Package: eye (via r-universe)

September 14, 2024

Title Analysis of Eye Data

Version 1.2.1.9000

Description There is no ophthalmic researcher who has not had headaches from the handling of visual acuity entries. Different notations, untidy entries. This shall now be a matter of the past. Eye makes it as easy as pie to work with VA data - easy cleaning, easy conversion between Snellen, logMAR, ETDRS letters, and qualitative visual acuity shall never pester you again. The eye package automates the pesky task to count number of patients and eyes, and can help to clean data with easy re-coding for right and left eyes. It also contains functions to help reshaping eye side specific variables between wide and long format. 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

URL https://github.com/tjebo/eye

BugReports https://github.com/tjebo/eye/issues

Language en-US

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

VignetteBuilder knitr

Depends R (>= 4.1)

Imports cli (>= 3.0.1), dplyr (>= 1.0.7), english (>= 1.2-6), lubridate (>= 1.7.10), magrittr (>= 2.0.1), pillar (>= 1.6.2), purrr (>= 0.3.4), rlang (>= 0.4.11), stringr (>= 1.4.0), tibble (>= 3.1.3), tidyr (>= 1.1.3), tidyselect (>= 1.1.1)

Suggests eyedata (>= 0.1.0), knitr (>= 1.33), rmarkdown (>= 2.10), testthat (>= 3.0.4)

Repository https://tjebo.r-universe.dev

RemoteUrl https://github.com/tjebo/eye

RemoteRef HEAD

RemoteSha 35757117d2fbda4ce1953ed8a6789813d5f20b2b

Contents

`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

```
library(eyedata)
blink(amd2)

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 *Cleaning up Visual acuity entries*

Description

VA cleaning:

1. tidyNA: Replacing empty placeholders (".",""", "(any number of empty space)", "NULL", "NA", "N/A" , "-") - any cases - with NA
2. 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"), message = TRUE)

cleanVA(x, quali = c("nlp", "lp", "hm", "cf"), message = TRUE)
```

Arguments

x	Vector with VA entries
quali	strings for qualitative visual acuity entries
message	message for replaced NA values

Value

character vector

See Also

Other VA cleaner: va()

eyes *Count patients and eyes*

Description

Counts number of subjects and right and left eyes. Columns are guessed.

Usage

```
eyes(x, id_col = NULL, eye_col = NULL, dropunknown = TRUE, details = FALSE)

eyestr(x, ..., english = "small", caps = FALSE)
```

Arguments

x	required. (data frame)
id_col	Subject identifying column, passed as (quoted) character Can also be abbreviated to "id" as per partial matching
eye_col	Eye identifying column, passed as (quoted) character. Can also be abbreviated to "eye" as per partial matching
dropunknown	introduces NA for values not recognized by recodeye
details	if TRUE, will add information about which and how many subjects have only one eye or both included, and provide a list of subject IDs for each
...	passed to eyes
english	Which numbers to be written in plain english: choose "small" for numbers till 12, "all" (all numbers), or "none" (or any other string!) for none
caps	if TRUE, first number will have capital first letter

Value

List (of class "eyes" with count of patients and eyes if "details = TRUE", an list of class "eyes_details" will be returned

eyestr: Character string - can be directly pasted into reports

Column guessing

id_col and **eye_col** arguments overrule the name guessing for the respective columns (here, cases need to match). Both arguments can be abbreviated (*id* or *eye*) as per partial argument name matching.

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

patient ID columns:

- First, eyes is looking for names that contain both strings "pat" and "id" (the order doesn't matter) - you can change this with set_eye_strings
- 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 column:

- eyes primarily looks for columns called either "eye" or "eyes", (you can change this with set_eye_strings) and if they are not present, columns containing string "eye" (e.g., EyeName will be recognized)

Eye coding

The following codes are recognized: (change this with set_eye_strings)

- integer coding 0:1 and 1:2, right eye being the lower number.
- right eyes: c("r", "re", "od", "right") and

- left eyes: c("l", "le", "os", "left") and
- both eyes: c("b", "both", "ou")

If your eye column contains other values, they will be dropped to NA (dropunknown) or kept (and then only patients will be counted, because coding remains unclear). Recommend then to recode with recodeye

eyestr

eyestr creates a string which can be pasted into reports. It currently only supports "x eyes of n patient(s)" This is a limitation, but I guess in the vast majority of cases will be "correct". To use for other categories (e.g., "people" or "participants"), use `eyes(...)[1]`

Examples

```
library(eyedata)
eyes(amd2)

## If you code your eyes with different strings,
## e.g., because you are using a different language,
## you can change this either with `set_eye_strings`
set_eye_strings(right = c("droit", "od"), left = c("gauche", "og"))

## restore defaults with
set_eye_strings()
# Examples for the usage of eyestr
eyestr(amd2)

set.seed(1)
ls_dat <-
  lapply(c(1, 12, 13),
    function(x) data.frame(id = as.character(1:x),
                           eye = sample(c("r", "l"), x, replace = TRUE)))

lapply(ls_dat, eyestr, english = "small")
lapply(ls_dat, eyestr, english = "all")
lapply(ls_dat, eyestr, english = "all", caps = TRUE)
lapply(ls_dat, eyestr, english = "none")
lapply(ls_dat, eyestr, english = "none")
```

`getage` *getage*

Description

calculates age in years, as durations or periods

Usage

```
getage(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

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

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

---

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 eye_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")
```

---

parse_snellen *parsing snellen fractions to numeric values*

Description

parsing snellen fractions to numeric values

Usage

```
parse_snellen(y)
```

Arguments

y	vector

---

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 class blink

S3 methods for class eyes

S3 methods for class eyes_details

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, ...)

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

## S3 method for class 'eyes_details'
print(x, show = 6, ...)
```

Arguments

x	object of class "eyes_details"
...	arguments passed to print.default
show	how many subjects to be shown before printing the footnote

Value

No return value, called for side effects (printing)

`recodeye` *Recode eyes*

Description

recoding eyes to "r" and "l"

Usage

```
recodeye(x, to = NULL, eyestrings = NULL, dropunknown = TRUE)
```

Arguments

x	vector of strings
to	named vector to which eye codes. If unnamed, this order: c(r, l, b)
eyestrings	named list of substrings which should be converted to right and left eyes - if passed unnamed, this order: list(r, l, b)
dropunknown	introduces NA for values that are not part of eyestrings

Value

Character vector

string detection

recodeye will automatically detect the following strings: right = c("r", "re", "od", "right"), left = c("l", "le", "os", "left"), both = c("b","both","ou")

You can change this with set_eye_strings

to and eyecode arguments

If passed, should ideally be of same length, and have the respective eyes at the same index (or with the same name!). If the lengths are not equal, e.g., if only "to" is passed with n elements, the shorter argument will be will be cut down to the first n elements of the longer argument.

Note that all unique strings which are part of the column should be contained in the "eyecode" argument.

numeric coding

Currently numeric coding only accepts binary coding (right and left eye). In order to use numeric coding for "both eyes" as well, a workaround using the eyestrings argument is suggested.

See Also

Other string matching functions: getElem, sort_substr(), str_search

Examples

```
x <- c("r", "re", "od", "right", "l", "le", "os", "left", "both", "ou")
recodeye(x)

## chose the resulting codes
recodeye(x, to = c("od", "os", "ou"))

x <- 1:2
recodeye(x)

## If you code your eyes with different strings,
## e.g., because you are using a different language,
## you can change this either with the eyestrings argument,
french <- c("OD", "droit", "gauche", "OG")
recodeye(french, eyestrings = list(r = c("droit", "od"), l = c("gauche", "og")))

## or change it more globally with `set_eye_strings`
set_eye_strings(right = c("droit", "od"), left = c("gauche", "og"))
recodeye(french)

## restore defaults with
set_eye_strings()
```

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()`

`set_eye_strings` *Set list of codes*

Description

This sets the list of codes used throughout the eye package for the coding of all kind of stuff. If you want to change recognized codes, this is the place to do it. See examples below how to easily overwrite it. It is important that you must pass them as a character vector!

cases are always ignored, so you don't need to worry about this bit.

Usage

```
set_eye_strings(
  right = c("r", "re", "od", "right"),
  left = c("l", "le", "os", "left"),
  both = c("b", "both", "ou"),
  iop = c("iop", "gat", "nct"),
  iop_partial = c("pressure"),
  va = c("va", "bcva"),
  va_method = c("etdrs", "snellen", "logmar"),
  va_partial = c("acuit"),
  id = c("pat", "id"),
  eye = c("eye", "eyes"),
  quali = c("nlp", "lp", "hm", "cf"),
  ...
)
```

Arguments

`right`	right eyes
`left`	left eyes
`both`	both eyes
`iop`	IOP codes
`iop_partial`	partial strings used to find IOP columns
`va`	VA codes
`va_method`	VA methods (used to recognize VA columns - when those strings occur "fully", i.e., not as part of sth else)
`va_partial`	Also used to find VA columns - looking for partial strings
`id`	patient column codes
`eye`	eye column codes
`quali`	quali VA codes
`...`	currently not used, but might be needed in the future

Examples

```
# To expand recognized codes for eyes, e.g. if you want to use French names
set_eye_strings(right = c("droit", "od"), left = c("gauche", "og"))

# To restore the defaults, simply call set_eye_strings empty
set_eye_strings()
```

`snellen_steps` *Convert plus minus entries*

Description

Convert plus minus entries

Usage

```
snellensteps(x, smallstep)
```

Arguments

`x`	Vector with VA entries of class snellen - needs to be in format xx/yy
`smallstep`	if plusminus shall be considered as logmar equivalent

Value

character vector of Snellen entries

See Also

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

Other VA converter: `VAwrapper`, `plausibility_methods`, `va_methods`, `va_mixed()`, `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.

Usage

```
va(x, from = NULL, to = NULL, type = "ft", smallstep = FALSE, noplus = FALSE)
```

Arguments

x	Vector with visual acuity entries. Must be atomic. Snellen fractions need to be entered with "/"
from	will force to evaluate from which notation to convert - Must be "etdrs", "logmar", "snellen" or "snellendec". Ignored if the value should not be plausible.
to	To which class to convert. "etdrs", "logmar" or "snellen" - any case allowed. If NULL (default), will simply "clean up" VA entries. This may then result in a vector of "mixed" VA notations.
type	To which Snellen notation to convert: "m", "dec" or "ft"
smallstep	how +/- entries are evaluated. FALSE: increase/decrease Snellen fractions by lines. TRUE: plus/minus entries equivalent to 0.02 logmar
noplus	ignoring plus/minus entries and just returning the snellen fraction. This overrides the smallstep argument.

Value

vector of va class. See also "VA classes"

VA conversion

- **logMAR to ETDRS**: logMAR rounded to the first digit and converted with the visual acuity chart (see section VA chart)
- **Snellen to logMAR**: logMAR = -1 * log10(snellen_frac)
- **Snellen to ETDRS**: ETDRS = 85 + 50 * log10(snellen_frac) doi:10.1097/iae.0b013e3181d87e04
- **ETDRS to logMAR**: logMAR = -0.02 * etdrs + 1.7 Beck et al. doi:10.1016/s00029394(02)01825-1
- **Hand movements and counting fingers** are converted following Schulze-Bonsel et al. - https://doi.org/10.1167/iovs.05-0981
- **(No) light perception** are converted following the suggestions by Michael Bach

Qualitative visual acuity entries

In order to calculate with qualitative entries counting fingers, hand movement and (no) perception of light, **use logMAR** ! Qualitative visual acuity lower than counting fingers is assigned 0 ETDRS letter, in order to keep it as a measurement (not: NA). It is very difficult to justify a "negative" letter score in a test which only has a specific range (0-100).

- **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.

VA chart

You can find with eye:::va_chart. This chart and VA conversion formulas are based on charts in Holladay et al.doi:10.1016/j.jcrs.2004.01.014, Beck et al. doi:10.1016/s00029394(02)018251Beck et al., and Gregori et al.doi:10.1097/iae.0b013e3181d87e04. The etdrs values for NLP and PL are deliberately set at those values because they are unlikely to happen by chance as a wrong entry (and as integers), and it has internal reasons that make conversion easier.

Accepted VA formats / Plausibility checks

- Snellen fractions (meter/ feet) need to be entered as fraction with "/". Any fractions allowed. You can get creative with your snellens. see **"Examples"**
- ETDRS must be integer-equivalent between 0 and 100 (integer equivalent means, it can also be a character vector)
- logMAR must be -0.3 <= x <= 3.0
- Snellen decimal must be 0 < x <= 2
- Qualitative must be either of PL, LP, NLP, NPL, HM, CF (any case allowed)
- Plausibility checks are performed for the automatically or manually defined notation.
- Any element which is implausible/ not recognized will be converted to NA

Entries with mixed VA notations

Use va_mixed instead.

Snellen "+/-" entries

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

```
- if entry -2 to +2 : take same Snellen value
- if < -2 : take Snellen value one line below
- if > +2 : take Snellen value one line above
```

If smallstep = TRUE, each snellen optotype will be considered equivalent to 0.02 logmar (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

convertVA returns a vector of three classes:

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

See Also

Other Ophthalmic functions: va_mixed()

Other VA converter: VAwrapper, plausibility_methods, snellen_steps, va_methods, va_mixed(), 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")
```

VAwrapper *VA conversion wrapper*

Description

Simple convenience wrapper around va to get desired VA class

Usage

```
to_logmar(x, ...)

to_etdrs(x, ...)

to_snellen(x, ...)

as_logmar(x, ...)
```

```
as_etdrs(x, ...)

as_snellen(x, ...)
```

Arguments

x	vector of visual acuities
...	parameters passed to va

Value

vector with visual acuity of class as_(...) or to_(...) See also convertVA: "VA classes"

VA conversion

For details see va and convertVA

See Also

Other VA converter: `plausibility_methods`, `snellen_steps`, `va_methods`, `va_mixed()`, `va()`, `which_va()`

Examples

```
x <- c(23, 56, 74, 58) ## ETDRS letters
to_logmar(x)
to_snellen(x)
to_snellen(x, type = "dec")

x <- c("NLP", "0.8", "34", "3/60", "2/200", "20/50", "  ", ".", "-", "NULL")
to_snellen(x, from = "snellendec")
to_snellen(x, from = "etdrs")
to_snellen(x, from = "logmar")
```

---

va_mixed *VA classes*

Description

va_mixed is a wrapper around va on all possible VA notations. By default, c("snellen", "etdrs", "logmar", "snellendec") will be converted - in that order! For tricky cases see details and examples. Note that va_mixed will not give nice messages which values are transformed from which notation, and which values were replaced with NA.

Usage

```
va_mixed(x, to, possible)
```

Arguments

x	vector with mixed VA entries
to	to which notation to be converted
possible	which possible VA notations - and the precedence given, see details

Details

Mixed entries are challenging, but unfortunately seem to occur in real life data. It will be fairly individual what you have in yours, but it should hopefully not happen that you have *all* possible notations. Snellen fractions are usually not challenging because they contain a "/", thus are easy to recognize.

Most problematic are values between 0 and 3, in particular full integers - this can be EDTRS, snellen decimal notation or logmar. If your data doesn't have snellen decimal notation, specify this with "possible", e.g. with `possible = c("snellen", "etdrs", "logmar")`. If you know that you don't have any ETDRS value less than 4, you can safely give precedence to logmar instead, like this: `possible = c("snellen", "logmar", "etdrs")` @examples

awfully mixed notation!! (and note the wrong -1 value)

x <- c(NA, "nlp", 1:2, 1.1, -1, "20/40", "4/6", "6/1000", 34) va_mixed(x, to = "snellen")

"I only have snellen and snellen decimal notation in my data"

va_mixed(x, to = "snellen", possible = c("snellen", "snellendec"))

"I have snellen, logmar and etdrs in my data, and there is no etdrs value

less than 4" va_mixed(x, to = "snellen", possible = c("snellen", "logmar", "etdrs"))

See Also

Other Ophthalmic functions: `va()`

Other VA converter: `VAwrapper`, `plausibility_methods`, `snellen_steps`, `va_methods`, `va()`, `which_va()`

Index