binary yes and no answers to the GDS. While the functions are made in such a way that any type of binary coding works well, the default is set to be `yes = 1`, `no = 0`. These can be altered by applying the `gds_values` functions to the other functions asking for the coding schema.

References:

Depression Screening Scale: A Preliminary Report, *J Psychiatr Res*, 17 (1), 37-49, doi: 10.1016/0022-3956(82)90033-4

E L Lesher 1, J S Berryhill (1994), Validation of the Geriatric Depression Scale – Short Form Among Inpatients, *J Clin Psychol*, 50 (2), 256-60, doi: 10.1002/1097-4679(199403)50:2<256::aid-jclp2270500218>3.0.co;2-e

Usage

```
gds_compute_sum(
  data,
  cols = dplyr::matches("[0-3][0-9]$"),
  cols_rev = dplyr::matches("01$|05$|07$|09$|15$|19$|21$|27$|29$|30$"),
  values = gds_values()
)

gds_factorise(gds_sum)

gds_compute(
  data,
  cols = dplyr::matches("[0-9][0-9]$"),
```

```
  cols_rev = dplyr::matches("01$|05$|07$|09$|15$|19$|21$|27$|29$|30$"),
  values = gds_values(),
  prefix = "gds_",
  keep_all = TRUE
)
```

Arguments

`data`	data.frame with GDS data in it
`cols`	GDS data columns
`cols_rev`	Columns for reversal of binary code
`values`	named vector of 2 providing the coding for Yes and No answers c(Yes = 1, No = 2)
`gds_sum`	numeric vector of GDS sums
`prefix`	string to prefix column names of computed values
`keep_all`	logical, append to data.frame

Value

numeric

factor

data frame

Functions

- `gds_compute_sum()`: Calculate the total GDS score
- `gds_factorise()`: Create a factor from the sum of the GDS scores

See Also

Other gds_functions: `gds_alter_values()`, `gds_binary()`, `gds_values()`

Other gds_functions: `gds_alter_values()`, `gds_binary()`, `gds_values()`

Other gds_functions: `gds_alter_values()`, `gds_binary()`, `gds_values()`

`gds_values` *Specify coding scheme for GDS questions*

Description

Function to easily set the response coding used in the GDS data.

Usage

```
gds_values(yes = 1, no = 0)
```

Arguments

yes	value indicating a positive answer
no	value indicating a negative answer

Value

list of yes and no values

See Also

Other gds_functions: gds_alter_values(), gds_binary(), gds_compute_sum()

Examples

```
gds_values()
gds_values(yes = "YES", no = "NO")
```

---

income_bin2nok *Turn income bins to mean of bin*

Description

Older collected income data for LCBC collected income information in 7 bins. Newer data collects continuous income data. This function converts binned income data from these 7 categories into the mean income value for each bin.

Usage

```
income_bin2nok(x)
```

Arguments

x	income bin vector

Value

numeric

Examples

```
x <- c("< 200k", "600k - 699k", "400k - 499k",
       "> 700k", "500k - 599k", "300k - 399k",
        "200k - 299k")
income_bin2nok(x)
```

income_nok2other *Translate NOK to other currency*

Description

In order to compare income in Norway to other countries, currency conversions might be necessary. This function multiplies with the rate provided.

Usage

```
income_nok2other(x, rate = 0.1)
```

Arguments

x	currency
rate	currency translation rate (defaul 0.10 for euro)

Value

numeric

Examples

```
income_nok2other(c(100, 2930, 13649))
income_nok2other(c(100, 2930, 13649), 0.5)
```

ipaq_compute_met *Compute met from IPAQ*

Description

The purpose of the International Physical Activity questionnaires (IPAQ) is to provide a set of well-developed instruments that can be used internationally to obtain comparable estimates of physical activity. There are two versions of the questionnaire. The short version is suitable for use in national and regional surveillance systems and the long version provide more detailed information often required in research work or for evaluation purposes.

Scoring:

Scoring of the IPAQ is based on a metric called METs, which are multiples of the resting metabolic rate. The IPAQ scoring description can be found here

Continuous Score:
Expressed as MET-min per week: MET level x minutes of activity/day x days per week
MET levels:

- **Light** - 3.3 METs
- **Moderate** - 4.0 METs

- **Vigorous** - 8.0 METs

Total MET-minutes/week = Light (3.3 x min x days) + Mod (4.0 x min x days) + Vig (8.0 x min x days)

Categorical Score:
Three levels (categories) of physical activity are proposed:

Category 1: Low:
This is the lowest level of physical activity. Those individuals who not meet criteria for categories 2 or 3 are considered low/inactive.

Category 2: Moderate:
Any one of the following 3 criteria:

- 3 or more days of vigorous activity of at least 20 minutes per day **OR**
- 5 or more days of moderate-intensity activity or walking of at least 30 minutes per day **OR**
- 5 or more days of any combination of walking, moderate-intensity or vigorous intensity activities achieving a minimum of at least 600 MET-min/week.

Category 3: High:
Any one of the following 2 criteria:

- Vigorous-intensity activity on at least 3 days and accumulating at least 1500 MET-minutes/week **OR**
- 7 or more days of any combination of walking, moderate-intensity or vigorous intensity activities achieving a minimum of at least 3000 MET-minutes/week

References:

Depression Screening Scale: A Preliminary Report, *J Psychiatr Res*, 17 (1), 37-49, doi: 10.1016/0022-3956(82)90033-4

E L Lesher 1, J S Berryhill (1994), Validation of the Geriatric Depression Scale – Short Form Among Inpatients, *J Clin Psychol*, 50 (2), 256-60, doi: 10.1002/1097-4679(199403)50:2<256::aid-jclp2270500218>3.0.co;2-e

Usage

```
ipaq_compute_met(minutes = ipaq_2, days = ipaq_1b, met = ipaq_mets()$light)

ipaq_compute_sum(vigorous, moderate, light)

ipaq_compute(
  data,
  mets = ipaq_mets(),
  light_days = ipaq_5b,
  light_mins = ipaq_6,
  mod_days = ipaq_3b,
  mod_mins = ipaq_4,
  vig_days = ipaq_1b,
  vig_mins = ipaq_2,
  prefix = "ipaq_",
  keep_all = TRUE
)
```

Arguments

minutes	vector of numeric minutes
days	vector of numeric days
met	met number (light = 3.3, moderate = 4.0, vigorous = 8)
vigorous	Vector with vigorous met calculated
moderate	Vector with moderate met calculated
light	Vector with light met calculated
data	data.frame containing all the ipaq data
mets	list generated with ipaq_mets() (default = ipaq_mets())
light_days	column with the days of light activity
light_mins	column with the minutes of light activity
mod_days	column with the days of moderate activity
mod_mins	column with the minutes of moderate activity
vig_days	column with the days of vigorous activity
vig_mins	column with the minutes of vigorous activity
prefix	string to prefix column names of computed values
keep_all	logical, append to data.frame

Value

data.frame

Functions

- ipaq_compute_met(): Calculate mets of an activity type
- ipaq_compute_sum(): Calculate the IPAQ sum based on activities and mets

See Also

Other ipaq_functions: ipaq_mets(), ipaq_time_alter()

Examples

```
ipaq_vig_mins <- c(60, 20, 60, 25, 90, 20, 0, 75, 60, 30)
ipaq_vig_days <- c(1, 3, 2, 5, 6, 1, 1, 2, 2, 4)
ipaq_compute_met(ipaq_vig_mins, ipaq_vig_days, met = 8.0)
light = c(1300, 300)
moderate = c(200, 400)
vigorous = c(0, 1300)
ipaq_compute_sum(vigorous , moderate, light)
```

ipaq_mets *IPAQ mets*

Description

IPAQ calculations require specification of met (resting metabolic rate), which are not necessarily static values. While there are defaults for each of the three categories, there should be the possibility to alter these with newer research.

Usage

```
ipaq_mets(light = 3.3, moderate = 4, vigorous = 8)
```

Arguments

light	numeric. default 3.3
moderate	numeric. default 4.0
vigorous	numeric. default 8.0

Details

This is a convenience function if users need to alter the default values for one or more of the categories and is compatible with the remaining IPAQ functions in this package.

Value

list of three

See Also

Other ipaq_functions: ipaq_compute_met(), ipaq_time_alter()

Examples

```
ipaq_mets()
ipaq_mets(moderate = 5.1)
```

ipaq_time_alter *Alter the time instant columns to decminals*

Description

Time is often punched as HH:MM in order to preserve correct time calculations. The ipaq calculation recure time to be in decimal minutes. This function easily changes HH:MM into decminal minutes in a data.frame It alters columns directly in the data.frame

Usage

```
ipaq_time_alter(data, cols = c(ipaq_2, ipaq_4, ipaq_6, ipaq_7))
```

Arguments

data	data with columns to alter
cols	columns to alter, in tidyselect format

Value

data.frame

See Also

Other ipaq_functions: ipaq_compute_met(), ipaq_mets()

Examples

```
dat <- data.frame(
   time_1  = c("12:34", "09:33", "22:14"),
   time_2  = c("10:55", "16:45", "18:02")
)
ipaq_time_alter(dat, cols = c(time_1, time_2))
```

is_hm *Utility function to locate hm columns*

Description

is_hm locates columns that are time (hm) classes

Usage

```
is_hm(x)
```

Arguments

x	vector

Value

logical vector of length==ncol(data)

Examples

```
## Not run:
is_hm(data)

## End(Not run)
```

---

is_hms *Utility function to locate hms columns*

Description

`is_hms` locates columns that are time (hms) classes

Usage

```
is_hms(x)
```

Arguments

x	vector

Value

logical vector of length==ncol(data)

Examples

```
## Not run:
is_hms(data)

## End(Not run)
```

`psqi_compute_comp2` *Compute all PSQI components and global score*

Description

Despite the prevalence of sleep complaints among psychiatric patients, few questionnaires have been specifically designed to measure sleep quality in clinical populations. The Pittsburgh Sleep Quality Index (PSQI) is a self-rated questionnaire which assesses sleep quality and disturbances over a 1-month time interval. Nineteen individual items generate seven "component" scores: subjective sleep quality, sleep latency, sleep duration, habitual sleep efficiency, sleep disturbances, use of sleeping medication, and daytime dysfunction. The sum of scores for these seven components yields one global score. ##Scoring

component	name	description
1	subjective sleep quality	Answer to q 6
2	sleep latency	Scaled sum of number of minutes before sleep (q 2) and evaluation of sleep withi
3	sleep duration	Scaled score of number of hours before one falls asleep (q 4), scaled to a 5 point s
4	habitual sleep efficiency	hours of sleep (q 4) divided by bedtime (q 1) subtracted from rising time (q 3), an
5	sleep disturbances	Sum of evaluation of sleep within 30min (q 5a) and all remaining questions on sle
6	use of sleeping medication	Answer to question on use of sleep medication (q 7)
7	daytime dysfunction	Sum of evaluation of staying awake (q 8) and evaluation of keeping enthusiastic (
global score	sum of the above.	If any of the above is not possible to calculate, the global sum is also not calculate

Data requirements:

Column names:
Questions with multiple subquestions should be named in a similar manner, suffixed by the alphabetical index (**psqi_5a**, **psqi_5b** etc.). For questions 5j and 10j, the frequency of occurence should have the names **psqi_5j** and **psqi_10e**, and the freehand explanations should have any type of **suffix** after this to indicate a text answers (i.e. **psqi_5j_Desc** or **psqi_5j_string**, **psqi_5j_freehand**). As an example, LCBC has the following set-up:

- **psqi_1**
- **psqi_2**
- **psqi_3**
- **psqi_4**
- **psqi_5a psqi_5b psqi_5c psqi_5d psqi_5e psqi_5f psqi_5g psqi_5h psqi_5i psqi_5j psqi_5j_Coded psqi_5j_Desc**
- **psqi_6**
- **psqi_7**
- **psqi_8**
- **psqi_9**
- **psqi_10 psqi_10a psqi_10b psqi_10c psqi_10d psqi_10e psqi_10e_Desc psqi_10e_Coded**
- **psqi_11a psqi_11b psqi_11c psqi_11d**

4-option questions coding:
All 4-option questions need to be coded **0-3**, not **1-4**.

Time formats:
For question 1, 3 and 4 (bedtime, rising time, hours of sleep), data should be punched as "**HH:MM**". Question 2 should be punched as minutes in numbers.

References:

Buysse et al. (1989) The Pittsburgh sleep quality index: A new instrument for psychiatric practice and research, *Psychiatry Research*, 28:2, 193-213

Usage

```
psqi_compute_comp2(min_before_sleep, no_sleep_30min)

psqi_compute_comp3(hours_sleep)

psqi_compute_comp4(hours_sleep, bedtime, risingtime, ...)

psqi_compute_comp5(data, sleep_troubles = matches("^psqi_05[b-j]$"))

psqi_compute_comp7(keep_awake, keep_enthused)

psqi_compute_global(data, cols = matches("comp[1-7]+_"), max_missing = 0)

psqi_compute(
  data,
  components = 1:7,
  bedtime = psqi_01,
  min_before_sleep = psqi_02,
  risingtime = psqi_03,
  hours_sleep = psqi_04,
  no_sleep_30min = psqi_05a,
  sleepquality = psqi_06,
  medication = psqi_07,
  keep_awake = psqi_08,
  keep_enthused = psqi_09,
  sleep_troubles = matches("^psqi_05[b-j]$"),
  max_missing = 0,
  ...,
  prefix = "psqi_",
  keep_all = TRUE
)
```

Arguments

`min_before_sleep`	column name with no. minutes before sleep (numeric) (psqi_02)
`no_sleep_30min`	column name with evaluation of sleep within 30min (0-3) (psqi_05a)
`hours_sleep`	column name with hours of sleep (decimal hours) (psqi_04)
`bedtime`	column name with bedtime (HH:MM:SS) (psqi_01)

risingtime	column name with rising time (HH:MM:SS) (psqi_03)
...	other arguments to psqi_compute_time_in_bed
data	data frame
sleep_troubles	columns containing sleep problem evaluations (0-3) (psqi_05(b-j))
keep_awake	column name with evaluation of staying awake (0-3) (psqi_08)
keep_enthused	column name with evaluation of keeping enthusiastic (0-3) (psqi_09)
cols	columns containing the components
max_missing	Integer specifying the number of missing values to accept in the PSQI components, before the global PSQI value is set to missing. Defaults to 0. If max_missing > 0, the global PSQI value is computed by weighting each non-missing entry with 7 / (7 - max_missing).
components	integer vector of components to calculate. If all 7, global is added also
sleepquality	column name with evaluation of sleep quality (0-3) (psqi_06)
medication	column name with use of sleep mediation (0-3) (psqi_07)
prefix	string to prefix column names of computed values
keep_all	logical, append to data.frame

Value

a data.frame containing only the calculated components

Functions

- psqi_compute_comp2(): calculate the component 2 (sleep latency)
- psqi_compute_comp3(): calculate the component 3 (sleep duratione)
- psqi_compute_comp4(): calculate the component 4 (habitual sleep efficiency)
- psqi_compute_comp5(): calculate the component 5 (sleep disturbance)
- psqi_compute_comp7(): calculate the component 7 (daytime dysfunction)
- psqi_compute_global(): calculate the global scores, sum of all components

---

psqi_compute_time_in_bed

PSQI compute time in bed

Description

PSQI compute time in bed

Usage

```
psqi_compute_time_in_bed(
  risingtime,
  bedtime,
  risingtime_func = lubridate::hm,
  bedtime_func = lubridate::hm
)
```

Arguments

risingtime	column name with rising time (HH:MM:SS) (psqi_03)
bedtime	column name with bedtime (HH:MM:SS) (psqi_01)
risingtime_func	function to convert time to Period
bedtime_func	function to convert time to Period

---

tas_compute *Compute the TAS factors*

Description

Compute the TAS factors

Usage

```
tas_compute(
  data,
  reverse_cols = c(tas_04, tas_05, tas_10, tas_18),
  identify_cols = c(tas_01, tas_03, tas_06, tas_07, tas_09, tas_13, tas_14),
  describe_cols = c(tas_02, tas_04, tas_11, tas_12, tas_17),
 thinking_cols = c(tas_05, tas_08, tas_10, tas_15, tas_16, tas_18, tas_19, tas_20),
  prefix = "tas_",
  keep_all = TRUE
)
```

Arguments

data	Data containing TAS data
reverse_cols	Columns that need reversing
identify_cols	Columns for the "identify feeling" factor
describe_cols	Columns for the "describing feelings" factor
thinking_cols	Columns for the "externally oriented thinking" factor
prefix	string to prefix column names of computed values
keep_all	logical, append to data.frame

time_alter — *Turn strings of H:M to time*

Description

Turn strings of H:M to time

Usage

```
time_alter(x, unit = "minute", time_func = lubridate::hm)
```

Arguments

`x`	string of class lubridate::hms "HH:MM:SS"
`unit`	unit to convert to
`time_func`	a function to convert x to a lubridate time vector. Default is lubridate::hms

Examples

```
time <- c("02:33")
time_alter(time)
time_alter(time, "minute")
```

time_deci2period — *Turn time into period*

Description

Turn strings in class hms into periods of time.

Usage

```
time_deci2period(x, unit = "hour", type = "hm")
```

Arguments

`x`	string of class lubridate::hms "HH:MM:SS"
`unit`	unit to convert to
`type`	hms or hm

Value

Period

Examples

```
time_deci2period(8.5)
time_deci2period(1.25, "minute")
```

time_factor *Factor time of day*

Description

Takes a vector of HH:MM (HH:MM:SS) information and categorizes these by a 4 level factor of time of day.

Usage

```
time_factor(x, time_func = lubridate::hms, tod = time_of_day())
```

Arguments

x	character vector of times
time_func	a function to convert x to a lubridate time vector. Default is lubridate::hms
tod	list defining when the breakpoints for the various time of day distinctions.

Value

factor vector

Examples

```
time_factor(c("12:23", "15:59", "22:10", "8:13"))
```

time_hms2deci *Turn string time into decimal*

Description

Turn string time into decimal

Usage

```
time_hms2deci(x, unit = "hour")
```

Arguments

x	string of class lubridate::hms "HH:MM:SS"
unit	unit to convert to

Value

numeric vector

Examples

```
time <- lubridate::hms("02:33:12")
time_hms2deci(time)
time_hms2deci(time, "minute")
```

`time_of_day` *Create list of time of day break points*

Description

Create list of time of day break points

Usage

```
time_of_day(morning = c(5, 12), afternoon = c(12, 17), evening = c(17, 21))
```

Arguments

`morning`	vector of two for the hours where morning start or end in 24H
`afternoon`	vector of two for the hours where afternoon start or end in 24H
`evening`	vector of two for the hours where evening start or end in 24H

Value

list of fours times of day classifying the 24H of the day

Examples

```
time_of_day()
```

`zygo_calc` *Zygocity - Calculate item*

Description

Classification:

This note contains a brief description of the algorithm used to determine zygocity in recruitment in the 2000s.

Name	Answer questions about...	Used for
Drop	You and your twin were like two drops of water in childhood	Pairs and singles
Stranger	Strangers had trouble telling the difference when you were children	Pairs and singles
Eye	Similarity in terms of eye color	Pairs
Voice	Similarity in terms of voice	Single

Dexter	Similarity in Dexterity	Pairs and Singles
Belief	What you believe yourself	Pairs and Singles

"Single" twins here means those who have responded alone, i.e. there is no data available for both in the pair. The similarity questions that are not found in the table above, e.g. whether or not family members had problems distinguishing the twins is not used in the classification.

Weights:

During calculations of the entire zygocity score, weights are applied to the different categories, depending on whether one or both twins have responded to the questionnaire.

Name	Answer questions about...	Factor single	Factor pair
Drop	You and your twin were like two drops of water	1.494	2.111
Stranger	Strangers had trouble seeing the difference	0.647	0.691
Eye	Similarity in terms of eye color		0.394
Voice	Similarity in terms of voice	0.347	
Dexter	Dexterity Similarity	0.458	0.366
Belief	What you believe yourself	0.417	0.481
	Constant term in the formula	0.007	- 0.087

Usage

```
zygo_calc(x, type, n = "single", recode = TRUE)
```

Arguments

x	integer vector of answers to one of the questionnaire questions. Should not be longer than 2.
type	type of question the vector is from. "drop", "stranger, "dexterity", "voice", "eye", or "belief".
n	string indicating number of twins in the pair available. Either "single" or "pair".
recode	logical indicating if data should be recoded from 1-5(7) to -1. 0. 1.

Value

single value of calculated score based on recoded vector and multiplied with correct factor weight.

Examples

```
zygo_calc(c(1), type = "eye")
zygo_calc(c(1,3), type = "belief", n = "pair")
zygo_calc(c(4), type = "voice")
```

zygo_compute — *Compute weighted zygocity*

Description

The Zygocity questionnaire was developed by the Norwegian Public Health Institute (FHI; Folkehelseinstituttet) for their twin registry studies. Its a series of questions probing the similarities between twins, to determine if they are mono- or dizygotic.

Usage

```
zygo_compute(
  data,
  twin_col,
  cols,
  recode = TRUE,
  prefix = "zygo_",
  keep_all = FALSE
)
```

Arguments

`data`	dara.frame with the relevant data
`twin_col`	column that codes for twin pairs. Each twin should have the same identifier here.
`cols`	columns that contain the zygocity data. Use tidy-selectors
`recode`	logical indicating if data should be recoded from 1-5(7) to -1. 0. 1.
`prefix`	string to prefix column names of computed values
`keep_all`	logical, append to data.frame

Details

Classification:

This note contains a brief description of the algorithm used to determine zygocity in recruitment in the 2000s.

Name	Answer questions about...	Used for
Drop	You and your twin were like two drops of water in childhood	Pairs and singles
Stranger	Strangers had trouble telling the difference when you were children	Pairs and singles
Eye	Similarity in terms of eye color	Pairs
Voice	Similarity in terms of voice	Single
Dexter	Similarity in Dexterity	Pairs and Singles
Belief	What you believe yourself	Pairs and Singles

"Single" twins here means those who have responded alone, i.e. there is no data available for both in the pair. The similarity questions that are not found in the table above, e.g. whether or not family members had problems distinguishing the twins is not used in the classification.

Weights:
During calculations of the entire zygocity score, weights are applied to the different categories, depending on whether one or both twins have responded to the questionnaire.

Name	Answer questions about...	Factor single	Factor pair
Drop	You and your twin were like two drops of water	1.494	2.111
Stranger	Strangers had trouble seeing the difference	0.647	0.691
Eye	Similarity in terms of eye color		0.394
Voice	Similarity in terms of voice	0.347	
Dexter	Dexterity Similarity	0.458	0.366
Belief	What you believe yourself	0.417	0.481
	Constant term in the formula	0.007	- 0.087

Coding:
"Form value" is the value the answer option has in the data file. "Score value" is the value used in the algorithm when zygocity is calculated.

Variable	Answer option	Form value	Score value
Drop	Like two drops of water	1	1
	Like most siblings	2	-1
	Don't know	3	0
Stranger	Often	1	1
	Occasionally	2	0
	Never	3	-1
	Don't know	4	0
Belief	Monozygotic	1	1
	Dizygotic	2	-1
	Don't know	3	0
Eye, Voice & Dexter	Exactly the same	1	1
	Almost like	2	0
	Different	3	-1
	Don't know	4	0

No answer option is used directly in the calculations, only the score values. In the following, it is these values (-1, 0 or 1) that are used in the algorithms. E.g. has Drop in the formula value 1 for a positive answer to whether the twins were equal to two drops of water.
The higher the absolute value of the final score, the more certain / clearer the classification. For answers that reveal greater uncertainty about the similarity (e.g. a greater proportion of "almost" and "don't know"), the value will be closer to zero.

Pair formula:
For pairs where both have answered, the pair's average values for all score values are first calculated. That is Drop = (Drop1 + Drop2) / 2, etc., where Drop1 is the score value of the response from twin 1 and Drop2 is the score value of the response from twin 2 in the same pair.

The sign of this "pair score" is then used to determine zygocity in the same way as for "single": Negative value means double, positive value means single.

Single formula:

If only one twin in the pair has responded, the following is calculated:

The sign of this "single score" is then used to determine the zygocity: Negative value means double egg, positive value means single egg.

Column names:
By default, the functions assume that columns have names in the manner of `zygocity_XX` where `XX` is a zero-padded (i.e. zero in front of numbers below 9, eg. `09`) question number of the inventory. You may have column names in another format, but in that case you will need to supply to the functions the names of those columns using tidy-selectors (see the tidyverse packages for this). The columns should adhere to some naming logic that is easy to specify.

Data values:
The values in the columns should be the item number of the question that was answered (i.e. `1`, `2`, or `3`, and for some questions also `4`).

Value

data.frame with computed values

`zygo_recode` *Zygocity - recode variables*

Description

Coding:

"Form value" is the value the answer option has in the data file. "Score value" is the value used in the algorithm when zygocity is calculated.

Variable	Answer option	Form value	Score value
Drop	Like two drops of water	1	1
	Like most siblings	2	-1
	Don't know	3	0
Stranger	Often	1	1
	Occasionally	2	0
	Never	3	-1
	Don't know	4	0
Belief	Monozygotic	1	1
	Dizygotic	2	-1
	Don't know	3	0
Eye, Voice & Dexter	Exactly the same	1	1
	Almost like	2	0
	Different	3	-1
	Don't know	4	0

No answer option is used directly in the calculations, only the score values. In the following, it is these values (-1, 0 or 1) that are used in the algorithms. E.g. has Drop in the formula value 1 for a positive answer to whether the twins were equal to two drops of water.

Usage

```
zygo_recode(x, type)
```

Arguments

x	vector of numbers, either 1:3 or 1:4
type	Type of question to recode. Can either be 05, 06, 07 or 08, or drop, stranger, dexterity, voice, eye or belief.

Value

return a vector with 0, -1 or 1.

Examples

```
zygo_recode(c(1:4, NA), type = "eye")
zygo_recode(c(1:4, NA), type = "voice")
zygo_recode(c(1:3, NA), type = "drop")
```

---

zygo_type *Find how many twins have answered*

Description

The zygocity calculations are different depending on wheather both twins have answered the questionnaire or not. This convenience function help determine, based on the column coding for twin pairs, if one or two twins are present in the data with complete viable data. If both twins are in the data, but one twin has incomplete data, the function will return "single" for the remaining twin.

Usage

```
zygo_type(data, twin_col, cols = starts_with("zygo"))
```

Arguments

data	dara.frame with the relevant data
twin_col	column that codes for twin pairs. Each twin should have the same identifier here.
cols	columns that contain the zygocity data. Use tidy-selectors

Value

full data frame with twin type appended

zygo_weighted *Calculate weighted zygocity item*

Description

Calculate the item score of a question. Function takes a single vector, with information on the question type and the twin type ('single' or 'pair') and calculates the zygocity item score.

Usage

```
zygo_weighted(x, type, n = "single")
```

Arguments

x	vector or recoded zygocity data (-1, 0, 1)
type	string. one of 'drop', 'stranger', 'dexterity', 'voice', 'eye' or 'belief.'
n	string, 'pair' if both twins have answered, 'single' if not.

Value

numeric vector of weighted data

Index