Package 'eye'

June 25, 2020

Title Analysis of Eye Data

Version 0.1.0

Description A tool to facilitate common tasks in ophthalmic research: Conversion between different visual acuity notations (Snellen, logMAR and ETDRS), counting of patients, recode right and left eyes and reshape eye side specific variables between wide and long format. The 'eye' package also contains a real life data set of people with intravitreal injections with anti-vascular endothelial growth factor (anti-VEGF), made available by Fasler et al. (2019) <doi:10.1136/bmjopen-2018-027441>. Visual acuity conversion is based on Schulze-Bonsel et al. (2006) <doi:10.1167/iovs.05-0981>, Gregori et al. (2010) <doi:10.1097/iae.0b013e3181d87e04>, Beck et al. (2003) <doi:10.1016/s0002-9394(02)01825-1> and Bach (2007) <http:michaelbach.de/sci/acuity.html>.

License MIT + file LICENSE

Language en-US

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

Depends R (>= 4.0)

Suggests testthat, knitr, rmarkdown

Imports dplyr (>= 1.0.0), cli (>= 2.0.2), english (>= 1.2-5), lubridate (>= 1.7.8), magrittr (>= 1.5), purrr (>= 0.3.4), rlang (>= 0.4.6), stringr (>= 1.4.0), tibble (>= 3.0.1), tidyr (>= 1.0.3), tidyselect (>= 1.1.0)

VignetteBuilder knitr

NeedsCompilation no

Author Tjebo Heeren [aut, cre] (<https://orcid.org/0000-0001-5297-2301>), Antoine Fabri [ctb]

Maintainer Tjebo Heeren `<tjebo@gmx.de>`

Repository CRAN

Date/Publication 2020-06-25 12:20:03 UTC

R topics documented:

age 2
amd 3
blink 4
clean_va 6
eyes 7
hyperop 9
myop 10
print_methods 11
recodeye 12
reveal 13
reveal_methods 14
snellen_steps 15
va 16
va_chart 19
va_methods 20

Index **22**

`age` *age*

Description

calculates age in years, as durations or periods

Usage

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

Arguments

`from_date`	start date
`to_date`	end date
`period`	Calculating period (TRUE) or duration (FALSE- default)
`dec`	How many decimals are displayed

Value

Numeric vector

Author(s)

Antoine Fabri and Tjebo Heeren

See Also

OP on stackoverflow from which this function was inspired. Read about periods and durations

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 anonymized 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.

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

Details

The data was collected in Moorfields Eye Hospital, London, UK. (Information governance sign off Moorfields Eye Hospital 19/07/2018)

Data was accessed on the 25th May 2020

Spurious data entries

Note there are erroneous visual acuity entries in this data set which I noticed during the work on this package. The data set curator has been contacted and it it was concluded that these were erroneous entries in the original medical health records. I decided to keep the values in the data set and wait for the final decision how to proceed from the data set curator (if they are going to replace it with missing values or not). I believe this is a great example for the challenges of real life data and a reminder to remain vigilant when doing data analysis.

Source

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

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

Description

`blink` summarizes 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". For more details how blink works, see `vignette("eye")`

Usage

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

Arguments

`x`	data frame
`va_to`	to which VA notation (passed to `va()`)
`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 Summarizing 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

Value

object of class `blink` and `list`. Class blink contains the myopized data, count of patients and eyes, and summaries for visual acuities and intraocular pressure.

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 columns 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.

See Also

About tidyselection.

How to rename your columns (two threads on stackoverflow.com):

- Rename columns 1
- Rename columns 2

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", "N/A") - any cases - with NA
2. `convert_NLP()` Simplifying the notation for qualitative VA notation (NPL becomes NLP, PL becomes LP)
3. Removing non-Snellen character strings

Usage

```
clean_va(x, quali = c("nlp", "lp", "hm", "cf"))

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

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

Arguments

`x`	Vector with VA entries
`quali`	strings for qualitative visual acuity entries
`replace_PL`	named vector how to rename qualitative VA
`tolower`	if TRUE, x will be converted to lower first
`full`	vector of full strings to be replaced by NA

Value

character vector

See Also

Other VA cleaner: `va()`

Other VA cleaner: `va()`

`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 column.
`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 numbers.
`UK`	Logical, Use UK (English) style (TRUE) or USA (American) style (FALSE).

Details

eyes guesses columns that identify patients and eyes.

Value

eyes: Named integer vector with count of patients and eyes

eyestr: Character string - can be directly pasted into 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)
```

hyperop *Hyperopic eye data*

Description

Pivot eye-related variables to two columns

Usage

```
hyperop(x, cols, eye = NULL)
```

Arguments

x	data frame
cols	columns which should be made "wide". Tidyselection supported
eye	eye column (default looking for "eye" or "eyes", all cases)

Details

Basically the opposite of `myop()` - a slightly intelligent wrapper around `tidyr::pivot_longer()` and `tidyr::pivot_wider()` Will find the eye column, unify the codes for the eyes (all to "r" and "l") and pivot the columns wide, that have been specified in "cols".

Good names and tidy data always help!

For more information about shaping data and good names, see `vignette("eye")`, or `?blink` or `?myop`

Value

A tibble, see also tibble::tibble

See Also

About tidyselection

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_iop <- myop(iopva)
hyperop(myop_iop, cols = matches("va|iop"))
```

`myop` *Myopic eye data*

Description

Pivot "eye" variable to one column

Usage

```
myop(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 *also* 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

Value

A tibble, see also tibble::tibble

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")
```

`print_methods` *print eye classes*

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

Value

No return value, called for side effects (printing)

`recodeye` *Recode eyes*

Description

recoding eyes to "r" and "l"

Usage

```
recodeye(
  x,
  to = c("r", "l"),
  eyecodes = list(c("r", "re", "od", "right"), c("l", "le", "os", "left")),
  numcode = NULL
)
```

Arguments

x	vector of strings
to	to which codes. Right coded first.
eyecodes	list of substrings which should be converted to right and left eyes - first vector for right, then for left eyes
numcode	if you have numeric coding which is not 0:1 or 1:2 for right:left, specify it here.

Value

Character vector

See Also

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

Examples

```
x <- c("r", "re", "od", "right", "l", "le", "os", "left")
recodeye(x)
## chose the resulting codes
recodeye(x, to = c("right", "left"))
x <- 1:2
recodeye(x)
## or, if right is coded with 2)
recodeye(x, numcode = 2:1)
```

`reveal` *reveal*

Description

Shows commonly used summary statistics

Usage

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

Arguments

`x`	data frame, numeric vector, or list of numeric vectors
`by`	character vector with the names of the columns. Can be several variables!
`dec`	how many decimals are displayed
`funs`	not really meant to be used at the moment - change the Summarizing functions with a named(!) list of functions

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, funs, ...)

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

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

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

Arguments

`x`	data frame, numeric vector, or list of numeric vectors
`...`	further arguments passed to methods
`by`	character vector with the names of the columns. Can be several variables!
`dec`	how many decimals are displayed
`funs`	not really meant to be used at the moment - change the Summarizing functions with a named(!) list of functions

Value

data frame

See Also

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

---

snellen_steps   *Convert plus minus entries*

Description

used in conversion method for class snellen

- Removing "plus" and "minus" from snellen notation
  - if entry -1 to +3 : take same Snellen value
  - if <= -2 : take Snellen value one line below
  - if >+3 (unlikely, but unfortunately not impossible): Snellen value one line above

Usage

```
snellensteps(y)
```

Arguments

y	Vector with VA entries of class snellen - needs to be in format xx/yy

Value

character vector of Snellen entries

snellen_steps

Snellen are unfortunately often entered with "+/-", which is a violation of a psychophysical method designed to assign one unambiguous value to visual acuity, with non-arbitrary thresholds based on psychometric functions. Therefore, transforming "+/-" notation to actual results is in itself problematic and the below suggestion to convert it will remain an approximation to the most likely "true" result. Even more so, as the given conditions should work for charts with 4 or 5 optotypes in a line, and visual acuity is not always tested on such charts. Yet, I believe that the approach is still better than just omitting the letters or (worse) assigning a missing value to those entries.

See Also

https://en.wikipedia.org/wiki/Psychometric_function

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

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_logmar = TRUE,
  logmarstep = FALSE,
  mixed = FALSE
)
```

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_logmar	chose logmar when guessing between two notations (logmar vs. snellen decimal or logmar vs. etdrs)
logmarstep	how +/- entries are evaluated. FALSE: increase/decrease Snellen fractions by lines. TRUE: plus/minus entries equivalent to 0.02 logmar or 1 ETDRS letter
mixed	TRUE Elements will be converted one by one. Most plausibility checks will be overruled!

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, or the other alternative is selected with from_logmar = FALSE.

Value

vector of va class. See also "VA classes"

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 conversion 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)`.

Snellen "+/-" entries

By default, plus/minus entries are evaluated as intended by the test design: Snellen fractions increase/decrease only by lines.

```
- if entry -1 to +3 : take same Snellen value
- if <= -2 : take Snellen value one line below
- if >+3 (unlikely, but unfortunately not impossible):
```

If logmarstep = TRUE, each snellen optotype will be considered equivalent to 0.02 logmar or 1 ETDRS letter (assuming 5 letters in a row in a chart)

VA cleaning

For more details see `clean_va()`

1. `NA` is assigned to strings such as "." or "", "n/a" or " "
2. notation for qualitative entries is simplified.

VA classes

convert_VA returns a vector of three classes:

1. `va`
2. One of `snellen`, `logmar`, `etdrs` or `quali`.
3. Either of `character` (for Snellen and qualitative), `numeric` (for logMAR), or `integer` (for ETDRS).

See Also

Other VA converter: `snellen_steps`, `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., and Gregori et al..

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, logmarstep, ...)

## 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"
`logmarstep`	how plus/minus entries are evaluated. Default to increase/decrease snellen fractions by lines. If TRUE, each snellen optotype will be considered equivalent to 0.02 logmar or 1 ETDRS letter (assuming 5 letters in a row in a chart)

Details

VA can be snellen feet/meter/decimal, logMAR, ETDRS, or "qualitative" (Counting 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

Value

vector with visual acuity of class `va`. See also "VA classes"

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.

VA classes

convert_VA returns a vector of three classes:

1. `va`
2. One of `snellen`, `logmar`, `etdrs` or `quali`.
3. Either of `character` (for Snellen and qualitative), `numeric` (for logMAR), or `integer` (for ETDRS).

See Also

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

Index

∗Topic **datasets**
amd, 3
va_chart, 19

age, 2
amd, 3

blink, 4

clean_va, 6, *18*
clean_va(), *18*
convert_NLP *(*clean_va*)*, 6
convert_NLP(), *6*
convertVA *(*va_methods*)*, 20

eyes, *4, 5*, 7
eyes_to_string, *7, 8*
eyestr *(*eyes*)*, 7

getElem, *12*

hyperop, 9

isNAstring *(*clean_va*)*, 6
isNAstring(), *6*

myop, *4*, 10
myop(), *9*
myop_pivot, *11*
myop_rename, *11*
myopic *(*myop*)*, 10
myopizer, *11*

print.blink *(*print_methods*)*, 11
print.default, *12*
print.etdrs *(*print_methods*)*, 11
print.logmar *(*print_methods*)*, 11
print.snellen *(*print_methods*)*, 11
print_methods, 11

recodeye, 12
remCols, *5*
reveal, *4*, 13, *15*
reveal_methods, *13*, 14
reveal_split, *13*, *15*
revealEye *(*reveal_methods*)*, 14

set_codes, *10*, *12*
snellen_steps, 15, *18*, *21*
snellensteps *(*snellen_steps*)*, 15
sort_substr, *12*
sort_substr(), *11*
str_func_facs, *12*

tibble::tibble, *9*, *11*
tidyr::pivot_longer(), *9*
tidyr::pivot_wider(), *9*

va, *7*, *15*, 16, *19*, *21*
va(), *4*
va_chart, *17, 18*, 19, *21*
va_dissect, *15*, *17, 18*, *21*
va_dissect(), *18*
va_methods, *15*, *18, 19*, 20

which_va, *15*, *18*, *21*
which_va(), *17, 18*
which_va_dissect, *17*
which_va_dissect(), *18*