Package ‘eyetrackingR’

October 13, 2022

Type Package

Version 0.2.0

Title Eye-Tracking Data Analysis

Description Addresses tasks along the pipeline from raw data to analysis and visualization for eye-tracking data. Offers several popular types of analyses, including linear and growth curve time analyses, onset-contingent reaction time analyses, as well as several non-parametric bootstrapping approaches. For references to the approach see Mirman, Dixon & Magnuson (2008) <doi:10.1016/j.jml.2007.11.006>, and Barr (2008) <doi:10.1016/j.jml.2007.09.002>.

URL `http://www.eyetracking-r.com/`

BugReports `https://github.com/samhforbes/eyetrackingR/issues`

Date 2021-09-24

Depends R (>= 3.2.0), dplyr (>= 0.7.4)

License MIT + file LICENSE

LazyData true

Imports broom (>= 0.3.7), broom.mixed, ggplot2 (>= 2.0), lazyeval (>= 0.1.10), rlang, zoo (>= 1.7-12), tidyr (>= 0.3.1), purrr (>= 0.2.4)

Suggests pbapply, knitr, lme4 (>= 1.1-10), glmmTMB, MASS, Matrix, testthat, rmarkdown, doMC, foreach

VignetteBuilder knitr

RoxygenNote 7.1.1

NeedsCompilation no

Author Samuel Forbes [aut, cre],
Jacob Dink [aut],
Brock Ferguson [aut]

Maintainer Samuel Forbes <`samuel.h.forbes@gmail.com`>

Repository CRAN

Date/Publication 2021-09-27 10:00:14 UTC

R topics documented:

`add_aoi` *Add an area-of-interest to your dataset, based on x-y coordinates and the AOI rectangle.*

Description

Eyetracking-R requires that there is a column for each area-of-interest, specifying whether the gaze is within that area for each sample. This function creates an AOI column if needed.

Usage

```
add_aoi(
  data,
  aoi_dataframe,
  x_col,
  y_col,
  aoi_name,
  x_min_col = "L",
  x_max_col = "R",
  y_min_col = "T",
  y_max_col = "B"
)
```

Arguments

`data`	Your data
`aoi_dataframe`	A dataframe specifying the bounding-box for the AOI
`x_col, y_col`	What are the column names for the x and y coordinates in your dataset?
`aoi_name`	What is the name of this AOI?
`x_min_col, x_max_col`	What are the column names for the left and right edge of the AOI-bounding box? Default "L","R"
`y_min_col, y_max_col`	What are the column names for the top and bottom edge of the AOI-bounding box? Default "T","B"

Details

Many eyetracking software packages export your data with a column corresponding to each AOI; however, if your software does not do this, or if you had to define or revise your AOIs after running the experiment, then this function will add the necessary AOI columns for you. The function takes two dataframes: (1) your original data, (2) a dataframe specifying the bounding box for the AOI. The latter can specify a different bounding box for each trial, each subject, each image, or even each video-frame– anything you like. The two dataframes are simply joined by matching any columns they have in common (case sensitive!), so if there's a unique AOI for each "Trial" in the `aoi_dataframe`, and there's a "Trial" column in the `data` dataframe, then the unique AOI coordinates for each trial will be used.

Value

Dataset with a new column indicating whether gaze is in the AOI

`analyze_boot_splines` *Estimate confidence intervals for bootstrapped splines data*

Description

Deprecated. Performing this analysis should be done by calling `analyze_time_bins(test="boot_splines")`.

Usage

```
analyze_boot_splines(data)

## S3 method for class 'boot_splines_data'
analyze_boot_splines(data)
```

Arguments

`data`	The output of the `boot_splines_data` function

Details

Estimates a confidence interval over the difference between means (within- or between-subjects) from `boot_splines_data`. Confidence intervals are derived from the alpha argument in `boot_splines_data` (e.g., alpha = .05, CI=(.025,.975); alpha=.01, CI=(.005,.0995))

Value

A dataframe indicating means and CIs for each time-bin

Methods (by class)

- `boot_splines_data`:

Examples

```
## Not run:
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE )
response_window <- subset_by_window(data, window_start_time = 15500, window_end_time = 21000,
                                    rezero = FALSE)
response_time <- make_time_sequence_data(response_window, time_bin_size = 500, aois = "Animate",
                                         predictor_columns = "Sex",
                                         summarize_by = "ParticipantName")
```

```
# bootstrap resample 500 smoothed splines from the dataset,
# comparing females versus females at an alpha of .05
df_bootstrapped <- make_boot_splines_data(response_time,
                                          predictor_column = 'Sex',
                                          within_subj = FALSE,
                                          bs_samples = 500,
                                          alpha = .05,
                                          smoother = "smooth.spline")

# analyze the divergences that occurred
boot_splines_analysis <- analyze_boot_splines(df_bootstrapped)
summary(boot_splines_analysis)

## End(Not run)
```

`analyze_time_bins` *analyze_time_bins()*

Description

Runs a test on each time-bin of `time_sequence_data`. Supports `t.test`, `wilcox.test`, `(g)lm`, and `(g)lmer`. Also includes support for the "bootstrapped-splines" test (see `?make_boot_splines_data` and the divergence vignette for more info). By default, this function uses 'proportion-looking' (`Prop`) as the DV, which can be changed by manually specifying the formula. Results can be plotted to see how test-results or parameters estimates vary over time. P-values can be adjusted for multiple comparisons with `p_adjust_method`.

Usage

```
analyze_time_bins(data, ...)

## S3 method for class 'time_sequence_data'
analyze_time_bins(
  data,
  predictor_column,
  test,
  threshold = NULL,
  alpha = NULL,
  aoi = NULL,
  formula = NULL,
  treatment_level = NULL,
  p_adjust_method = "none",
  quiet = FALSE,
  ...
)
```

Arguments

`data`	The output of the ’make_time_sequence_data’ function
`...`	Any other arguments to be passed to the selected ’test’ function (e.g., paired, var.equal, etc.)
`predictor_column`	The variable whose test statistic you are interested in. If you are not interested in a predictor, but the intercept, you can enter "intercept" for this argument. Interaction terms are not currently supported.
`test`	What type of test should be performed in each time bin? Supports `t.test`, `wilcox.test`, `(g)lm`, and `(g)lmer`. Also includes support for the "bootstrapped-splines" test (see `?make_boot_splines_data` and the divergence vignette for more info).
`threshold`	Value of statistic used in determining significance
`alpha`	Alpha value for determining significance, ignored if threshold is given
`aoi`	Which AOI should be analyzed? If not specified (and dataframe has multiple AOIs), then AOI should be a predictor/covariate in your model (so ‘formula‘ needs to be specified).
`formula`	What formula should be used for the test? Optional for all but `(g)lmer`, if unset will use `Prop` ~ `[predictor_column]`. Change this if you want to use a custom DV.
`treatment_level`	If your predictor is a factor, regression functions like ‘lm‘ and ‘lmer‘ by default will treatment-code it. One option is to sum-code this predictor yourself before entering it into this function. Another is to use the ‘treatment_level‘ argument, which specifies the level of the predictor. For example, you are testing a model where ‘Target‘ is a predictor, which has two levels, ’Animate’ and ’Inanimate’. R will code ’Animate’ as the reference level, and code ’Inanimate’ as the treatment level. You’d therefore want to set ‘treatment_level = Inanimate‘.
`p_adjust_method`	Method to adjust `p.values` for multiple corrections (default="none"). See `p.adjust.methods`.
`quiet`	Should messages and progress bars be suppressed? Default is to show

Value

A dataframe indicating the results of the test at each time-bin.

Methods (by class)

- `time_sequence_data`:

Examples

```
## Not run:
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
```

```
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE
)
response_time <- make_time_sequence_data(data, time_bin_size = 250,
                                         predictor_columns = c("MCDI_Total"),
                                      aois = "Animate", summarize_by = "ParticipantName")
tb_analysis <- analyze_time_bins(response_time, predictor_column = "MCDI_Total",
                                 test = "lm", threshold = 2)
summary(tb_analysis)

## End(Not run)
```

`analyze_time_clusters` *Bootstrap analysis of time-clusters.*

Description

Takes data whose time bins have been clustered by test-statistic (using the `make_time_cluster_data` function) and performs a permutation test (Maris & Oostenveld, 2007). This analysis takes a summed statistic for each cluster, and compares it to the "null" distribution of sum statistics obtained by shuffling/resampling the data and extracting the largest cluster from each resample.

Usage

```
analyze_time_clusters(data, ...)

## S3 method for class 'time_cluster_data'
analyze_time_clusters(
  data,
  within_subj,
  samples = 2000,
  formula = NULL,
  shuffle_by = NULL,
  parallel = FALSE,
  quiet = FALSE,
  ...
)
```

Arguments

`data`	The output of the `make_time_cluster_data` function
`...`	Other args for to selected 'test' function; should be identical to those passed to `make_time_cluster_data` function
`within_subj`	Logical indicating whether to perform within-subjects bootstrap resampling.

`samples`	How many iterations should be performed in the bootstrap resampling procedure?
`formula`	Formula for test. Should be identical to that passed to make_time_cluster_data fxn (if arg ignored there, can be ignored here)
`shuffle_by`	Along which attribute should the data be resampled? Default is the predictor column. But if the predictor_column is numeric *and* within-subjects, then observations with the same predictor value could nevertheless correspond to distinct conditions/categories that should be shuffled separately. For example, when using vocabulary scores to predict looking behavior, a participant might get identical vocab scores for verbs and nouns; these are nevertheless distinct categories that should be re-assigned separately when bootstrap-resampling data. The 'shuffle_by' argument allows you to specify a column which indicates these kinds of distinct categories that should be resampled separately– but it's only needed if you've specified a numeric *and* within-subjects predictor column.
`parallel`	Use foreach for speed boost? By default off. May not work on Windows.
`quiet`	Display progress bar/messages? No progress bar when parallel=TRUE.

Value

A cluster-analysis object, which can be plotted and summarized to examine which temporal periods show a significant effect of the predictor variable

Methods (by class)

- `time_cluster_data`:

Examples

```
## Not run:
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE )
response_window <- subset_by_window(data, window_start_time = 15500, window_end_time = 21000,
                                    rezero = FALSE)
response_time <- make_time_sequence_data(response_window, time_bin_size = 500, aois = "Animate",
                                         predictor_columns = "Sex")

time_cluster_data <- make_time_cluster_data(data = response_time, predictor_column = "SexM",
                         aoi = "Animate", test = "lmer",
                         threshold = 1.5,
                        formula = LogitAdjusted ~ Sex + (1|Trial) + (1|ParticipantName))
summary(time_cluster_data)
plot(time_cluster_data)

# analyze time clusters in a non-parametric analysis
```

```
tc_analysis <- analyze_time_clusters(time_cluster_data, within_subj = FALSE,
                                     samples = 2000)
plot(tc_analysis)
summary(tc_analysis)

## End(Not run)
```

`clean_by_trackloss` *Clean data by removing high-trackloss trials/subjects.*

Description

Remove trials/participants with too much trackloss, with a customizable threshold.

Usage

```
clean_by_trackloss(
  data,
  participant_prop_thresh = 1,
  trial_prop_thresh = 1,
  window_start_time = -Inf,
  window_end_time = Inf
)
```

Arguments

`data`	Data already run through `make_eyetrackingr_data`
`participant_prop_thresh`	Maximum proportion of trackloss for participants
`trial_prop_thresh`	Maximum proportion of trackloss for trials
`window_start_time, window_end_time`	Time-window within which you want trackloss analysis to be based. Allows you to keep the entire trial window for data, but clean based on the trackloss within a subset of it

Value

Cleaned data

Examples

```
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE
)

# scrub all trials with greater than 25% trackloss, and all
# participants with greater than 25% trackloss on average
# during the timeperiod 15500-2100
data_clean <- clean_by_trackloss(data,
                                 participant_prop_thresh = .25,
                                 trial_prop_thresh = .25,
                                 window_start_time = 15500,
                                 window_end_time = 21000
)

# scrub all trials with greater than 25% trackloss, but leave participants with a high average
data_clean <- clean_by_trackloss(data,
                                 trial_prop_thresh = .25,
                                 window_start_time = 15500,
                                 window_end_time = 21000
)
```

`describe_data` *Describe dataset*

Description

Returns descriptive statistics about a column of choice. A simple convenience function that wraps `dplyr::group_by` and `dplyr::summarize`, allowing a quick glance at the data.

Usage

```
describe_data(
  data,
  describe_column,
  group_columns,
  quantiles = c(0.025, 0.975)
)
```

Arguments

`data`	Data already run through `make_eyetrackingr_data`
`describe_column`	The column to return descriptive statistics about.
`group_columns`	Any columns to group by when calculating descriptive statistics (e.g., participants, conditions, etc.)
`quantiles`	Numeric vector of length two with quantiles to compute (default: `c(.025, .975)`).

Value

A dataframe giving descriptive statistics for the `describe_column`, including mean, SD, var, min, max, and number of trials

Examples

```
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE
)
describe_data(data, describe_column = "Animate", group_columns = "ParticipantName")
```

`eyetrackingR` *eyetrackingR: A package for cleaning, analyzing, and visualizing eye-tracking datasets*

Description

This package addresses tasks along the pipeline from raw eye-tracking data to analysis and visualization. It offers several popular types of analyses, including linear and growth curve time analyses, onset-contingent reaction time analyses, and cluster mass analyses, as well as novel non-parametric approaches to time-series data.

Details

For more information and tutorials, visit http://www.eyetracking-r.com/.

`get_time_clusters` *Get information about the clusters in a cluster-analysis*

Description

Get information about the clusters in a cluster-analysis

Usage

```
get_time_clusters(object)

## S3 method for class 'time_cluster_data'
get_time_clusters(object)

## S3 method for class 'cluster_analysis'
get_time_clusters(object)
```

Arguments

`object`	The output of the `analyze_time_clusters` function

Value

A dataframe with information about the clusters

Methods (by class)

- `time_cluster_data`: Get time clusters dataframe
- `cluster_analysis`: Get time clusters dataframe

`make_boot_splines_data` *Bootstrap resample splines for time-series data.*

Description

Deprecated. Performing this analysis should be done by calling `analyze_time_bins(test="boot_splines")`.

Usage

```
make_boot_splines_data(
  data,
  predictor_column,
  within_subj,
  aoi,
  bs_samples,
  smoother,
  resolution,
  alpha,
  ...
)

## S3 method for class 'time_sequence_data'
make_boot_splines_data(
  data,
  predictor_column,
  within_subj,
  aoi = NULL,
  bs_samples = 1000,
  smoother = "smooth.spline",
  resolution = NULL,
  alpha = 0.05,
  ...
)
```

Arguments

`data`	The output of `time_sequence_data()`
`predictor_column`	What predictor var to split by? Maximum two conditions
`within_subj`	Are the two conditions within or between subjects?
`aoi`	Which AOI do you wish to perform the analysis on?
`bs_samples`	How many iterations to run bootstrap resampling? Default 1000
`smoother`	Smooth data using "smooth.spline," "loess," or "none" for no smoothing
`resolution`	What resolution should we return predicted splines at, in ms? e.g., 10ms = 100 intervals per second, or hundredths of a second. Default is the same size as time-bins.
`alpha`	p-value when the groups are sufficiently "diverged"
`...`	Ignored

Details

This method builds confidence intervals around proportion-looking data by bootstrap resampling. Data can be smoothed by fitting smoothing splines. This function performs the bootstrap resampling, `analyze_boot_splines` generates confidence intervals and tests for divergences.

Limited to statistical test between two conditions.

Value

A bootstrapped distribution of samples for each time-bin

Methods (by class)

- `time_sequence_data`:

Examples

```
## Not run:
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE )
response_window <- subset_by_window(data, window_start_time = 15500,
                                    window_end_time = 21000, rezero = FALSE)
response_time <- make_time_sequence_data(response_window, time_bin_size = 500, aois = "Animate",
                                         predictor_columns = "Sex",
                                         summarize_by = "ParticipantName")

df_bootstrapped <- make_boot_splines_data(response_time,
                                          predictor_column = 'Sex',
                                          within_subj = FALSE,
                                          bs_samples = 500,
                                          alpha = .05,
                                          smoother = "smooth.spline")

## End(Not run)
```

`make_eyetrackingr_data`

Convert raw data for use in eyetrackingR

Description

This should be the first function you use when using eyetrackingR for a project (potentially with the exception of ‘add_aoi‘, if you need to add AOIs). This function takes your raw dataframe, as well as information about your dataframe. It confirms that all the columns are the right format, based on this information. Further if `treat_non_aoi_looks_as_missing` is set to TRUE, it converts non-AOI looks to missing data (see the "Preparing your data" vignette for more information).

Usage

```
make_eyetrackingr_data(
  data,
  participant_column,
  trackloss_column,
  time_column,
  trial_column,
  aoi_columns,
  treat_non_aoi_looks_as_missing,
  item_columns = NULL
)
```

Arguments

`data`	Your original data. See details section below.
`participant_column`	Column name for participant identifier
`trackloss_column`	Column name indicating trackloss
`time_column`	Column name indicating time
`trial_column`	Column name indicating trial identifier
`aoi_columns`	Names of AOIs
`treat_non_aoi_looks_as_missing`	This is a logical indicating how you would like to perform "proportion-looking" calculations, which are central to eyetrackingR's eyetracking analyses. If set to TRUE, any samples that are not in any of the AOIs (defined with the `aoi_columns` argument) are treated as missing data; when it comes time for eyetrackingR to calculate proportion looking to an AOI, this will be calculated as "time looking to that AOI divided by time looking to all other AOIs." In contrast, if this parameter is set to FALSE, proportion looking to an AOI will be calculated as "time looking to that AOI divided by total time looking."
`item_columns`	Column names indicating items (optional)

Details

eyetrackingR is designed to deal with data in a (relatively) raw form, where each row specifies a sample. Each row should represent an equally spaced unit of time (e.g., if your eye-tracker's sample rate is 100hz, then each row corresponds to the eye-position every 10ms). This is in contrast to the more parsed data that the software bundled with eye-trackers can sometimes output (e.g., already parsed into saccades or fixations). For eyetrackingR, the simplest data is the best. This also maximizes compatibility: eyetrackingR will work with any eye-tracker's data (e.g., Eyelink, Tobii, etc.), since it requires the most basic format.

Value

Dataframe ready for use in eyetrackingR.

Examples

```
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE
)
```

`make_onset_data` *Make onset-contingent data.*

Description

Divide trials into which AOI participants started on. Calculate switches away from this AOI, using a rolling window to determine what length consitutes a switch. Augment original data with a column indicating whether each row is a switch-away sample.

Usage

```
make_onset_data(
  data,
  onset_time,
  fixation_window_length = NULL,
  target_aoi,
  distractor_aoi = NULL
)
```

Arguments

`data`	The original (verified) data
`onset_time`	When to check for participants' "starting" AOI?
`fixation_window_length`	Which AOI is currently being fixated is determined by taking a rolling average of this length (ms). This is the width of window for rolling average.
`target_aoi`	Which AOI is the target that should be switched *to*
`distractor_aoi`	Which AOI is the distractor that should be switched *from* (default = !target_aoi)

Value

Original dataframe augmented with column indicating switch away from target AOI

Examples

```
## Not run:
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE
)
response_window <- subset_by_window(data, window_start_time = 15500, window_end_time = 21000,
                                    rezero = FALSE)
inanimate_trials <- subset(response_window, grepl('(Spoon|Bottle)', Trial))
onsets <- make_onset_data(inanimate_trials, onset_time = 15500, target_aoi='Inanimate')

## End(Not run)
```

`make_switch_data` *Summarize data into time-to-switch from initial AOI.*

Description

Take trials split by initial-AOI, and determine how quickly participants switch away from that AOI

Usage

```
make_switch_data(data, predictor_columns, summarize_by)

## S3 method for class 'onset_data'
make_switch_data(data, predictor_columns = NULL, summarize_by = NULL)
```

Arguments

`data`	The output of `make_onset_data`
`predictor_columns`	Variables/covariates of interest when analyzing time-to-switch
`summarize_by`	Should the data be summarized along, e.g., participants, items, etc.? If so, give column name(s) here. If left blank, will leave trials distinct. The former is needed for more traditional analyses (t.tests, ANOVAs), while the latter is preferable for mixed-effects models (lmer)

Value

A dataframe indicating initial AOI and time-to-switch from that AOI for each trial/subject/item/etc.

Methods (by class)

- `onset_data`:

Examples

```
## Not run:
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE
)
response_window <- subset_by_window(data, window_start_time = 15500, window_end_time = 21000,
                                    rezero = FALSE)
inanimate_trials <- subset(response_window, grepl('(Spoon|Bottle)', Trial))
onsets <- make_onset_data(inanimate_trials, onset_time = 15500,
                          fixation_window_length = 100, target_aoi='Inanimate')

df_switch <- make_switch_data(onsets, predictor_columns = "MCDI_Total",
             summarize_by = "ParticipantName")
plot(df_switch, "MCDI_Total")

## End(Not run)
```

`make_time_cluster_data`

Make data for cluster analysis.

Description

Takes data that has been summarized into time-bins by `make_time_sequence_data()`, finds adjacent time bins that pass some test-statistic threshold, and assigns these adjacent bins into groups (clusters). Output is ready for a cluster permutation-based analyses (Maris & Oostenveld, 2007). Supports `t.test`, `wilcox.test`, `(g)lm`, and `(g)lmer`. Also includes support for the "bootstrapped-splines" test (see `?make_boot_splines_data` and the divergence vignette for more info). By default, this function uses 'proportion-looking' (`Prop`) as the DV, which can be changed by manually specifying the formula.

Usage

```
make_time_cluster_data(data, ...)

## S3 method for class 'time_sequence_data'
make_time_cluster_data(
```

```
    data,
    predictor_column,
    aoi = NULL,
    test,
    threshold = NULL,
    formula = NULL,
    treatment_level = NULL,
    ...
)
```

Arguments

`data`	The output of the `make_time_sequence_data` function
`...`	Any other arguments to be passed to the selected 'test' function (e.g., paired, var.equal, etc.)
`predictor_column`	The column name containing the variable whose test statistic you are interested in.
`aoi`	Which AOI should be analyzed? If not specified (and dataframe has multiple AOIs), then AOI should be a predictor/covariate in your model (so ‘formula‘ needs to be specified).
`test`	What type of test should be performed in each time bin? Supports `t.test`, `(g)lm`, or `(g)lmer`. Also includes experimental support for the "bootstrapped-splines" test (see `?make_boot_splines_data` and the divergence vignette for more info). Does not support `wilcox.test`.
`threshold`	Time-bins with test-statistics greater than this amount will be grouped into clusters.
`formula`	What formula should be used for test? Optional (for all but `(g)lmer`), if unset uses `Prop ~ [predictor_column]`
`treatment_level`	If your predictor is a factor, regression functions like ‘lm‘ and ‘lmer‘ by default will treatment-code it. One option is to sum-code this predictor yourself before entering it into this function. Another is to use the ‘treatment_level‘ argument, which specifies the level of the predictor. For example, you are testing a model where ‘Target‘ is a predictor, which has two levels, 'Animate' and 'Inanimate'. R will code 'Animate' as the reference level, and code 'Inanimate' as the treatment level. You'd therefore want to set ‘treatment_level = Inanimate‘.

Value

The original data, augmented with information about clusters. Calling summary on this data will describe these clusters. The dataset is ready for the `analyze_time_clusters` method.

Methods (by class)

- `time_sequence_data`: Make data for time cluster analysis

Examples

```
## Not run:
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE )
response_window <- subset_by_window(data, window_start_time = 15500, window_end_time = 21000,
                                    rezero = FALSE)

# identify clusters in the sequence data using a t-test with
# threshold t-value of 2

# (note: t-tests require a summarized dataset)
response_time <- make_time_sequence_data(response_window, time_bin_size = 500, aois = "Animate",
                                         predictor_columns = "Sex",
                                         summarize_by = "ParticipantName")

time_cluster_data <- make_time_cluster_data(data = response_time,
                                            predictor_column = "Sex",
                                            aoi = "Animate",
                                            test = "t.test",
                                            threshold = 2
)

# identify clusters in the sequence data using an lmer() random-effects
# model with a threshold t-value of 1.5.

# random-effects models don't require us to summarize
response_time <- make_time_sequence_data(response_window, time_bin_size = 500, aois = "Animate",
                                         predictor_columns = "Sex")

# but they do require a formula to be specified
time_cluster_data <- make_time_cluster_data(data = response_time,
                           predictor_column = "SexM",
                           aoi = "Animate",
                           test = "lmer",
                           threshold = 1.5,
                         formula = LogitAdjusted ~ Sex + (1|Trial) + (1|ParticipantName)
)

## End(Not run)
```

`make_time_sequence_data` *make_time_sequence_data()*

Description

Creates time-bins and summarizes proportion-looking within each time-bin.

Usage

```
make_time_sequence_data(
  data,
  time_bin_size,
  aois = NULL,
  predictor_columns = NULL,
  other_dv_columns = NULL,
  summarize_by = NULL
)
```

Arguments

`data`	The output of `make_eyetrackingr_data`
`time_bin_size`	How large should each time bin be? Units are whatever units your `time` column is in
`aois`	Which AOI(s) is/are of interest? Defaults to all specified in `make_eyetracking_r_data`
`predictor_columns`	Which columns indicate predictor variables, and therefore should be preserved in grouping operations?
`other_dv_columns`	Within each time-bin, this function will calculate not only proportion- looking, but also the mean of any columns specified here.
`summarize_by`	Should the data be summarized along, e.g., participants, items, etc.? If so, give column name(s) here. If left blank, will leave trials distinct. The former is needed for more traditional analyses (`t.test`, `ANOVA`), while the latter is preferable for mixed-effects models (`lmer`)

Details

Aside from proportion looking (`Prop`), this function returns several columns useful for subsequent analysis:

- `LogitAdjusted` - The logit is defined as `log( Prop / (1 - Prop) )`. This transformation attempts to map bounded `0,1` data to the real number line. Unfortunately, for data that is exactly 0 or 1, this is undefined. One solution is add a very small value to any datapoints that equal 0, and subtract a small value to any datapoints that equal 1 (we use 1/2 the smallest nonzero value for this adjustment).
- `Elog` - Another way of calculating a corrected logit transformation is to add a small value `epsilon` to both the numerator and denominator of the logit equation (we use 0.5).
- `Weights` - These attempt to further correct the Elog transformation, since the variance of the logit depends on the mean. They can be used in a mixed effects model by setting the `weights=Weights` in `lmer` (note that this is the reciprocal of the weights calculated in this empirical logit walkthrough, so you do *not* set `weights = 1/Weights` as done there.)

- `ArcSin` - The arcsine-root transformation of the raw proportions, defined as `asin(sqrt(Prop))`
- `ot` - These columns (ot1-ot7) represent (centered) orthogonal time polynomials, needed for growth curve analysis. See the vignette on growth curve models for more details.

Value

Data binned into time-bins, with proportion-looking and transformations as well as orthogonal time-polynomials for growth curve analysis

Examples

```
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE
)

# bin data in 250ms bins, and generate a dataframe
# with a single AOI (Animate) predicted by Sex, and summarized by ParticipantName
response_time <- make_time_sequence_data(data,
                                         time_bin_size = 250,
                                         predictor_columns = c("Sex"),
                                         aois = "Animate",
                                         summarize_by = "ParticipantName"
)

# optionally specify other columns in the data
# to be included in the generated dataframe
# (e.g., for use in statistical models)
# bin data in 250ms bins, and generate a dataframe
# with Animate and MCDI_Total summarized by ParticipantName
response_time <- make_time_sequence_data(data,
                                         time_bin_size = 250,
                                         predictor_columns = c("Sex","MCDI_Total"),
                                         aois = "Animate",
                                         summarize_by = "ParticipantName"
)
```

`make_time_window_data` *Make a dataset collapsing over a time-window*

Description

Collapse time across our entire window and return a dataframe ready for analyses

Usage

```
make_time_window_data(
  data,
  aois = NULL,
  predictor_columns = NULL,
  other_dv_columns = NULL,
  summarize_by = NULL
)
```

Arguments

`data`	The output of `make_eyetrackingr_data`
`aois`	Which AOI(s) is/are of interest? Defaults to all specified in `make_eyetracking_r_data`
`predictor_columns`	Which columns indicate predictor vars, and therefore should be preserved in grouping operations?
`other_dv_columns`	Within each participant/trial (or whatever is specified in `summarize_by`), this function will calculate not only proportion-looking, but also the mean of any columns specified here.
`summarize_by`	Should the data be summarized along, e.g., participants, items, etc.? If so, give column names here. If left blank, will leave trials distinct. The former is needed for more traditional analyses (`t.test`, `ANOVA`), while the latter is preferable for mixed-effects models (`lmer`)

Details

Aside from proportion looking (`Prop`), this function returns several columns useful for subsequent analysis:

- `LogitAdjusted` - The logit is defined as `log( Prop / (1 - Prop) )`. This transformation attempts to map bounded `0,1` data to the real number line. Unfortunately, for data that is exactly 0 or 1, this is undefined. One solution is add a very small value to any datapoints that equal 0, and subtract a small value to any datapoints that equal 1 (we use 1/2 the smallest nonzero value for this adjustment).
- `Elog` - Another way of calculating a corrected logit transformation is to add a small value `epsilon` to both the numerator and denominator of the logit equation (we use 0.5).
- `Weights` - These attempt to further correct the Elog transformation, since the variance of the logit depends on the mean. They can be used in a mixed effects model by setting the `weights=Weights` in `lmer` (note that this is the reciprocal of the weights calculated in this empirical logit walkthrough, so you do *not* set `weights = 1/Weights` as done there.)
- `ArcSin` - The arcsine-root transformation of the raw proportions, defined as `asin(sqrt(Prop))`

Value

Data with proportion-looking and transformations (logit, arc-sin, etc.)

Examples

```
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE
)

# generate a dataset summarizing an AOI (Animate) by ParticipantName
response_window_agg_by_sub <- make_time_window_data(data,
                                                     aois='Animate',
                                                     summarize_by = "ParticipantName"
)

## Not run:
# optionally included additional columns for use as predictors
# in later statistical models
response_window_agg_by_sub <- make_time_window_data(data,
                                                     aois='Animate',
                                                 predictor_columns=c('Age','MCDI_Total'),
                                                     summarize_by = "ParticipantName"
)

# plot the aggregated data for sanity check
plot(response_window_agg_by_sub, predictor_columns="Age", dv = "LogitAdjusted")

## End(Not run)
```

`plot.bin_analysis` *Plot test-statistic for each time-bin in a time-series*

Description

Plot the result from the `analyze_time_bins` function, with the statistic and threshold for each bin

Usage

```
## S3 method for class 'bin_analysis'
plot(x, type = NULL, ...)
```

Arguments

`x`	The output of `analyze_time_bins`

`type`	This function can plot the test-statistic ("statistic"), the parameter estimate +/- std. error ("estimate"), the p-value ("pvalue") or the negative-log-pvalue ("neg_log_pvalue"). When test gives critical-statistic, default is to plot the test-statistic. Otherwise, default is to plot the estimate. For wilcox, only p-values can be plotted.
`...`	Ignored

Value

A ggplot object

`plot.boot_splines_analysis`
Plot differences in bootstrapped-splines data

Description

Plot the means and CIs of bootstrapped spline difference estimates and intervals (either within-subjects or between-subjects)

Usage

```
## S3 method for class 'boot_splines_analysis'
plot(x, ...)
```

Arguments

`x`	The output of the `analyze_boot_splines` function
`...`	Ignored

Value

A ggplot object

`plot.boot_splines_data`
Plot bootstrapped-splines data

Description

Plot the means and CIs of bootstrapped splines (either within-subjects or between-subjects)

Usage

```
## S3 method for class 'boot_splines_data'
plot(x, ...)
```

Arguments

`x`	The output of the `make_boot_splines_data` function
`...`	Ignored

Value

A ggplot object

`plot.cluster_analysis` *Visualize the results of a cluster analysis.*

Description

Plots the result of the bootstrapping cluster analysis. A histogram of the sum statistics for the shuffled (null) distribution, with the sum statisics for each of the clusters indicated by dashed lines.

Usage

```
## S3 method for class 'cluster_analysis'
plot(x, ...)
```

Arguments

`x`	object returned by cluster_analysis()
`...`	Ignored

Value

A ggplot object

`plot.eyetrackingR_data_summary`
Plot some summarized data from eyetrackingR

Description

Plots the data returned from `describe_data`. Like that function, this is a convenient wrapper good for sanity checks.

Usage

```
## S3 method for class 'eyetrackingR_data_summary'
plot(x, ...)
```

Arguments

`x`	The data returned by `make_time_window_data()`
`...`	Ignored

Value

A ggplot object

`plot.onset_data` *Plot onset-contingent data*

Description

Divide trials into which AOI participants started on; plot proportion looking away from that AOI.

Usage

```
## S3 method for class 'onset_data'
plot(x, predictor_columns = NULL, ...)
```

Arguments

`x`	The output of the `make_onset_data` function
`predictor_columns`	Column(s) by which to facet the data. Maximum two columns. Will perform median split if numeric.
`...`	Ignored

Value

A ggplot object

`plot.switch_data` *Plot mean switch-from-initial-AOI times.*

Description

Boxplot of mean switch time aggregated by subjects within each FirstAOI, potentially faceted by predictor_columns.

Usage

```
## S3 method for class 'switch_data'
plot(x, predictor_columns = NULL, ...)
```

Arguments

`x`	The output of the `make_switch_data` function
`predictor_columns`	Column(s) by which to facet the data. Maximum two columns. Will perform median split if numeric.
`...`	Ignored

Value

A ggplot object

---

plot.time_cluster_data

Plot test-statistic for each time-bin in a time-series, highlight clusters. Plot time_cluster_data, highlights clusters of above-threshold time-bins.

Description

Plot test-statistic for each time-bin in a time-series, highlight clusters. Plot time_cluster_data, highlights clusters of above-threshold time-bins.

Usage

```
## S3 method for class 'time_cluster_data'
plot(x, type = NULL, ...)
```

Arguments

`x`	The output of `make_time_cluster_data`
`type`	This function can plot the test-statistic ("statistic"), the parameter estimate +/- std. error ("estimate"), the p-value ("pvalue") or the negative-log-pvalue ("neg_log_pvalue"). When test gives critical-statistic, default is to plot the test-statistic. Otherwise, default is to plot the estimate. For wilcox, only p-values can be plotted; for boot-splines, p-values cannot be plotted.
`...`	Ignored

Value

A ggplot object

`plot.time_sequence_data` *Plot time-sequence data*

Description

Plot the timecourse of looking. Each AOI will be plotted in a separate pane, and data can be split into groups by a predictor column. Data is collapsed by subject for plotting. Supports overlaying the predictions of a growth-curve mixed effects model on the data

Usage

```
## S3 method for class 'time_sequence_data'
plot(x, predictor_column = NULL, dv = "Prop", model = NULL, ...)
```

Arguments

`x`	Your data from `make_time_sequence_data`. Will be collapsed by subject for plotting (unless already collapsed by some other factor).
`predictor_column`	Data can be grouped by a predictor column (median split is performed if numeric)
`dv`	What measure of gaze do you want to use? (`Prop`, `Elog`, or `ArcSin`)
`model`	(Optional) A growth-curve mixed effects model (from `lmer`) that was used on the `time_sequence_data`. If model is given, this function will overlay the predictions of that model on the data
`...`	Ignored

Value

A ggplot object

Examples

```
## Not run:
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE
)
response_time <- make_time_sequence_data(data, time_bin_size = 250,
                                         predictor_columns = c("MCDI_Total"),
                                     aois = "Animate", summarize_by = "ParticipantName")
```

```
# visualize time results
plot(response_time, predictor_column = "MCDI_Total")

## End(Not run)
```

`plot.time_window_data` *Plot a time-window dataset*

Description

Plots the data returned from `make_time_window_data`. Data can be mapped onto (up to two) predictor columns. If no predictor columns are supplied, AOI is placed on the x-axis; otherwise, data for each AOI is set in a separate facet.

Usage

```
## S3 method for class 'time_window_data'
plot(x, predictor_columns = NULL, dv = "Prop", ...)
```

Arguments

`x`	The data returned by make_time_window_data()
`predictor_columns`	Up to two columns indicating predictors. The first maps to the X-axis, the second to group/color. If the latter is numeric, a median split is performed.
`dv`	Which dv should be used in plotting? Raw proportion-looking ("Prop"), empirical logit ("Elog"), or "ArcSin"?
`...`	Ignored

Details

Data are collapsed by-participants for plotting.

Value

A ggplot object

Examples

```
## Not run:
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
```

```
                                aoi_columns = c('Animate','Inanimate'),
                                treat_non_aoi_looks_as_missing = TRUE)
response_window_agg_by_sub <- make_time_window_data(data,
                                                     aois='Animate',
                                                 predictor_columns=c('Age','MCDI_Total'))

plot(response_window_agg_by_sub, predictor_columns="Age", dv = "LogitAdjusted")

## End(Not run)
```

`print.cluster_analysis` *Print Method for Cluster Analysis*

Description

Print Method for Cluster Analysis

Usage

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

Arguments

`x`	The output of the `analyze_clusters` function
`...`	Ignored

Value

Prints information about the bootstrapped null distribution, as well as information about each cluster.

`reclass` *Add the original class/attributes back onto result (usually of dplyr operation)*

Description

Add the original class/attributes back onto result (usually of dplyr operation)

Usage

```
reclass(x, result, ...)
```

Arguments

`x`	The original object, class inforamation you want to restore.
`result`	Some transformation of `x`, which may have removed its class/attributes.
`...`	Ignored

Value

The `result`, now with class/attribute information restored.

`simulate_eyetrackingr_data`

Simulate an eyetrackingR dataset

Description

This function creates an eyetrackingR dataset (i.e., already run through make_eyetrackingr_data). This can be helpful for examining the false-alarm and sensitivity of analysis-techniques via simulations.

Usage

```
simulate_eyetrackingr_data(
  num_participants = 16,
  num_items_per_condition = 6,
  trial_length = 5000,
  pref = 0.5,
  pref_window = c(1, trial_length),
  noisy_window = NULL,
  ...
)
```

Arguments

`num_participants`	Number of participants
`num_items_per_condition`	Number of trials per-subject per-condition.
`trial_length`	How long is the trial (in ms)?
`pref`	Their preference between the two AOIs in the "high" condition, where 1 is 100 preference). In the "low" condition, their preference between the two AOIs is equal, so default is no effect of condition.
`pref_window`	Vector of length two, specifying start and end of time-window in which participants expressed the preference specified in `pref`. Default is the entire trial
`noisy_window`	Vector of length two, specifying start and end of time-window in which there was substantial trackloss during the trial.
`...`	Ignored

Value

Dataframe with eye-tracking data

`subset_by_window` *Extract a subset of the dataset within a time-window in each trial.*

Description

One of the more annoying aspects of preparing raw eyetracking data is filtering data down into the relevant window within the trial, since for many experiments the precise start and end time of this window can vary from trial to trial. This function allows for several approaches to subsetting data into the relevant time- window– see 'Details' below.

Usage

```
subset_by_window(
  data,
  rezero = TRUE,
  remove = TRUE,
  window_start_msg = NULL,
  window_end_msg = NULL,
  msg_col = NULL,
  window_start_col = NULL,
  window_end_col = NULL,
  window_start_time = NULL,
  window_end_time = NULL,
  quiet = FALSE
)
```

Arguments

`data`	Your original dataset
`rezero`	Should the beginning of the window be considered the zero point of the timestamp? Default TRUE
`remove`	Should everything before the beginning and after the end of the window be removed? Default TRUE. If set to FALSE and `rezero` is set to FALSE, an error is thrown (since in this case, the function would not do anything).
`window_start_msg`	For method (1). A message that is present only in the row whose time corresponds to the trial start time. Common for eyetrackers that send a message at trial/stimuli start.
`window_end_msg`	For method (1). A message that is present only in the row whose time corresponds to the trial end time. Common for eyetrackers that send a message at trial-end/keypress/lookaway/etc.
`msg_col`	For method (1). If you are indicating the trial start/end with a message column, this is the name of that column.

`window_start_col`	For method (2). A column that gives the start time for each trial.
`window_end_col`	For method (2). A column that gives the end time for each trial.
`window_start_time`	For method (3). Number indicating a start time that applies to all trials.
`window_end_time`	For method (3). Number indicating an end time that applies to all trials.
`quiet`	Suppress messages? Default FALSE

Details

1. The trial start/end times can be indicated by a message that is sent (e.g., TRIAL_START) in a particular row for each trial. In this case, the timestamp of that row is used.
2. The trial start/end times can be indicated in by a column that specifies trial start/end times for each trial.
3. The trial start/end times can be indicated by the actual start and stop time, the same across all trials (the simplest case).

If you only have a start time but the end time doesn't need adjusting, then leave the end time argument blank; and vice versa.

This function can either rezero your data (the trial start time you select is the new zero-time-point), or not. The former is useful when performing initial data-cleaning (e.g., different trial-starts on each trial, as indicated by a message), and the latter is useful if you want to "zoom in" on a particular portion of your data while keeping obvious the fact that there were other parts of the trial (e.g., an image always appears 5000ms-7000ms in the trial, so for one analysis you are only interested in this portion).

Value

Subsetted data

Examples

```
data("word_recognition")
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE
)

# zoom in to 15500-21000ms
response_window <- subset_by_window(data,
                                    window_start_time = 15500,
                                  window_end_time = 21000, rezero = FALSE, remove = TRUE)

# zoom in to 15500-21000ms and re-zero so timestamps start at 0
response_window <- subset_by_window(data,
```

```
                                    window_start_time = 15500,
                                    window_end_time = 21000,
                                    rezero = TRUE,
                                    remove = TRUE)

# keep all data, but re-zero it
response_window <- subset_by_window(data,
                                    window_start_time = 0,
                                    rezero = TRUE,
                                    remove = FALSE)
```

`summary.bin_analysis` *Summary Method for Time-bin Analysis*

Description

Summary Method for Time-bin Analysis

Usage

```
## S3 method for class 'bin_analysis'
summary(object, ...)
```

Arguments

`object`	The output of the `analyze_time_bins` function
`...`	Ignored

Value

Prints information about each run of statistically significant time-bins, separately for positive and negative

---

`summary.boot_splines_analysis` *Summary Method for Bootstrapped Splines Analysis*

Description

Summary Method for Bootstrapped Splines Analysis

Usage

```
## S3 method for class 'boot_splines_analysis'
summary(object, ...)
```

Arguments

`object`	The output of the `boot_splines_data` function
`...`	Ignored

Value

Prints a list of divergence-times.

`summary.cluster_analysis`
Summary Method for Cluster Analysis

Description

Summary Method for Cluster Analysis

Usage

```
## S3 method for class 'cluster_analysis'
summary(object, ...)
```

Arguments

`object`	The output of the `analyze_clusters` function
`...`	Ignored

Value

Prints information about the bootstrapped null distribution, as well as information about each cluster.

`summary.time_cluster_data`
Summary Method for Cluster Analysis

Description

Summary Method for Cluster Analysis

Usage

```
## S3 method for class 'time_cluster_data'
summary(object, ...)
```

Arguments

`object`	The output of the `analyze_clusters` function
`...`	Ignored

Value

Prints information about the bootstrapped null distribution, as well as information about each cluster.

`trackloss_analysis` *Analyze trackloss.*

Description

Get information on trackloss in your data.

Usage

```
trackloss_analysis(data)
```

Arguments

`data`	The output of `make_eyetrackingr_data`

Value

A dataframe describing trackloss by-trial and by-participant

Examples

```
data(word_recognition)
data <- make_eyetrackingr_data(word_recognition,
                               participant_column = "ParticipantName",
                               trial_column = "Trial",
                               time_column = "TimeFromTrialOnset",
                               trackloss_column = "TrackLoss",
                               aoi_columns = c('Animate','Inanimate'),
                               treat_non_aoi_looks_as_missing = TRUE
)

tl_analysis <- trackloss_analysis(data)
```

`word_recognition` *Data collected in an infant eyetracking study*

Description

Data from a simple 2-alternative forced choice (2AFC) word recognition task administered to 19- and 24-month-olds. On each trial, infants were shown a picture of an animate object (e.g., a horse) and an inanimate object (e.g., a spoon). After inspecting the images, they disappeared and they heard a label referring to one of them (e.g., "The horse is nearby!"). Finally, the objects re-appeared on the screen and they were prompted to look at the target (e.g., "Look at the horse!").

Usage

```
word_recognition
```

Format

A data frame with 53940 rows and 10 variables:

ParticipantName	Uniaue participant ID
Sex	M or F
Age	Age, in months
TrialNum	Unique Trial Number
Trial	Name of item shown on trial (also unique for each participant)
TimeFromTrialOnset	Time within trial
Subphase	Subphase within trial (see above)
TimeFromSubphaseOnset	Time within subphase
AOI	Which AOI are they looking at
Animate	Are they looking at the animate AOI?
Inanimate	Are they looking at the inanimate AOI?
TrackLoss	Does current sample not have valid tracking data?
MCDI_Total	Total vocabulary score on MCDI
MCDI_Nouns	Noun vocabulary score on MCDI
MCDI_Verbs	Verb vocabulary score on MCDI...

Source

Ferguson, B., Graf, E., & Waxman, S. R. (2014). Infants use known verbs to learn novel nouns: Evidence from 15- and 19-month-olds. Cognition, 131(1), 139-146.

Index