Package 'eye'

June 8, 2020

Title Analysis of eye data

Version 0.0.0.9000

Description eye is dedicated to facilitate ophthalmic research.
Its core functions va(), eyes(), myop(), blink(), reveal() and
age() are designed to help with common tasks in eye research:
Visual acuity conversion for snellen, logMAR and ETDRS, counting patients and
eyes, summarising data with common statistics, calculating age of patients,
and reshaping eye-specific data from wide to long.
eye also contains a real life data set of people who received intravitreal
injections due to age-related macular degeneration in
Moorfields Eye Hospital, London, UK.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.0

Depends R (>= 4.0)

Suggests testthat,
knitr,
rmarkdown

Imports dplyr (>= 0.8.5),
cli (>= 2.0.2),
english (>= 1.2-5),
lubridate (>= 1.7.8),
ggplot2 (>= 3.3.0),
magrittr (>= 1.5),
rlang (>= 0.4.6),
stringr (>= 1.4.0),
tibble (>= 3.0.1),
tidyr (>= 1.0.3),
tidyselect (>= 1.1.0)

VignetteBuilder knitr

R topics documented:

age *age*

Description

calculates age in years, either as duration or as period

Usage

```
age(from_date, to_date = lubridate::now(), period = FALSE, dec = 1)
```

Arguments

`from_date`	start date
`to_date`	end date
`period`	default FALSE: output as a duration. If TRUE, output as a period
`dec`	How many decimals are displayed

See Also

original thread on stackoverflow.com from which this function was inspired

Other convenience functions: `csv()`

Examples

```
age("1984-10-16")

dob <-  c("1984-10-16", "2000-01-01")
test_date <-  as.Date(dob) + c(15000, 20000)
age(dob, test_date)
```

`amd` *Real life data of patients with neovascular AMD*

Description

A dataset containing anonymised real life human subjects data on eyes with treatment naive neovascular age-related macular degeneration (AMD), which underwent intravitreal anti-VEGF therapy with ranibizumab and/or aflibercept.

The data was collected in Moorfields Eye Hospital, London, UK.

(Information governance sign off Moorfields Eye Hospital 19/07/2018)

Usage

```
data("amd")
```

Format

A data frame with 40764 rows and 7 variables:

Id	Anonymized patient identifier
Eye	Left or right eye of patient (0 = right, 1 = left)
FollowupDays	Days after date of first appointment (0 = first appointment)
BaselineAge	Age (years) at day of first appointment
Gender	Gender of patient (0 = male, 1 = female)
VA_ETDRS_Letters	Visual acuity in Early Treatment Diabetic Retinopathy Study letters
InjectionNumber	Current number of injection at appointment date

Source

https://datadryad.org/stash/dataset/doi:10.5061/dryad.97r9289

`blink` *Your data in a blink of an eye*

Description

`blink` summarises your data tailored to the need of ophthalmic research: It looks for VA and IOP columns and summarises those with common statistics. In order to make it work, it requires specific column naming - please see section "column names" and "data coding".

Usage

```
blink(x, va_cols = NULL, iop_cols = NULL, fct_level = 0:4)
```

Arguments

`x`	data frame
`va_cols`	if specified, overruling automatic VA columns selection. tidyselection supported
`iop_cols`	if specified, overruling automatic IOP columns selection. tidyselection supported
`fct_level`	Remove columns for summarising when all unique values fall into range. character or numeric vector, default `1:4`

Details

`blink` is basically a wrapper around myop, eyes and reveal:

- Duplicate rows are always removed
- Column names are prepared for myopization (see myop)
- VA will always be converted to logmar

Data coding

- Only common codes supported:
- **eyes**: "r", "re", "od", "right" - or numeric coding r:l = 0:1 or 1:2
- **Visual acuity**: "VA", "BCVA", "Acuity"
- **Intraocular pressure**: "IOP", "GAT", "NCT", "pressure"

Column name rules

- No spaces!
- Do not use numeric coding for eyes in column names
- Separate eye and VA and IOP codes with **underscores** ("bcva_l_preop", "VA_r", "left_va", "IOP_re")
- Avoid separate VA or IOP codes if this is not actually containing VA/ IOP data (e.g. "stableVA" instead of "stable_va", ChangeIOP instead of "change_IOP")
- Keep names short
- Don't use underscores when you don't have to. Consider each section divided by an underscore as a relevant characteristic of your variable. ("preop" instead of "pre_op", "VA" instead of "VA_ETDRS_Letters")
- Use common codes for your patient column (see eyes, section Guessing) (e.g., "pat", "patient" or "ID", ideally both: "patientID" or "patID")
- **Don't be too creative with your names!**

Names examples

Good names:

`-c("patid","surgery_right","iop_r_preop","va_r_preop","iop_r","iop_l")`

OK names

`-c("Id","Eye","BaselineAge","VA_ETDRS_Letters","InjectionNumber")`: Names are long and there are two unnecessary underscore in the VA column. Better just "VA" `-c("id","r","l")`: All names are commonly used (good!), but which dimension of "r"/"l" are we exactly looking at?

Bad names (eye will fail)

- `c("id","iopr","iopl","VAr","VAl")`: eye won't be able to recognize IOP and VA columns
- `c("id","iop_r","iop_l","stable_iop_r","stable_iop_l")`: eye *may* wrongly identify the (probably logical) columns "stable_iop" as columns containing IOP data. Better maybe: "stableIOP_l"
- `c("person","goldmann","vision")`: eye will not recognize that at all

tidy data

blink and myop work more reliably with clean data (any package will, really!). clean data.

column removal

Done with remCols: Removes colums that only contain values defined in *fct_levels* or logicals from selected columns (currently for both automatically and manually selected columns). fct_levels are removed because they are likely categorical codes.

Examples

```
blink(amd)

messy_df <- data.frame( id = letters[1:3],
iop_r_preop = sample(21:23), iop_r_postop = sample(11:13),
iop_l_postop = sample(11:13), iop_l_preop = sample(31:33),
va_r_preop = sample(41:43),  va_l_preop = sample(41:43),
va_r_postop = sample(51:53), va_l_postop = sample(45:47)
)
blink(messy_df)
```

`clean_va` *Visual acuity entry cleaner*

Description

VA cleaning:

1. `isNAstring()`: Replacing empty placeholders (".","", "(any number of empty space)", "NULL", "NA") with NA with
2. Removing "plus" and "minus" from snellen notation
3. `convert_NLP()` Simplifying the notation for qualitative VA notation (NPL becomes NLP, PL becomes LP) .

Usage

```
clean_va(x)

convert_NLP(x, replace_PL = c(pl = "lp", npl = "nlp"))

isNAstring(x, full = c("\\.+", "", "\\s+", "n/a", "na", "null"))
```

Arguments

`x`	Vector with VA entries
`replace_PL`	named vector how to rename qualitative VA
`full`	vector of full strings to be replaced by NA

See Also

Other VA cleaner: `va()`

Other VA cleaner: `va()`

`csv` *csv*

Description

wrapper around write.csv with default 'row.names = FALSE'. Will use the name of the data frame for the generated .csv file.

Usage

```
csv(x, name = NULL)
```

Arguments

`x`	data frame
`name`	Filename (Default: Name of dataframe). If provided, character string (.csv extension added automatically)

See Also

Other convenience functions: `age()`

Examples

```
## Not run:
csv(amd)

## End(Not run)
```

`eyes` *Count patients and eyes*

Description

Counts number of patients and eyes (right and left).

`eyestr`: identical to `eyes(x,report = TRUE,...)`

Usage

```
eyes(x, id = NULL, eye = NULL, report = FALSE, ...)

eyestr(x, id = NULL, eye = NULL, small_num = TRUE, para = FALSE, UK = FALSE)
```

Arguments

`x`	required. (data frame)
`id`	Patient identifying column
`eye`	Eye identifying colum.
`report`	if TRUE, text returned for report
`...`	passed to eyes_to_string
`small_num`	If TRUE: writing numbers <= 12 as words
`para`	If TRUE: Adding "A total of" to comply with most journal standards and to avoid awkward long numnbers.
`UK`	Logical, Use UK (English) style (TRUE) or USA (American) style (FALSE).

Details

`eyes` guesses columns that identify patients and eyes.

Value

Vector with count of patients and eyes

`eyestr`: Text which can be used for reports

Guessing

For any below, **cases are always ignored** (you can write in upper or lower case, as you please)

id and **eye** arguments overrule the name guessing for the respective columns.

patient ID columns:

- First, `eyes` is looking for names that contain both strings "pat" and "id" (the order doesn't matter)
- Next, it will look for columns that are plainly called "ID"
- Last, it will search for all names that contain either "pat" or "id"

eye variable column:

- `eyes` looks for columns called either "eye" or "eyes"

Eye coding

- eyes recognizes integer coding 0:1 and 1:2, with right being the lower number. For strings coding it recognizes right eyes: c("r", "re", "od", "right") and left eyes: c("l", "le", "os", "left")

Report

Using eyes_to_string to parse the output of eyes into a text which you can use for reports. Arguments to `eyes_to_string` are passed via **...**:

- **small_num** If TRUE (default): numbers <= 12 as words
- **para** If TRUE (not default): Adding "A total of" to comply with most journal standards and to avoid awkward long numbers.
- **UK** TRUE: UK style (English) or FALSE (default): US style (American).

Examples

```
eyes(amd)
eyestr(amd, para = TRUE)
```

`eyes_to_string` *Eye count to strings*

Description

Eye count to strings

Usage

```
eyes_to_string(x, small_num = TRUE, para = FALSE, UK = FALSE)
```

Arguments

`x`	vector of one or two
`small_num`	If TRUE: writing numbers <= 12 as words
`para`	If TRUE: Adding "A total of" to comply with most journal standards and to avoid awkward long numnbers.
`UK`	Logical, Use UK (English) style (TRUE) or USA (American) style (FALSE).

Value

String to paste into text

geom_trail *geom_trail*

Description

Mark a trail with a plot just like the base plot type = "b". You can also leave the dots blank, thus allowing use of text instead of points (see examples).

Usage

```
geom_trail(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)

GeomTrail
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()` or `aes_()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x,10)`).
`stat`	The statistical transformation to use on the data for this layer, as a string.
`position`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`na.rm`	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.
`show.legend`	logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

`...`	Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

Format

An object of class `GeomTrail` (inherits from `GeomPoint`, `Geom`, `ggproto`, `gg`) of length 3.

Additional arguments

gap gap between points and lines

Aesthetics

`geom_trail` understands the following aesthetics (required aesthetics are in bold):

- `x`
- `y`
- `alpha`
- `color`
- `linetype`
- `size` size of points - 0 for empty space
- `linesize`

See Also

The geom was modfied from the suggestion by user teunbrand on Stackoverflow in this thread

Examples

```
library(ggplot2)
library(dplyr)

ggplot(pressure, aes(temperature, pressure)) +
  geom_ribbon(aes(ymin = pressure - 50, ymax = pressure + 50), alpha = 0.2) +
  geom_trail()

amd_aggr <-
amd %>%
  group_by(
    age_cut10 = cut_width(BaselineAge, 10),
    days_cut90 = cut_width(FollowupDays, 90, labels = seq(0, 810, 90))
  ) %>%
  summarise(mean_va = mean(VA_ETDRS_Letters))

p <- ggplot(amd_aggr, aes(days_cut90, mean_va, color = age_cut10)) +
       theme_classic() +
       labs(
         x = "Follow up time [Days]", y = "Mean VA [ETDRS letters]",
         color = "Age strata"
       )

p + geom_trail(aes(group = age_cut10))
```

```
p + geom_trail(aes(group = age_cut10), size = 0) +
  geom_text(aes(label = round(mean_va, 0)), show.legend = FALSE)
```

`makeContent.trail` *grid draw method geom trail*

Description

underlying drawing method for paths in geom_trail

Usage

```
## S3 method for class 'trail'
makeContent(x)
```

Arguments

`x`	grob object passed to method

Author(s)

Teun van den Brand

`myop` *Myopic eye data*

Description

Pivot "eye" variable to one column

Usage

```
myop(x, var = "value")

myope(x, var = "value")

myopic(x, var = "value")
```

Arguments

`x`	data frame
`var`	Character vector of length 1 specifying the variable if there is only one column per eye with no further info on the variable (default "value")

Details

Out of convenience, data is often entered in a very "wide" format: there will be two columns for the same variable, one column for each eye. myop will pivot the eye variable to one column and keep all other variables wide. E.g., eight columns that store data of four variables for two eyes will be pivoted to 5 columns (one eye and four further variable columns, see also *examples*).

myop requires a specific data format

If there is a column called "eye" or "eyes", myop will not make any changes - because the data is then already assumed to be in long format. If you neverthess have columns with eye-specific values, then you have messy data. Maybe, you could remove or rename the "eye" column and then let myop do the work.

myop will only recognize meaningful coding for eyes:

- Right eyes: *"r", "re", "od", "right"*
- Left eyes: *"l", "le", "os", "left"*
- for other codes see also set_codes The strings for eyes need to be **separated by period or underscores**. (Periods will be replaced by underscores). Any order is allowed.
- **Will work**: "va_r", "right_morningpressure", "night_iop.le", "gat_os_postop"
- **Will fail**: "VAr", "rightmorningPressure", "night_IOPle", "gatOSpostop"

An exception is when there is only one column for each eye. Then the column names can consist of eye strings (see above) only. In this case, *var* will be used to "name" the resulting variable.

If there are only eye columns in your data (should actually not happen), myop will create identifiers by row position.

Please always check the result for plausibility. Depending a lot on how the data was entered, the results could become quite surprising. There is basically a nearly infinite amount of possible combinations of how to enter data, and it is likely that myop will not be able to deal with all of them

internal preparation

- Rename data names with myop_rename, replacing "." with "_"
- Use of `sort_substr()` - sorting eye strings first, then strings coding for methods (IOP/VA), then the rest.

myopization

The actual work is done with myopizer and myop_pivot

Examples

```
# Example to clean a bit messy data frame
iopva <- data.frame(
  id = c("a", "e", "j", "h"),
  va_r = c(37L, 36L, 33L, 38L),
  iop_r = c(38L, 40L, 33L, 34L),
  va_l = c(30L, 39L, 37L, 40L),
  iop_l = c(31L, 34L, 33L, 31L)
)
myop(iopva)

iop_wide <- data.frame(id = letters[1:3],  r = 11:13 , l = 14:16)
# the variable has not been exactly named, so you can specify
# it with the var argument
myop(iop_wide, var = "iop")
```

`myopizer` *myopizer*

Description

internal function, checks and prepares data frames for "myopization": Removing duplicates, returning myopized data if criteria fulfilled (No "eye" column, more than one variable column with eye codes as partial string). Names need to be prepared with myop_rename beforehand.

Usage

```
myopizer(x, var = "value")
```

Arguments

`x`	data frame
`var`	Character vector of length 1 specifying the variable if there is only one column per eye with no further info on the variable (default "value")

See Also

Other myopizer: `myop_pivot()`, `myop_rename()`

`print_methods` *print VA class*

Description

S3 methods for VA classes "snellen", "logmar" and "etdrs". **snellen** is always also a character class-because it is more categorical than continuous. **logmar** and **etdrs** are both numerics (logMAR is double, etdrs is integer).

S3 methods for blink class

Usage

```
## S3 method for class 'snellen'
print(x, ...)

## S3 method for class 'logmar'
print(x, ...)

## S3 method for class 'etdrs'
print(x, ...)

## S3 method for class 'blink'
print(x, ...)
```

Arguments

x	object of class "blink"
...	arguments passed to print.default

`reveal` *reveal*

Description

Shows commonly used summary statistics

Usage

```
reveal(x, by = NULL, dec = 1)
```

Arguments

x	data frame, numeric vector, or list of numeric vectors
by	character vector with the names of the columns
dec	how many decimals are displayed

Details

Character vectors (or character columns) will be removed.

Value

data frame

See Also

Other revealer: `reveal_methods`, `reveal_split()`

Examples

```
x = y = z = c(rnorm(20), NA)
mylist <- list(x = x, y = y, z = z)
## vectors
reveal(x)
reveal(1:10)
## named or unnamed list
reveal(mylist)
set.seed(42)
mydf <- cbind(group = rep(letters[1:3], 4),
setNames(as.data.frame(replicate(c(rnorm(11), NA), n = 3)), letters[24:26]))
## data frames
reveal(mydf)
## data frames by group
reveal(mydf, by = "group")
```

`reveal_methods` *reveals little helper*

Description

S3 generic and methods

Usage

```
revealEye(x, ...)

## S3 method for class 'list'
revealEye(x, by, dec, ...)

## S3 method for class 'numeric'
revealEye(x, dec, ...)

## S3 method for class 'data.frame'
revealEye(x, dec, ...)

## Default S3 method:
revealEye(x, dec, ...)
```

Arguments

`x`	atomic vector, list of numeric vectors or data frame
`...`	further arguments passed to methods
`by`	grouping variable - character vector with column names can be several variables!
`dec`	how many decimals are displayed

Value

data frame

See Also

Other revealer: `reveal_split()`, `reveal()`

`set_codes` *Set string codes*

Description

currently only internal use for convenient set of coding strings

Usage

```
set_codes(
  r = c("r", "re", "od", "right"),
  l = c("l", "le", "os", "left"),
  iop = c("iop", "gat", "nct"),
  va = c("va", "bcva"),
  method = c("etdrs", "snellen", "logmar"),
  id = c("pat", "id"),
  quali = c("nlp", "lp", "hm", "cf"),
  vaPart = "acuit",
  ...
)
```

Arguments

`r`	right eyes
`l`	left eyes
`iop`	IOP codes
`va`	VA codes
`method`	VA methods
`id`	patient column codes
`quali`	quali VA codes
`vaPart`	partial VA codes
`...`	further codes to set

See Also

Other string matching functions: `getElem`, `sort_substr()`, `str_func_facs`

`va` *Visual acuity notation conversion*

Description

Cleans and converts visual acuity notations (classes) between Snellen (decimal, meter and feet), ETDRS, and logMAR. `va` detects the VA class and will convert to logMAR as default.

Usage

```
va(x, to = "logmar", type = NULL, from = NULL)
```

Arguments

`x`	Vector with visual acuity entries. Must be atomic. Snellen fractions need to be entered with "/"
`to`	To which class to convert. "etdrs", "logmar" or "snellen" - any case allowed
`type`	To which Snellen notation to convert: "m", "dec" or "ft"
`from`	From which class to convert. "etdrs", "logmar" or "snellen" - any case allowed. Ignored if implausible

Details

Each class can be converted from one to another, and va() converts to logMAR by default. In case of ambiguous detection, logMAR is selected as default, but you can overrule this with "from". However, from will be ignored if it does not make sense. For further details, see the sections below.

Value

vector of class set with `to` argument

VA conversion

- **logMAR to ETDRS**: logMAR rounded to the first digit and converted with the chart.
- **Snellen to logMAR**: logMAR = -1 * log10(snellen_frac)
- **Snellen to ETDRS**: ETDRS = 85 + 50 * log10(snellen_frac) Gregori et al..
- **ETDRS to logMAR**: logMAR = -0.02 * etdrs + 1.7 Beck et al.
- **Hand movements and counting fingers** are converted following Schulze-Bonsel et al.
- **(No) light perception** are converted following the suggestions by Michael Bach
- **To Snellen**: Although there seems to be no good statistical reason to convert back to Snellen, it is a very natural thing to eye specialists to think in Snellen. A conversion to snellen gives a good gauge of how the visual acuity for the patients are. However, back-conversion should not be considered an exact science and any attempt to use formulas will result in very weird Snellen values that have no correspondence to common charts. Therefore, Snellen matching the nearest ETDRS and logMAR value in the va_chart are used.

Accepted VA formats

- Snellen fractions (meter/ feet) need to be entered as fraction with "/".
- **when converting to ETDRS or logMAR**: any fraction is allowed , e.g. 3/60 and 2/200 will also be recognized.
- **When converting between Snellen fractions**: *has to be either 6/ or 20/*. Other fractions will not be recognized - see **"Examples"**
- ETDRS must be integer-equivalent between 0 and 100 (integer equivalent means, it can also be a character vector)
- logMAR must be between -0.3 and 3.0
- Qualitative must be either of PL, LP, NLP, NPL, HM, CF (any case allowed)
- Any element which is not recognized will be converted to NA
- Vectors containing several notations ("mixed") are guessed and converted element by element with which_va_dissect and va_dissect

VA detection

- Internally done with `which_va()` based on the following rules
- if x integer and 3 < x <= 100: `etdrs`
- if x integer and 0 <= x <= 3: `logmar`, but you can choose `etdrs`
- if x numeric and -0.3 <= x <= 3: `logmar`
- if x numeric and all x in intersection(va_chart$logMAR, va_chart$snellen_dec): `logmar`, but you can choose `snellen`

- *non-mixed class*: if all x in va_chart$snellen_dec: snellen
- *mixed class* (which_va_dissect): snellen_dec not supported.
- if character and format x/y: snellen (fraction)
- if one of "CF", "HM", "LP", "PL", "NLP", or "NPL": quali
- if numeric x beyond the ranges from above: NA
- Any other string or NA: NA

Detection and convertion is on a vector as a whole by which_va(). If a "mixed" VA notation is found, which_va_dissect() and va_dissect() will be called instead for each VA vector element individually.

Problematic cases

There can be ambiguous cases for detection (detection defaults to logmar): x is one of 0,1,2,3 - This can be ETDRS and logMAR. x is one of c(1.5, 1, 0.8, 0.5, 0.4, 0.3, 0.2, 0.1, 0) - This can be snellen decimal or logMAR.

snellen decimals are a particular challenge and va may wrongly assign logMAR - this could happen if there are unusual snellen decimal values in the data which are not part of va_chart. E.g., check the values with unique(x).

VA cleaning

NAare assigned to missing entries or empty strings such as "." or "", "plus" and "minus" from Snellen entries are removed and the notation for qualitative entriesis simplified. For more details see clean_va()

See Also

Other VA converter: va_dissect(), va_methods, which_va()

Other VA cleaner: clean_va()

Examples

```
## will automatically detect VA class and convert to logMAR by default
## ETDRS letters
x <- c(23, 56, 74, 58)
va(x)

## ... or convert to snellen
va(x, to = "snellen")

## snellen, mixed with categories. Also dealing with those "plus/minus" entries
va(c("NLP", "NPL", "PL", "LP", "HM", "CF", "6/60", "20/200", "6/9",
 "20/40", "20/40+3", "20/50-2"))

## A mix of notations is also possible
x <- c("NLP", "0.8", "34", "3/60", "2/200", "20/40+3", "20/50-2")
va(x)

## Any fraction is possible, and empty values
x <- c("CF", "3/60", "2/200", "", "20/40+3", ".", "      ")
va(x)

## but this not any fraction when converting from one class to the other
```

```
x <- c("3/60", "2/200", "6/60", "20/200", "6/9")
va(x, to="snellen", type = "m")
```

va_chart — *Visual acuity conversion chart*

Description

Conversion between snellen, logMAR and ETDRS. Snellen feet, meter and decimal supported. Three qualitative common vision measures included (light perception, hand movement and counting fingers). Further details for conversion used can be found in va and va_methods

Usage

```
data("va_chart")
```

Format

A data frame with 29 rows and 5 variables:

snellen_ft snellen VA in feet

snellen_m snellen VA in meter

snellen_dec decimal snellen VA

logmar logMAR VA

etdrs VA in ETDRS letters

quali VA categories

See Also

- This chart and VA conversion formulas are based on charts in Holladay et al., Beck et al., Gregori et al., and https://www.nidek-intl.com/visual_acuity.html.

Categories **(no) light perception**, **counting fingers** and **hand movements** are converted following Schulze-Bonsel et al. and Michael Bach's suggestions

va_methods — *VA conversion methods*

Description

S3 methods for VA conversion

Usage

```
convertVA(x, to, ...)

## S3 method for class 'quali'
convertVA(x, to, snellnot, ...)

## S3 method for class 'snellen'
convertVA(x, to, snellnot, ...)

## S3 method for class 'logmar'
convertVA(x, to, snellnot, ...)

## S3 method for class 'etdrs'
convertVA(x, to, snellnot, ...)

## Default S3 method:
convertVA(x, to, snellnot, ...)
```

Arguments

`x`	vector of visual acuities
`to`	to which VA class to convert
`...`	further arguments passed to methods
`snellnot`	which snellen notation. One of "ft", "m" or "dec"

Details

VA can be snellen feet/meter/decimal, logMAR, ETDRS, or "qualitative" (Couting fingers, etc.)

- Snellen fractions need to be either form 6/x or 20/x
- ETDRS must be between 0 and 100
- logMAR must be between -0.3 and 3.0
- Qualitative must be PL, LP, NLP, NPL, HM, CF (any case allowed)

Any element which is not recognized will be converted to NA

For other conversion rules see va

Conversion

Although there seems to be no good statistical reason to convert back to Snellen, it is a very natural thing to eye specialists to think in Snellen. A conversion to snellen gives a good gauge of how the visual acuity for the patients are. However, back-conversion should not be considered an exact science and any attempt to use formulas will result in very weird Snellen values that have no correspondence to common charts. Therefore, Snellen matching the nearest ETDRS and logMAR value in the va_chart are used.

Further:

- logMAR to ETDRS: logMAR rounded to the first digit and converted with the chart.
- Snellen to logMAR: logMAR = -1 * log10(snellen_frac)
- Snellen to ETDRS: ETDRS = 85 + 50 * log10(snellen_frac) Gregori et al..
- ETDRS to logMAR: logMAR = -0.02 * etdrs + 1.7 Beck et al.

See Also

Other VA converter: `va_dissect()`, `va()`, `which_va()`

`which_va` *Guessing the VA class*

Description

Guessing the VA notation (VA class). Requires x that was prepared with clean_va

- `which_va`: guessing VA class for entire vector
- `which_va_dissect`: guessing VA class for each vector element

Usage

```
which_va(x)

which_va_dissect(elem)
```

Arguments

`x`	Vector with VA entries
`elem`	element of vector to convert

See Also

Other VA converter: `va_dissect()`, `va_methods`, `va()`

Index