Package 'talkr'

May 8, 2026

Type Package

Title Plotting Conversation Data

Version 0.1.3

Description Visualisation, analysis and quality control of conversational data.
Rapid and visual insights into the nature, timing and quality of
time-aligned annotations in conversational corpora.
For more details, see
Dingemanse et al., (2022) <doi:10.18653/v1/2022.acl-long.385>.

License Apache License (>= 2)

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.3.2

Depends R (>= 3.5.0)

Imports cowplot, dplyr, ggplot2, ggthemes, knitr, stats, stringr,
tidyr, tidyselect

Suggests rmarkdown, testthat (>= 3.0.0), pkgdown, ggrepel, utils

Config/testthat/edition 3

NeedsCompilation no

Author Mark Dingemanse [aut, cre],
Barbara Vreede [aut],
Eva Viviani [aut],
Pablo Rodríguez-Sánchez [aut],
Andreas Liesenfeld [ctb],
Netherlands eScience Center [cph, fnd]

Maintainer Mark Dingemanse <mark.dingemanse@ru.nl>

Repository CRAN

Date/Publication 2024-11-19 11:50:03 UTC

Contents

`add_lines` *Add information for line-by-line visualization*

Description

This function adds columns to the dataset that adds a line ID, and changes columns with timestamps relative to the beginning of the line, so data can be visualized line-by-line. The participant column is also adjusted to create a Y-coordinate for each speaker. The line duration is set to 60 seconds by default.

Usage

```
add_lines(data, time_columns = c("begin", "end"), line_duration = 60000)
```

Arguments

`data`	dataset to divide into lines
`time_columns`	columns with timestamps that need to be adjusted to line-relative time
`line_duration`	length of line (in ms)

Details

This transformation can be done for multiple columns with time-stamped data. Use the ‘time_columns‘ argument to supply the names of one or more columns that should be transformed.

Value

data set with added columns: ‘line_id‘, ‘line_participant‘, and ‘line_column‘ for every column in ‘time_columns‘

---

calculate_timing *Calculate conversation properties*

Description

A dataframe is generated with conversation properties related to timing. This data is made for quality control purposes only, and does not contain sophisticated transition calculation methods. For this, we refer to the python package ‘scikit-talk‘.

Usage

```
calculate_timing(data)
```

Arguments

data	talkr data frame

Value

data frame containing the UIDs and calculated columns turn_duration, transition_time

---

check_columns *Check the presence of necessary columns in a dataset*

Description

Check the presence of necessary columns in a dataset

Usage

```
check_columns(data, columns)
```

Arguments

data	dataset to check
columns	a vector of column names that must be present

Value

nothing, but throws an error if a column is missing

check_talkr *Check the presence of talkr-workflow columns in the dataset.*

Description

Uses check_columns() to check for: - begin - end - participant - utterance - source - uid

Usage

```
check_talkr(data)
```

Arguments

data	dataset to check

Details

Verifies that begin and end columns are numeric, and likely indicate milliseconds.

check_time *Verify that timing columns are numeric and likely indicate milliseconds.*

Description

Verify that timing columns are numeric and likely indicate milliseconds.

Usage

```
check_time(column, name)
```

Arguments

column	vector with timing information
name	name of the column

Value

nothing, but throws an error if the column is not numeric and warns if the column may not indicate milliseconds

GeomToken *GeomToken*

Description

GeomToken

GeomTurn

geom_token *Plot individual tokens*

Description

From a separate data frame containing tokenized data, plot individual tokens at their estimated time. Data must be provided separately, and should contain a column with the participant (y) and a column with the time (x).

Usage

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

Arguments

data	A tokenized data frame (see 'tokenize()').
mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used the override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count".

- For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

Value

A ggplot2 layer corresponding to a token

geom_turn *Show turn-taking in visualized conversations*

Description

Show turn-taking in visualized conversations

Usage

```
geom_turn(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  na.rm = FALSE,
  height = 0.5,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by 'ggplot2::aes()'. Requires specification of 'begin' and 'end' of turns. Inherits from the default mapping at the top level of the plot, if 'inherit.aes' is set to 'TRUE' (the default).
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used the override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.

position — A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".
- For more information and other ways to specify the position, see the layer position documentation.

... — Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

na.rm — If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

height — The height of the turn-taking rectangles

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().

Value

A ggplot2 layer corresponding to a turn-taking rectangle

get_ifadv *Get IFADV data*

Description

IFA Dialog Video corpus data Available in the public repository: https://github.com/elpaco-escience/ifadv

Usage

```
get_ifadv(
  source = "https://raw.githubusercontent.com/elpaco-escience/ifadv/csv/data/ifadv.csv"
)
```

Arguments

source	(default = "https://raw.githubusercontent.com/elpaco-escience/ifadv/csv/data/ifadv.csv")

Details

This function requires an internet connection.

Value

A data frame containing the IFADV dataset

init *Initialize a 'talkr' dataset*

Description

From a dataframe object, generate a talkr dataset. This dataset contains columns that are used throughout the talkr infrastructure to visualize conversations and language corpora. Initializing a talkr dataset is the first step in the talkr workflow.

Usage

```
init(
  data,
  source = "source",
  begin = "begin",
  end = "end",
  participant = "participant",
  utterance = "utterance",
  format_timestamps = "ms"
)
```

Arguments

data	A dataframe object
source	The column name identifying the conversation source (e.g. a filename; is used as unique conversation ID). If there are no different sources in the data, set this parameter to 'NULL'.
begin	The column name with the begin time of the utterance (in milliseconds)
end	The column name with the end time of the utterance (in milliseconds)
participant	The column name with the participant who produced the utterance
utterance	The column name with the utterance itself
format_timestamps	The format of the timestamps in the begin and end columns. Default is "ms", which expects milliseconds. '%H:%M:%OS' will format eg. 00:00:00.010 to milliseconds (10). See '?strptime' for more format examples.

Value

A dataframe object with columns needed for the talkr workflow

---

plot_density *Make a density plot of a specific column*

Description

Make a density plot of a specific column

Usage

```
plot_density(
  data,
  colname,
  title = "Density",
  xlab = "value",
  ylab = "density"
)
```

Arguments

data	data frame containing the column
colname	column name for which the density should be plotted
title	plot title
xlab	x-axis label
ylab	y-axis label

Value

recorded plot

plot_quality *Check source quality by plotting timing data*

Description

Check source quality by plotting timing data

Usage

```
plot_quality(data, source = "all", saveplot = FALSE)
```

Arguments

data	talkr data frame
source	source to be checked (default is "all", no source is selected)
saveplot	save plot to file (default is FALSE)

Value

list of recorded plots

plot_scatter *Make a scatter plot of two columns*

Description

Make a scatter plot of two columns

Usage

```
plot_scatter(
  data,
  colname_x,
  colname_y,
  title = "Scatter",
  xlab = "x",
  ylab = "y"
)
```

Arguments

data	data frame containing the columns
colname_x	name of column plotted on x-axis
colname_y	name of column plotted on y-axis
title	plot title
xlab	x-axis label
ylab	y-axis label

Value

recorded plot

`report_stats` *Report corpus-level and conversation-level statistics*

Description

Basic conversation statistics are reported to the console: - Corpus-level statistics, reporting on the dataset as a whole; - Conversation-level statistics, reporting per source.

Usage

```
report_stats(data)
```

Arguments

`data`	talkr dataset

Details

The input for this function must be a ‘talkr‘ dataset, containing the columns ‘source‘, ‘participant‘, ‘begin‘, and ‘end‘. Time stamps in the columns ‘begin‘ and ‘end‘ must be in milliseconds. To easily transform a dataset to a ‘talkr‘ dataset, consult ‘talkr::init()‘.

Value

No return, just prints a summary to the console

`theme_turnPlot` *T heme for the turn plot*

Description

T heme for the turn plot

Usage

```
theme_turnPlot(base_size = 11, base_family = "serif", ticks = TRUE)
```

Arguments

`base_size`	int
`base_family`	chr
`ticks`	bool

Value

ggplot2 custom theme for turn plots

`tokenize` *Generate a token-specific dataframe*

Description

From a dataframe with utterances, generate a dataframe that separates tokens in utterances, and assesses their relative timing. The returned data contains information about the original utterance (‘uid‘), as well as the number of tokens in the utterance (‘nwords‘), and the relative time of the token in the utterance (‘relative_time‘).

Usage

```
tokenize(data, utterancecol = "utterance")
```

Arguments

`data`	a talkr dataset
`utterancecol`	the name of the column containing the clean utterance (defaults to "utterance")

Details

The relative time is calculated with each token in an utterance having an equal duration (the duration of the utterance divided by the number of words), and the first token in the utterance beginning at the beginning of the utterance.

The input column provided with the argument ‘utterancecol‘ is used to generate the tokens. It is advised to provide a version of the utterance that has been cleaned and stripped of special characters. Cleaning is not performed in this function. Spaces are used to separate tokens.

Value

a dataframe with details about each token in the utterance

Index