Package ‘VWPre’

January 20, 2025

Type Package

Title Tools for Preprocessing Visual World Data

Version 1.2.4

Date 2020-11-28

Author Vincent Porretta [aut, cre],
Aki-Juhani Kyröläinen [aut],
Jacolien van Rij [ctb],
Juhani Järvikivi [ctb]

Maintainer Vincent Porretta <vincentporretta@gmail.com>

Description Gaze data from the Visual World Paradigm requires significant preprocessing prior to plotting and analyzing the data. This package provides functions for preparing visual world eye-tracking data for statistical analysis and plotting. It can prepare data for linear analyses (e.g., ANOVA, Gaussian-family LMER, Gaussian-family GAMM) as well as logistic analyses (e.g., binomial-family LMER and binomial-family GAMM). Additionally, it contains various plotting functions for creating grand average and conditional average plots. See the vignette for samples of the functionality. Currently, the functions in this package are designed for handling data collected with SR Research Eyelink eye trackers using Sample Reports created in SR Research Data Viewer. While we would like to add functionality for data collected with other systems in the future, the current package is considered to be feature-complete; further updates will mainly entail maintenance and the addition of minor functionality.

Depends R (>= 3.5.0), dplyr (>= 0.7.0)

Imports rlang (>= 0.1.1), ggplot2 (>= 2.2.0), mgcv (>= 1.8-16), shiny (>= 0.14.2), tidyr (>= 0.6.0), stats (>= 3.3.2)

License GPL-3

LazyData true

Suggests knitr, rmarkdown, gridExtra, itsadug

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.1.0

NeedsCompilation no

Repository CRAN

Date/Publication 2020-11-29 17:10:02 UTC

Contents

`.check_for_PupilPre` *Internal helper function, not intended to be called externally.*

Description

Internal helper function, not intended to be called externally.

Usage

```
.check_for_PupilPre(type, suggest)
```

Arguments

`type`	A string, "NotAvailable" of "UseOther".
`suggest`	A string, PupilPre function name.

Value

Text feedback and instruction.

`align_msg` *Aligns samples to a specific message.*

Description

`align_msg` examines the data from each recording event and locates the first instance of the specified message in the column SAMPLE_MESSAGE. The function creates a new column containing the aligned series which can be utilized by subsequent functions for checking and creating the time series column.

Usage

```
align_msg(data, Msg = NULL)
```

Arguments

`data`	A data table object output from `prep_data`.
`Msg`	An obligatory string containing the message to be found in the column SAMPLE_MESSAGE or a regular expression for locating the appropriate message.

Value

A data table object.

Examples

```
## Not run:
# To align the samples to a specific message...
library(VWPre)
df <- align_msg(data = dat, Msg = "ExperimentDisplay")

# For a more complete tutorial on VWPre plotting functions:
vignette("SR_Message_Alignment", package="VWPre")

## End(Not run)
```

---

bin_prop *Bins the sample data and calculates proportion looks by interest area*

Description

bin_prop calculates the proportion of looks (samples) to each interest area in a particular window of time (bin size). This function first checks to see if the procedure is possible given the sampling rate and desired bin size. It then performs the calculation and downsampling, returning new columns corresponding to each interest area ID (e.g., 'IA_1_C', 'IA_1_P'). The extention '_C' indicates the count of samples in the bin and the extension '_P' indicates the proportion. N.B.: This function will work for data with a maximum of 8 interest areas.

Usage

```
bin_prop(data, NoIA = NULL, BinSize = NULL, SamplingRate = NULL)
```

Arguments

data	A data table object output by select_recorded_eye or check_samplingrate.
NoIA	A positive integer indicating the number of interest areas defined when creating the study.
BinSize	A positive integer indicating the size of the binning window (in milliseconds).
SamplingRate	A positive integer indicating the sampling rate (in Hertz) used to record the gaze data, which can be determined with the function check_samplingrate.

Value

A data table with additional columns (the number of which depends on the number of interest areas specified) added to data.

Examples

```
## Not run:
library(VWPre)
# Bin samples and calculation proportions...
df <- bin_prop(dat, NoIA = 4, BinSize = 20, SamplingRate = 1000)

## End(Not run)
```

---

check_all_msgs *Output all messages with timestamps*

Description

check_all_msgs outputs all sample messages present in the data. Optionally, the result can be output to a dataframe containing all event-level information.

Usage

```
check_all_msgs(data, ReturnData = FALSE)
```

Arguments

data	A data table object output by prep_data, align_msg, or create_time_series.
ReturnData	A logical indicating whether to return a data table containing Message Time information for each event.

Value

All Sample Messages found in the data.

Examples

```
## Not run:
library(VWPre)
# Check all messages in the data...
check_all_msgs(data = dat)

## End(Not run)
```

`check_eye_recording` *Check which eyes were recorded during the experiment*

Description

`check_eye_recording` quickly checks which eyes contain gaze data either using the EYE_TRACKED column (if available) or the Right and Left interest area columns. It prints a summary and suggests which setting to use for the `Recording` parameter in the function `select_recorded_eye`.

Usage

```
check_eye_recording(data)
```

Arguments

`data`	A data table object output by `create_time_series`.

Value

Text feedback and instruction.

Examples

```
## Not run:
library(VWPre)
# Create a unified columns for the gaze data...
check_eye_recording(dat)

## End(Not run)
```

`check_ia` *Check the interest area IDs and labels*

Description

`check_ia` examines both the interest area IDs and interest area labels (and their mapping) for both eyes. It returns a summary of the information and will stop if the data are not consistent with the requirements for further processing.

Usage

```
check_ia(data)
```

Arguments

`data`	A data table object output by `relabel_na`.

Value

The value(s) and label(s) of interest areas and how they map for each eye.

Examples

```
## Not run:
library(VWPre)
# Check the interest area information...
check_ia(dat)

## End(Not run)
```

`check_msg_time` *Check the time value(s) at a specific message*

Description

`check_msg_time` examines the time point of a specific Sample Message for each event. Depending on the format of the data, it will use one of three columns: TIMESTAMP, Align, or Time. If times at which the message occurs are equal, it will return a single value. Optionally, the result can be output to a dataframe containing all event-level information.

Usage

```
check_msg_time(data, Msg = NULL, ReturnData = FALSE)
```

Arguments

`data`	A data table object output by `relabel_na`, `align_msg`, or `create_time_series`.
`Msg`	A character string containing the exact message to be found in the column SAMPLE_MESSAGE or a regular expression for locating the appropriate message.
`ReturnData`	A logical indicating whether to return a data table containing Message Time information for each event.

Value

The value(s) of Time (in milliseconds) at which the Sample Message is found.

Examples

```
## Not run:
library(VWPre)
# Check the Sample Message time...
check_msg_time(data = dat)

## End(Not run)
```

check_samples_per_bin *Check the number of samples in each bin*

Description

check_samples_per_bin determines the number of samples in each bin produced by bin_prop. This function may be helpful for determining the obligatory parameter ‘ObsPerBin‘ which is input to transform_to_elogit.

Usage

```
check_samples_per_bin(data)
```

Arguments

data	A data table object output by bin_prop.

Value

A printed summary of the number of samples in each bin.

Examples

```
## Not run:
library(VWPre)
# Determine the number of samples per bin...
check_samples_per_bin(dat)

## End(Not run)
```

check_samplingrate *Determine the sampling rate present in the data*

Description

check_samplingrate determines the sampling rate in the data. This function is helpful for determining the obligatory parameter input to bin_prop. If different sampling rates were used, the function adds a sampling rate column, which can be used to subset the data for further processing.

Usage

```
check_samplingrate(data, ReturnData = FALSE)
```

Arguments

data	A data table object output by select_recorded_eye.
ReturnData	A logical indicating whether to return a data table containing a new column called SamplingRate

Value

A printed summary and/or a data table object

Examples

```
## Not run:
library(VWPre)
# Determine the sampling rate...
check_samplingrate(dat)

## End(Not run)
```

`check_time_series` *Check the new time series*

Description

`check_time_series` examines the first value in the Time column for each event. If they are equal, it will return a single value. The returned value(s) will vary depending on the interest period (if defined), message alignment (if completed), and the Adjustment parameter ('Adj') supplied to `create_time_series`. Optionally, the result can be output to a dataframe containing all event-level information.

Usage

```
check_time_series(data, ReturnData = FALSE)
```

Arguments

`data`	A data table object output by `create_time_series`.
`ReturnData`	A logical indicating whether to return a data table containing Start Time information for each event.

Value

The value(s) of Time (in milliseconds) at which events begin relative to the onset of the auditory stimulus.

Examples

```
## Not run:
library(VWPre)
# Check the starting Time column...
check_time_series(data = dat)

## End(Not run)
```

`create_binomial` *Creates a success/failure column for each IA based on counts.*

Description

`create_binomial` uses interest area count columns to create a success/failure column for each IA which is suitable for logistic regression. N.B.: This function will work for data with a maximum of 8 interest areas.

Usage

```
create_binomial(
  data,
  NoIA = NULL,
  ObsPerBin = NULL,
  ObsOverride = FALSE,
  CustomBinom = NULL
)
```

Arguments

`data`	A data table object output by either `bin_prop` or `transform_to_elogit`.
`NoIA`	A positive integer indicating the number of interest areas defined when creating the study.
`ObsPerBin`	A positive integer indicating the number of observations to use in the calculation. Typically, this will be the number of samples per bin, which can be determined with `check_samples_per_bin`.
`ObsOverride`	A logical value controlling restrictions on the value provided to ObsPerBin. Default value is FALSE.
`CustomBinom`	An optional parameter specifying a vector containing two integers corresponding to the interest area IDs to be combined.

Value

A data table with additional columns (the number of which depends on the number of interest areas specified) added to `data`.

Examples

```
## Not run:
library(VWPre)
# Create binomial (success/failure) column...
df <- create_binomial(data = dat, NoIA = 4, ObsPerBin = 20)

## End(Not run)
```

`create_time_series` *Create a time series column*

Description

`create_time_series` standardizes the starting point for each event, creates a time series for each event including the offset for the amount of time prior to (or after) the zero point. The time series is indicated in a new column called Time.

Usage

```
create_time_series(data, Adjust = 0)
```

Arguments

`data`	A data table object output by `relabel_na` or `align_msg`.
`Adjust`	Optionally an integer value or a text string. If an integer (positive or negative), this will indicate an amount of time in milliseconds. The value is subtracted from the time points: positive values shift the zero forward; negative values shift the zero backward. If a text string, this will be the name of a column in the data set which contains values indicating when the critical stimulus was presented relative to the zero point.

Value

A data table object.

Examples

```
## Not run:
library(VWPre)
# To create the Time column...
df <- create_time_series(data = dat, Adjust = "SoundOnsetColumn")
# or
df <- create_time_series(data = dat, Adjust = -100)
# or
df <- create_time_series(data = dat, Adjust = 100)

## End(Not run)
```

`custom_ia` *Map gaze data to newly defined interest areas*

Description

`custom_ia` uses a lookup data frame to map Left and Right gaze data to newly defined/supplied interest areas for each recording event. The lookup data should contain columns Event, IA_LABEL, IA_ID, Top, Bottom, Left, Right, which specify the Interest area label, its corresponding ID, and the boundaries (in pixel values) for each recording event. The function will overwrite existing columns RIGHT_INTEREST_AREA_LABEL, RIGHT_INTEREST_AREA_ID, LEFT_INTEREST_AREA_LABEL, and LEFT_INTEREST_AREA_ID.

Usage

```
custom_ia(data, iaLookup = NULL)
```

Arguments

`data`	A data table object output by `prep_data`.
`iaLookup`	A data frame object containing by-event mapping information.

Value

A data table object.

Examples

```
## Not run:
library(VWPre)
# Map gaze data to newly defined interest areas...
df <- custom_ia(data = dat, iaLookup = LookUpDF)

# For a more complete tutorial on VWPre plotting functions:
vignette("SR_Interest_Areas", package="VWPre")

## End(Not run)
```

`ds_options` *Determine downsampling options based on current sampling rate*

Description

`ds_options` determines the possible rates to which the current sampling rate can downsampled. It then prints the options in both bin size (milliseconds) and corresponding sampling rate (Hertz).

Usage

```
ds_options(SamplingRate, OutputRates = "Suggested")
```

Arguments

`SamplingRate`	A positive integer indicating the sampling rate (in Hertz) used to record the gaze data, which can be determined with the function `check_samplingrate`.
`OutputRates`	A string ("Suggested" or "All") controlling if all rates are output, or if only whole rates (default) are output.

Value

A printed summary of options (bin size and rate) for downsampling.

Examples

```
## Not run:
library(VWPre)
# Determine downsampling options...
ds_options(SamplingRate = 1000)

## End(Not run)
```

`fasttrack` *Fast-track basic preprocessing*

Description

`fasttrack` is a meta-function for advanced users who are already familiar with the package functions and do not need to take remedial actions such as recoding interest areas, remapping gaze data, or performing message alignment.It takes all necessary arguments for the component functions to produce proportion looks and can output either empirical logits or binomial data. The function returns a dataframe containing the result of the series of subroutines.

Usage

```
fasttrack(
  data = data,
  Subject = NULL,
  Item = NA,
  EventColumns = c("Subject", "TRIAL_INDEX"),
  NoIA = NoIA,
  Adjust = 0,
  Recording = NULL,
  WhenLandR = NA,
  BinSize = NULL,
  SamplingRate = NULL,
  ObsPerBin = NULL,
  ObsOverride = FALSE,
  Constant = 0.5,
  CustomBinom = NULL,
  Output = NULL
)
```

Arguments

`data`	A data frame object created from an Eyelink Sample Report.
`Subject`	An obligatory string containing the column name corresponding to the subject identifier.
`Item`	An optional string containing the column name corresponding to the item identifier; by default, NA.
`EventColumns`	A vector specifying the columns which will be used for creating the Event variable; by default, Subject and TRIAL_INDEX.
`NoIA`	A positive integer indicating the number of interest areas defined when creating the study.
`Adjust`	An integer indicating amount of time in milliseconds by which to offset the time series.
`Recording`	A string indicating which eyes were used for recording gaze data.
`WhenLandR`	A string indicating which eye ("Right" or "Left) to use if gaze data is available for both eyes (i.e., Recording = "LandR").
`BinSize`	A positive integer indicating the size of the binning window (in milliseconds).
`SamplingRate`	A positive integer indicating the sampling rate (in Hertz) used to record the gaze data.
`ObsPerBin`	A positive integer indicating the desired number of observations to be used in the calculations.
`ObsOverride`	A logical value controlling restrictions on the value provided to ObsPerBin. Default value is FALSE.
`Constant`	A positive number used for the empirical logit and weights calculation; by default, 0.5 as in Barr (2008).
`CustomBinom`	An optional parameter specifying a vector containing two integers corresponding to the interest area IDs to be combined.
`Output`	An obligatory string containing either "ELogit" or "Binomial".

Value

A data table containing formatting and calculations.

Examples

```
## Not run:
library(VWPre)
# Perform meta-function on data
```

```
df <- fasttrack(data = dat, Subject = "RECORDING_SESSION_LABEL", Item = "itemid",
       EventColumns = c("Subject", "TRIAL_INDEX"), NoIA = 4, Adjust = 100,
Recording = "LandR", WhenLandR = "Right", BinSize = 20,
SamplingRate = 1000, ObsPerBin = 20, Constant = 0.5,
Output = "ELogit")

## End(Not run)
```

`make_pelogit_fnc` *Create function for back-transforming empirical logits to proportions*

Description

`make_pelogit_fnc` creates a function that can transform empirical logit values back to probability scale using the number of samples and constant that were used in the original transformation. This function can then be use to backtransform value predicted by a statistical model.

Usage

```
make_pelogit_fnc(ObsPerBin = NULL, Constant = NULL)
```

Arguments

`ObsPerBin`	A positive integer indicating the number of observations used in the original transformation calculation.
`Constant`	A positive number used in the original transformation calculation.

Value

A function.

Examples

```
## Not run:
library(VWPre)
# Make backtransformation function
pelogit <- make_pelogit_fnc(20, 0.5)

## End(Not run)
```

mark_trackloss *Mark trackloss by blink and/or screen size*

Description

mark_trackloss marks data points related to trackloss for those in blink, off-screen, or both.

Usage

```
mark_trackloss(data, Type = NULL, ScreenSize = NULL)
```

Arguments

data	A data table object output by select_recorded_eye.
Type	A string indicating "Blink", "OffScreen", or "Both".
ScreenSize	A numeric vector specifying (in pixels) the dimensions of the x and y of the screen used during the experiment.

Value

An object of type data table as described in tibble.

Examples

```
## Not run:
library(VWPre)
# Mark trackloss...
df <- mark_trackloss(data = dat, Type = "Both", ScreenSize = c(1920, 1080))

## End(Not run)
```

plot_avg *Plots average looks to interest areas.*

Description

plot_avg calculates the grand or conditional averages of looks to each interest area along with standard error. It then plots the results. N.B.: This function will work for data with a maximum of 8 interest areas and 2 conditions.

Usage

```
plot_avg(
  data,
  type = NULL,
  xlim = NA,
  IAColumns = NULL,
  Averaging = "Event",
  Condition1 = NULL,
  Condition2 = NULL,
  Cond1Labels = NA,
  Cond2Labels = NA,
  ErrorBar = TRUE,
  VWPreTheme = TRUE,
  ConfLev = 95,
  CItype = "simultaneous",
  ErrorBand = FALSE,
  ErrorType = "SE"
)
```

Arguments

`data`	A data table object output by either `bin_prop`. `transform_to_elogit`, or `create_binomial`.
`type`	A character string indicating "proportion" or "elogit" which influences how standard error and confidence intervals are calculated.
`xlim`	A vector of two integers specifying the limits of the x-axis.
`IAColumns`	A named character vector specifying the desired interest area columns with custom strings for the legend.
`Averaging`	A character string indicating how the averaging should be done. "Event" (default) will produce the overall mean in the data, while "Subject" or "Item" (or, in principle, any other column name) will calculate the grand mean by that factor.
`Condition1`	A string containing the column name corresponding to the first condition, if available.
`Condition2`	A string containing the column name corresponding to the second condition, if available.
`Cond1Labels`	A named character vector specifying the desired custom labels of the levels of the first condition.
`Cond2Labels`	A named character vector specifying the desired custom labels of the levels of the second condition.
`ErrorBar`	A logical indicating whether error bars should be included in the plot.
`VWPreTheme`	A logical indicating whether the theme included with the function should be applied, or ggplot2's base theme (to which any other custom theme could be added).
`ConfLev`	A number indicating the confidence level of the CI.

CItype	A string indicating "simultaneous" or "pointwise". Simultaneous performs a Bonferroni correction for the interval.
ErrorBand	A logical indicating whether error bands should be included in the plot.
ErrorType	A string indicating "SE" or "CI". For SE, the calculation varies for empirical logits and proportions. Further, for CI, the calculation on proportions uses the Wald method.

Examples

```
## Not run:
library(VWPre)
# For plotting the grand average with the included theme and SE bars
plot_avg(data = dat, type = "elogit", xlim = c(0, 1000),
   IAColumns = c(IA_1_ELogit = "Target", IA_2_ELogit = "Rhyme",
   IA_3_ELogit = "OnsetComp", IA_4_ELogit = "Distractor"),
   Averaging = "Event", Condition1 = NA, Condition2 = NA,
   Cond1Labels = NA, Cond2Labels = NA,
   ErrorBar = TRUE, VWPreTheme = TRUE, ErrorType = "SE",
   ErrorBand = FALSE)

# For plotting conditional averages (one condition) with the included theme
# and 95% simultaneous CI bars.
# This produces plots arranged horizontally
plot_avg(data = dat, type = "elogit", xlim = c(0, 1000),
   IAColumns = c(IA_1_ELogit = "Target", IA_2_ELogit = "Rhyme",
   IA_3_ELogit = "OnsetComp", IA_4_ELogit = "Distractor"),
   Averaging = "Event", Condition1 = NA, Condition2 = "talker",
   Cond1Labels = NA,
   Cond2Labels = c(CH1 = "Chinese 1", CH10 = "Chinese 3", CH9 = "Chinese 2",
   EN3 = "English 1"), ErrorBar = TRUE, VWPreTheme = TRUE,
   ErrorBand = FALSE, ErrorType = "CI", ConfLev = 95, CItype = "simultaneous")

# For plotting conditional averages (two conditions) for one interest area
with the included theme and 95% simultaneous CI bands.
# This produces plots arranged in grid format.
plot_avg(data = dat, type = "elogit", xlim = c(0, 1000),
   IAColumns = c(IA_1_ELogit = "Target"), Averaging = "Event",
   Condition1 = "talker", Condition2 = "Exp",
   Cond1Labels = c(CH1 = "Chinese 1", CH10 = "Chinese 3", CH9 = "Chinese 2",
   EN3 = "English 1"), Cond2Labels = c(High = "H Exp", Low = "L Exp"),
   ErrorBar = FALSE, VWPreTheme = TRUE, ErrorBand = TRUE,
   ErrorType = "CI", ConfLev = 95, CItype = "simultaneous")

#' # For a more complete tutorial on VWPre plotting functions:
vignette("SR_Plotting", package="VWPre")

## End(Not run)
```

`plot_avg_cdiff` *Plots average difference between two conditions.*

Description

`plot_avg_cdiff` calculates the average of differences between two specified conditions along with standard error and then plots the results.

Usage

```
plot_avg_cdiff(
  data,
  IAColumn = NULL,
  xlim = NA,
  type = NULL,
  Averaging = "Subject",
  Condition = NULL,
  CondLabels = NA,
  ErrorBar = TRUE,
  VWPreTheme = TRUE,
  ConfLev = 95,
  CItype = "simultaneous",
  ErrorBand = FALSE,
  ErrorType = "SE"
)
```

Arguments

`data`	A data table object output by either `bin_prop`. `transform_to_elogit`, or `create_binomial`.
`IAColumn`	A character vector specifying the desired column corresponding to the interest area.
`xlim`	A vector of two integers specifying the limits of the x-axis.
`type`	A character string indicating "proportion" or "elogit", which influences how standard error and confidence intervals are calculated.
`Averaging`	A character string indicating how the averaging should be done. "Subject" (default) will produce the grand mean in the data, while "Item" (or, in principle, any other column name) will calculate the grand mean by that factor.
`Condition`	A list containing the column name corresponding to the condition and factor levels to be used for calculating the difference.
`CondLabels`	A named character vector specifying the desired labels of the levels of the condition.
`ErrorBar`	A logical indicating whether error bars should be included in the plot.

`VWPreTheme`	A logical indicating whether the theme included with the function should be applied, or ggplot2's base theme (to which any other custom theme could be added).
`ConfLev`	A number indicating the confidence level of the CI.
`CItype`	A string indicating "simultaneous" or "pointwise". Simultaneous performs a Bonferroni correction for the interval.
`ErrorBand`	A logical indicating whether error bands should be included in the plot.
`ErrorType`	A string indicating "SE" or "CI".

Examples

```
## Not run:
library(VWPre)
# For plotting average difference between conditions...
plot_avg_cdiff(data = dat, xlim = c(0, 1000), type = "proportion",
             IAColumn = "IA_1_P", Condition = list(talker = c("EN3", "CH1")),
             CondLabels = NA, ErrorBar = TRUE, VWPreTheme = TRUE,
             ErrorBand = FALSE, ErrorType = "SE")

# For a more complete tutorial on VWPre plotting functions:
vignette("SR_Plotting", package="VWPre")

## End(Not run)
```

`plot_avg_contour` *Plots average contour surface of looks to a given interest area.*

Description

`plot_avg_contour` calculates the conditional average of proportions or empirical logit looks to a given interest area by Time and a specified continuous variable. It then applies a 3D smooth (derived using `gam`) over the surface and plots the results as a contour plot.

Usage

```
plot_avg_contour(
  data,
  IA = NULL,
  type = NULL,
  Var = NULL,
  Averaging = "Event",
  VarLabel = NULL,
  xlim = NA,
  VWPreTheme = TRUE,
  Colors = c("gray20", "gray90")
)
```

Arguments

`data`	A data table object output by either `bin_prop`. `transform_to_elogit`, or `create_binomial`.
`IA`	A string specifying the column name of the IA to use.
`type`	A character string indicating "proportion" or "elogit".
`Var`	A string containing the column name corresponding to the continuous variable.
`Averaging`	A character string indicating how the averaging should be done. "Event" (default) will produce the overall mean in the data, while "Subject" or "Item" (or, in principle, any other column name) will calculate the grand mean by that factor.
`VarLabel`	A string specifying the axis label to use for `Var`.
`xlim`	A vector of two integers specifying the limits of the x-axis.
`VWPreTheme`	A logical indicating whether the theme included with the function, or ggplot2's base theme (which any other custom theme could be added).
`Colors`	A vector of two strings specifying the colrs of the contour shading - The default values represent grayscale.

Examples

```
## Not run:
library(VWPre)
# For plotting a conditional contour surface...
plot_avg_contour(data = dat, IA = "IA_1_ELogit", type = "elogit",
                 Var = "Rating", VarLabel = "Accent Rating", xlim = c(0,1000),
                 VWPreTheme = FALSE, Colors = c("red", "white"))

# For a more complete tutorial on VWPre plotting functions:
vignette("SR_Plotting", package="VWPre")

## End(Not run)
```

`plot_avg_diff` *Plots average difference between looks to two interest areas.*

Description

`plot_avg_diff` calculates the grand or conditional averages of differences between looks to two interest area along with standard error. It then plots the results.

Usage

```
plot_avg_diff(
  data,
  DiffCols = NULL,
  xlim = NA,
  type = NULL,
  Averaging = "Event",
  Condition1 = NULL,
  Condition2 = NULL,
  Cond1Labels = NA,
  Cond2Labels = NA,
  ErrorBar = TRUE,
  VWPreTheme = TRUE,
  ConfLev = 95,
  CItype = "simultaneous",
  ErrorBand = FALSE,
  ErrorType = "SE"
)
```

Arguments

`data`	A data table object output by either `bin_prop`. `transform_to_elogit`, or `create_binomial`.
`DiffCols`	A named character vector specifying the desired columns corresponding to the interest areas.
`xlim`	A vector of two integers specifying the limits of the x-axis.
`type`	A character string indicating "proportion" or "elogit" which influences how standard error and confidence intervals are calculated.
`Averaging`	A character string indicating how the averaging should be done. "Event" (default) will produce the overall mean in the data, while "Subject" or "Item" (or, in principle, any other column name) will calculate the grand mean by that factor.
`Condition1`	A string containing the column name corresponding to the first condition, if available.
`Condition2`	A string containing the column name corresponding to the second condition, if available.
`Cond1Labels`	A named character vector specifying the desired labels of the levels of the first condition.
`Cond2Labels`	A named character vector specifying the desired labels of the levels of the second condition.
`ErrorBar`	A logical indicating whether error bars should be included in the plot.
`VWPreTheme`	A logical indicating whether the theme included with the function should be applied, or ggplot2's base theme (to which any other custom theme could be added).
`ConfLev`	A number indicating the confidence level of the CI.
`CItype`	A string indicating "simultaneous" or "pointwise". Simultaneous performs a Bonferroni correction for the interval.
`ErrorBand`	A logical indicating whether error bands should be included in the plot.
`ErrorType`	A string indicating "SE" or "CI".

Examples

```
## Not run:
library(VWPre)
# For plotting average differences with SE bars...
plot_avg_diff(data = dat, xlim = c(0, 1000), type = "proportion",
             DiffCols = c(IA_1_P = "Target", IA_2_P = "Rhyme"),
             Condition1 = NA, Condition2 = NA, Cond1Labels = NA, Cond2Labels = NA,
             ErrorBar = TRUE, VWPreTheme = TRUE, ErrorBand = FALSE,
             ErrorType = "SE")

# For plotting conditional average differences (one condition) with the
# included theme and 95% pointwise CI bars.
plot_avg_diff(data = dat, xlim = c(0, 1000), , type = "proportion",
             DiffCols = c(IA_1_P = "Target", IA_2_P = "Rhyme"),
           Condition1 = "talker", Condition2 = NA, Cond1Labels = c(CH1 = "Chinese 1",
           CH10 = "Chinese 3", CH9 = "Chinese 2", EN3 = "English 1"),
           Cond2Labels = NA, ErrorBar = TRUE,
           VWPreTheme = TRUE, ErrorBand = FALSE,
             ErrorType = "CI", ConfLev = 95, CItype = "pointwise")

# For plotting conditional average differences (two conditions) with the
# included theme and 95% simultaneous CI bands.
plot_avg_diff(data = dat, xlim = c(0, 1000), , type = "proportion",
             DiffCols = c(IA_1_P = "Target", IA_2_P = "Rhyme"),
           Condition1 = "talker", Condition2 = "Exp", Cond1Labels = c(CH1 = "Chinese 1",
           CH10 = "Chinese 3", CH9 = "Chinese 2", EN3 = "English 1"),
           Cond2Labels = c(High = "H Exp", Low = "L Exp"), ErrorBar = FALSE,
           VWPreTheme = TRUE, ErrorBand = TRUE,
             ErrorType = "CI", ConfLev = 95, CItype = "simultaneous")

# For a more complete tutorial on VWPre plotting functions:
vignette("SR_Plotting", package="VWPre")

## End(Not run)
```

---

plot_indiv_app *Plots diagnostic average plots of subjects/items.*

Description

`plot_indiv_app` calculates and plots interest area averages for a given subject/item.

Usage

```
plot_indiv_app(data)
```

Arguments

`data`	A data table object output by either `bin_prop`. `transform_to_elogit`, or `create_binomial`.

Examples

```
## Not run:
library(VWPre)
# For plotting subject/item averages
plot_indiv_app(data = dat)

## End(Not run)
```

---

plot_transformation_app *Plots diagnostic plots of the empirical logit transformation.*

Description

`plot_transformation_app` plots the empirical logit values for a given number of observations and constant against proportions, in order to examine the effect of these variables on the resulting transformation.

Usage

```
plot_transformation_app()
```

Examples

```
## Not run:
library(VWPre)
# For plotting the empirical logit transformation
plot_transformation_app()

## End(Not run)
```

---

plot_var_app *Plots diagnostic plots of subject/item variance.*

Description

`plot_var_app` calculates and plots within-subject/item standard deviation, along with standardized by-subject/item means for a given interest area, within a given time window.

Usage

```
plot_var_app(data)
```

Arguments

`data`	A data table object output by either `bin_prop`. `transform_to_elogit`, or `create_binomial`.

Examples

```
## Not run:
library(VWPre)
# For plotting variability in the data
plot_var_app(data = dat)

## End(Not run)
```

`prep_data` *Check the classes of specific columns and re-assigns as necessary.*

Description

`prep_data` converts the data frame to a data table and examines the required columns (RECORDING_SESSION_LABEL, LEFT_INTEREST_AREA_ID, RIGHT_INTEREST_AREA_ID, LEFT_INTEREST_AREA_LA
RIGHT_INTEREST_AREA_LABEL, TIMESTAMP, and TRIAL_INDEX) and optional columns (SAMPLE_MESSAGE, LEFT_GAZE_X, LEFT_GAZE_Y, RIGHT_GAZE_X, and RIGHT_GAZE_Y). It renames the subject and item columns, ensures required/optional columns are of the appropriate data class, and creates a new column called Event which indexes each unique series of samples corresponding to the combination of Subject and TRIAL_INDEX (can be changed), necessary for performing subsequent operations.

Usage

```
prep_data(
  data,
  Subject = NULL,
  Item = NA,
  EventColumns = c("Subject", "TRIAL_INDEX")
)
```

Arguments

`data`	A data frame object created from an Eyelink Sample Report.
`Subject`	An obligatory string containing the column name corresponding to the subject identifier.
`Item`	An optional string containing the column name corresponding to the item identifier; by default, NA.
`EventColumns`	A vector specifying the columns which will be used for creating the Event variable; by default, Subject and TRIAL_INDEX.

Value

An object of type data table as described in tibble.

Examples

```
## Not run:
# Typical DataViewer output contains a column called "RECORDING_SESSION_LABEL"
# corresponding to the subject.
# To prepare the data...
library(VWPre)
df <- prep_data(data = dat, Subject = "RECORDING_SESSION_LABEL", Item = "ItemCol")

## End(Not run)
```

`recode_ia` *Recode interest area IDs and/or interest area labels*

Description

`recode_ia` replaces existing interest area IDs and/or labels for both eyes. For subsequent data processing, it is important that the ID values range between 0 and 8 (with 0 representing Outside all predefined interest areas).

Usage

```
recode_ia(data, IDs = NULL, Labels = NULL)
```

Arguments

`data`	A data table object output by `relabel_na`.
`IDs`	A named character vector specifying the desired interest area IDs and the corresponding existing IDs where the first element is the old value and the second element is the new value.
`Labels`	A named character vector specifying the desired interest area labels and the corresponding existing labels where the first element is the old value and the second element is the new value.

Value

A data table with the same dimensions as `data`.

Examples

```
## Not run:
library(VWPre)
# To recode both IDs and Labels...
df <- recode_ia(data=dat, IDs=c("234"="2", "0"="0", "35"="3", "11"="1",
"4"="6666"), Labels=c(Outside="Outside", Target="NewTargName",
Dist2="NewDist2Name", Comp="NewCompName", Dist1="NewDist1Name"))

# For a more complete tutorial on VWPre plotting functions:
vignette("SR_Interest_Areas", package="VWPre")
```

```
## End(Not run)
```

relabel_na *Relabel samples containing 'NA' as outside any interest area*

Description

`relabel_na` examines interest area columns (LEFT_INTEREST_AREA_ID, RIGHT_INTEREST_AREA_ID, LEFT_INTEREST_AREA_LABEL, and RIGHT_INTEREST_AREA_LABEL) for cells containing NAs. If NA, the missing values in the ID columns are relabeled as 0 and missing values in the LABEL columns are relabeled as 'Outside'.

Usage

```
relabel_na(data, NoIA = NULL)
```

Arguments

`data`	A data table object output by `prep_data`.
`NoIA`	A positive integer indicating the number of interest areas defined when creating the study.

Value

A data table with the same dimensions as `data`.

Examples

```
## Not run:
library(VWPre)
# To relabel the NAs...
df <- relabel_na(data = dat, NoIA = 4)

## End(Not run)
```

rename_columns — *Rename default column names for interest areas.*

Description

rename_columns will replace the default numerical coding of the interest area columns with more meaningful user-specified names. For example, IA_1_C and IA_1_P could be converted to IA_Target_C and IA_Target_P. Again, this will work for upto 8 interest areas.

Usage

```
rename_columns(data, Labels = NULL)
```

Arguments

data	A data table object output by either bin_prop. transform_to_elogit, or create_binomial.
Labels	A named character vector specifying the interest areas and the desired names to be inserted in place of the numerical labelling.

Value

A data table object with renamed columns.

Examples

```
## Not run:
library(VWPre)
# For renaming default interest area columns
dat2 <- rename_columns(dat, Labels = c(IA1="Target", IA2="Rhyme",
                           IA3="OnsetComp", IA4="Distractor"))

## End(Not run)
```

rm_extra_DVcols — *Checks for and removes unnecessary DV output columns.*

Description

rm_extra_DVcols checks for unnecessary DataViewer output columns and removes them, unless specified.

Usage

```
rm_extra_DVcols(data, Keep = NULL)
```

Arguments

`data`	A data frame object created from an Eyelink Sample Report.
`Keep`	An optional string or character vector containing the column names of SR sample report columns the user would like to keep in the data set.

Value

An object of type data table as described in tibble.

Examples

```
## Not run:
library(VWPre)
df <- rm_extra_DVcols(data = dat, Keep = NULL)

## End(Not run)
```

`rm_trackloss_events` *Removes events with excessive trackloss*

Description

`rm_trackloss_events` removes events with less data than the specified amount.

Usage

```
rm_trackloss_events(data = data, RequiredData = NULL)
```

Arguments

`data`	A data table object output by `mark_trackloss`.
`RequiredData`	A number indicating the percentage of data required to be included (i.e., removes events with less than this amount of data).

Value

An object of type data table as described in tibble.

Examples

```
## Not run:
library(VWPre)
# Remove events...
df <- rm_trackloss_events(data = dat, RequiredData = 50)

## End(Not run)
```

select_recorded_eye *Select the eye used during recording*

Description

select_recorded_eye examines each event and determines which eye contains interest area information, based on the Recording parameter (which can be determined using check_eye_recording). This function then selects the data from the recorded eye and copies it to new columns (IA_ID, IA_LABEL, IA_Data). The function prints a summary of the output.

Usage

```
select_recorded_eye(data, Recording = NULL, WhenLandR = NA)
```

Arguments

data	A data table object output by create_time_series.
Recording	A string indicating which eyes were used for recording gaze data ("R" when only right eye recording is present, "L" when only left eye recording is present, "LorR" when either the left or the right eye was recorded, "LandR" when both the left and the right eyes were recorded).
WhenLandR	A string indicating which eye ("Right" or "Left) to use if gaze data is available for both eyes (i.e., Recording = "LandR").

Value

A data table with four additional columns ('EyeRecorded', 'EyeSelected', 'IA_ID', 'IA_LABEL', 'IA_Data') added to data.

Examples

```
## Not run:
library(VWPre)
# Create a unified columns for the gaze data...
df <- select_recorded_eye(data = dat, Recording = "LandR", WhenLandR = "Right")

## End(Not run)
```

`transform_to_elogit` *Transforms proportion looks to empirical logits.*

Description

`transform_to_elogit` transforms the proportion of looks for each interest area to empirical logits. Proportions are inherently bound between 0 and 1 and are therefore not suitable for some types of analysis. Logits provide an unbounded measure, though range from negative infinity to infinity, so it is important to know that this logit function adds a constant (hence, empirical logit). Additionally this calculates weights which estimate the variance in each bin (because the variance of the logit depends on the mean). This is important for regression analyses. N.B.: This function will work for data with a maximum of 8 interest areas.

Usage

```
transform_to_elogit(
  data,
  NoIA = NULL,
  ObsPerBin = NULL,
  Constant = 0.5,
  ObsOverride = FALSE
)
```

Arguments

`data`	A data table object output by `bin_prop`.
`NoIA`	A positive integer indicating the number of interest areas defined when creating the study.
`ObsPerBin`	A positive integer indicating the number of observations to use in the calculation. Typically, this will be the number of samples per bin, which can be determined with `check_samples_per_bin`.
`Constant`	A positive number used for the empirical logit and weights calculation; by default, 0.5 as in Barr (2008).
`ObsOverride`	A logical value controlling restrictions on the value provided to ObsPerBin. Default value is FALSE.

Details

These calculations were adapted from: Barr, D. J., (2008) Analyzing 'visual world' eyetracking data using multilevel logistic regression, *Journal of Memory and Language*, *59*(4), 457–474.

Value

A data table with additional columns (the number of which depends on the number of interest areas specified) added to `data`.

Examples

```
## Not run:
library(VWPre)
# Convert proportions to empirical logits and calculate weights...
df <- transform_to_elogit(dat, NoIA = 4, ObsPerBin = 20, Constant = 0.5)

## End(Not run)
```

`VWdat` *This is a sample eye-tracking dataset included in the package*

Description

This is a sample eye-tracking dataset included in the package

Author(s)

Vincent Porretta

`VWPre` *VWPre: Tools for Preprocessing Visual World Data.*

Description

The VWPre package provides a set of functions for preparing Visual World data collected with SR Research Eyelink eye trackers.

Formatting functions

- The function `create_time_series` returns a time columns in milliseconds.
- The function `prep_data` returns a data table with correctly assigned classes for important columns.
- The function `relabel_na` returns a data table with samples containing 'NA' relabeled as outside any interest area.
- The function `recode_ia` returns a data table containing recoded interest area IDs and/or interest area labels.
- The function `select_recorded_eye` returns a data table with data from the the recorded eye in new columns (IA_ID and IA_LABEL).
- The function `custom_ia` returns a data table with gaze data remapped to new interest areas.
- The function `align_msg` returns a data table with newly aligned sample data in a new column (Align).
- The function `rm_extra_DVcols` removed DataViewer coumns that are not necessary for preprocessing with this package.

Calculation functions

- The function `bin_prop` returns a downsampled data table containing proportion of looks (samples) to each interest area in a particular window of time (bin size).
- The function `transform_to_elogit` returns a data table with proportion looks transformed to empirical logits with weights.
- The function `create_binomial` returns a data table with a new success/failure column for each IA which is suitable for logistic regression.

Trackloss functions

- The function `mark_trackloss` returns a data table with data information regarding trackloss of the sample.
- The function `rm_trackloss_events` returns a data table from which events without the minimum amount of quality data have been removed.

Fasttrack formatting function

- The function `fasttrack` a meta-function that returns a data table of processed data containing the result of the series of necessary subroutines. This is intended for experienced users doing basic preprocessing.

Data-checking functions

- The function `check_eye_recording` returns a summary of whether or not the dataset contains gaze data in both the Right and Left interest area columns.
- The function `check_time_series` returns the first value in the Time column for each event.
- The function `check_samples_per_bin` returns the number of samples in each bin.
- The function `check_samplingrate` returns the value corresponding to the sampling rate in the data.
- The function `ds_options` returns the binning (downsampling) options possible for the given sampling rate.
- The function `check_ia` returns a summary of the interest area IDs and Labels present in the data.
- The function `check_msg_time` returns a summary of the the time value at a given sample message for each recording event.
- The function `check_all_msgs` returns all messages in the data and their time stamp.

Plotting functions

- The function `plot_avg` returns a plot of the grand or conditional averages of proportion (or empirical logit) looks to each interest area along with error bars.
- The function `plot_avg_contour` returns a contour plot of the conditional average of proportion (or empirical logit) looks to a given interest area over Time and a specified continuous variable.

- The function `plot_avg_diff` returns a plot of the grand or conditional averages of the difference between looks to two interest areas (proportions or empirical logits) with error bars.
- The function `plot_avg_cdiff` returns a plot of the average difference between two conditions for looks to a given interest area (proportions or empirical logits) with error bars.
- The function `make_pelogit_fnc` returns a function that can backtransform predicted empirical logit to probability scale, particularly (though not exclusively) useful for plotting purposes.

Interactive functions

- The function `plot_transformation_app` opens a Shiny app for visualizing the effect of both number of observations and constant on the results of the empirical logit transformation and weight calculations.
- The function `plot_indiv_app` opens a Shiny app for inspecting by-subject or by-item averages for all interest areas, alongside the grand average (for proportion or empirical logit looks) within a specified time window.
- The function `plot_var_app` opens a Shiny app for inspecting by-subject or by-item Z-scores with respect to the overall mean for a given interest area within a specified time window.

Notes

- The vignettes are available via `browseVignettes(package = "VWPre")`.
- A list of all available functions is provided in `help(package = "VWPre")`.
- This package can be cited using the information obtained from `citation("VWPre")` or `print(citation("VWPre"), bibtex = TRUE)`

Author(s)

Vincent Porretta, Aki-Juhani Kyröläinen, Jacolien van Rij, Juhani Järvikivi

Maintainer: Vincent Porretta (<vincentporretta@gmail.com>)

University of Windsor, Canada

Index