Package: tidyfun (via r-universe)

May 24, 2026

Title Tidy Functional Data Wrangling and Visualization

Version 0.1.2

Description Represent, visualize, describe and wrangle functional data in tidy data frames, building on the 'tf' package. Provides data types for functional observations that work as columns in data frames, enabling manipulation with 'dplyr' verbs and visualization with 'ggplot2' geoms designed for functional data.

License MIT + file LICENSE

URL https://github.com/tidyfun/tidyfun, https://tidyfun.github.io/tidyfun/

BugReports https://github.com/tidyfun/tidyfun/issues

Depends R (>= 4.1), tf (>= 0.4.0)

Imports cli, dplyr, GGally, ggplot2, grid, pillar, purrr, rlang, tibble, tidyr (>= 1.0.0), tidyselect (>= 1.0.0)

Suggests covr, fda, fdasrvf, gridExtra, janitor, knitr, lme4, maps, mgcv, refund, rmarkdown, scales, testthat (>= 3.0.0), tidyverse, viridisLite

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/pak/sysreqs libicu-dev libssl-dev

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

Date/Publication 2026-04-24 11:18:05 UTC

RemoteUrl https://github.com/tidyfun/tidyfun

RemoteRef HEAD

RemoteSha d21a98b87d3e21d7f53895fef89897cf675ce987

Contents

`+.tf_ggplot` *Add layers to tf_ggplot objects*

Description

Add layers to tf_ggplot objects

Usage

```
## S3 method for class 'tf_ggplot'
e1 + e2
```

Arguments

`e1`	A tf_ggplot object
`e2`	A ggplot2 layer, scale, theme, etc.

Value

A modified `tf_ggplot` object.

autoplot.tf *Autoplot and autolayer methods for* tf *objects*

Description

Convenient plotting methods for tf objects. autoplot() creates a complete spaghetti plot, autolayer() creates a layer that can be added to an existing ggplot2::ggplot() or tf_ggplot().

Usage

```
## S3 method for class 'tf'
autoplot(object, ...)

## S3 method for class 'tf'
autolayer(object, ...)
```

Arguments

object	a tf object
...	passed to ggplot2::geom_line()

Value

A tf_ggplot() object for autoplot(), a ggplot2::layer() object for autolayer().

See Also

Other tidyfun visualization: ggcapellini, gglasagna(), ggspaghetti

Examples

```
library(ggplot2)
f <- tf_rgp(5)
autoplot(f)
ggplot() + autolayer(f)
tf_ggplot() + autolayer(f)
```

chf_df *Congestive heart failure accelerometry data*

Description

Activity data from a study of congestive heart failure (CHF) patients. Data were originally presented in Huang et al. (2019); these data are publicly available, with download information in the paper.

Usage

```
chf_df
```

Format

A tibble with 329 rows and 8 columns:

id (numeric) Subject identifier.

gender (character) `"Male"` or `"Female"`.

age (numeric) Age in years.

bmi (numeric) Body mass index.

event_week (numeric) Week of cardiac event.

event_type (character) Type of cardiac event.

day (ordered factor) Day of the week (`Mon` < `Tue` < ... < `Sun`).

activity (`tfd_reg`) Minute-by-minute activity counts over a 24-hour period (arg domain 1–1440).

Source

Data are from a study of physical activity in CHF patients conducted by Huang et al. The original data are publicly available; see the paper for download details.

References

Huang, L., Bai, J., Ivanescu, A., Harris, T., Maurer, M., Green, P., and Zipunnikov, V. (2019). Multilevel Matrix-Variate Analysis and its Application to Accelerometry-Measured Physical Activity in Clinical Populations. *Journal of the American Statistical Association*, 114(526), 553–564. doi:10.1080/01621459.2018.1482750

See Also

dti_df for another example dataset, `vignette("x04_Visualization", package = "tidyfun")` for usage examples.

Other tidyfun datasets: `dti_df`

Examples

```
chf_df

library(ggplot2)
chf_df |>
  dplyr::filter(id %in% 1:5) |>
  gglasagna(activity, order_by = mean)
```

`dti_df` *Diffusion tensor imaging data*

Description

Fractional anisotropy (FA) tract profiles for the corpus callosum (`cca`) and the right corticospinal tract (`rcst`) from a diffusion tensor imaging (DTI) study of multiple sclerosis patients and healthy controls.

The original data in refund::DTI include additional variables (`pasat`, `Nscans`, `visit.time`) that were not carried over here.

Usage

```
dti_df
```

Format

A tibble with 382 rows and 6 columns:

id (numeric) Subject identifier.

visit (integer) Visit number.

sex (factor) `"male"` or `"female"`.

case (factor) `"control"` or `"MS"` (multiple sclerosis status).

cca (`tfd_irreg`) FA tract profiles for the corpus callosum (up to 93 evaluation points, domain 0–1).

rcst (`tfd_irreg`) FA tract profiles for the right corticospinal tract (up to 55 evaluation points, domain 0–1).

Details

If you use this data as an example in written work, please include the following acknowledgment: *"The MRI/DTI data were collected at Johns Hopkins University and the Kennedy-Krieger Institute."*

Source

Data are from the Johns Hopkins University and the Kennedy-Krieger Institute. Also available in a different format as refund::DTI.

References

Goldsmith, J., Bobb, J., Crainiceanu, C., Caffo, B., and Reich, D. (2011). Penalized Functional Regression. *Journal of Computational and Graphical Statistics*, 20(4), 830–851. doi:10.1198/jcgs.2010.10007

Goldsmith, J., Crainiceanu, C., Caffo, B., and Reich, D. (2012). Longitudinal Penalized Functional Regression for Cognitive Outcomes on Neuronal Tract Measurements. *Journal of the Royal Statistical Society: Series C*, 61(3), 453–469. doi:10.1111/j.14679876.2011.01031.x

See Also

chf_df for another example dataset, `vignette("x04_Visualization", package = "tidyfun")` for usage examples.

Other tidyfun datasets: `chf_df`

Examples

```
dti_df

library(ggplot2)
dti_df |>
  dplyr::filter(visit == 1) |>
  tf_ggplot(aes(tf = cca, color = case)) +
  geom_line(alpha = 0.3)
```

`ggcapellini` *Glyph plots for* `tf` *objects*

Description

Plots a miniature glyph / sparkline for each entry of a `tf`-object. (Capellini are tiny spaghetti – *angel hair* pasta.) Aesthetics `x` and `y` specify the location of the glyphs, the `tf` aesthetic defines their shapes. The aliases `geom_cappelini`, `geom_cappellini`, and `geom_capelini` are also accepted.

Usage

```
stat_capellini(
  mapping = NULL,
  data = NULL,
  geom = "capellini",
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  arg = NULL,
  add_lines = FALSE,
  add_boxes = TRUE,
  width = NULL,
```

```
  height = NULL,
  ...
)

geom_capellini(
  mapping = NULL,
  data = NULL,
  stat = "capellini",
  position = "identity",
  ...,
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  arg = NULL,
  add_lines = TRUE,
  add_boxes = TRUE,
  width = NULL,
  height = NULL,
  box.colour = "#0000001A",
  box.linetype = 1,
  box.fill = NA,
  box.linewidth = 0.1,
  box.alpha = 0.1,
  line.colour = "black",
  line.linetype = 2,
  line.linewidth = 0.3,
  line.alpha = 0.5
)
```

Arguments

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.
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)).
geom	The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint.

- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`.
- For more information and other ways to specify the geom, see the layer geom 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.

`na.rm` remove NAs? defaults to `TRUE`

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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. `annotation_borders()`.

`arg` where to evaluate `tf`; defaults to the object's default evaluation grid.

`add_lines` should a reference line in the middle of the range of the functions' values be added to each glyph? defaults to TRUE

`add_boxes` should a box be added to frame each glyph? defaults to TRUE

`width` the width of the glyphs. Defaults to 2/3 of the `ggplot2::resolution()` of the variable for the `x`-aesthetic, this will be too small if any values are close together.

`height` the height of the glyphs. Defaults to 2/3 of the `ggplot2::resolution()` of the variable for the `y`-aesthetic, this will be too small if any values are close together.

`...` 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.

`stat`	the stat to use; defaults to `"capellini"`.
`box.colour`	aesthetic property of the box
`box.linetype`	aesthetic property of the box
`box.fill`	aesthetic property of the box
`box.linewidth`	aesthetic property of the box
`box.alpha`	aesthetic property of the box
`line.colour`	aesthetic property of the reference line
`line.linetype`	aesthetic property of the reference line
`line.linewidth`	aesthetic property of the reference line
`line.alpha`	aesthetic property of the reference line

Value

A `ggplot2::layer()` object for use in a ggplot.

`tf` aesthetic

Mandatory. Used to designate a column of class `tf` to be visualized as glyphs.

See Also

`GGally::glyphs()`

Other tidyfun visualization: `autoplot.tf()`, `gglasagna()`, `ggspaghetti`

Examples

```
library(ggplot2)
weather <- fda::CanadianWeather
canada <- data.frame(
  place = weather$place,
  region = weather$region,
  lat = weather$coordinates[, 1],
  lon = -weather$coordinates[, 2],
  region = weather$region
)
```

```
canada$temp <- tfd(t(weather$dailyAv[, , 1]), arg = 1:365)
canada$precipl10 <- tfd(t(weather$dailyAv[, , 3]), arg = 1:365) |> tf_smooth()
canada_map <-
  data.frame(maps::map("world", "Canada", plot = FALSE)[c("x", "y")])
# map of canada with annual temperature averages in red, precipitation in blue:
ggplot(canada, aes(x = lon, y = lat)) +
  geom_capellini(aes(tf = precipl10), width = 3, height = 5, colour = "blue") +
  geom_capellini(aes(tf = temp), width = 3, height = 5, colour = "red") +
  geom_path(data = canada_map, aes(x = x, y = y), alpha = 0.1) +
  coord_quickmap()

ggplot(canada, aes(x = lon, y = lat, colour = region)) +
  geom_capellini(aes(tf = precipl10),
    width = 5, height = 3,
    line.linetype = 1, box.fill = "white", box.alpha = 0.5, box.colour = NA
  )
```

---

ggerrorband *Error bands using* tf *objects as bounds*

Description

Plots a shaded region between tf-objects ymax and ymin. This is primarily intended to help with plotting confidence bands although other purposes are possible.

Usage

```
stat_errorband(
  mapping = NULL,
  data = NULL,
  geom = "errorband",
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  arg = NULL,
  ...
)

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

Arguments

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.
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)).
geom	The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point". • For more information and other ways to specify the geom, see the layer geom 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.
na.rm	remove NAs? defaults to TRUE
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. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
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. annotation_borders().
arg	where to evaluate tf; defaults to the object's default evaluation grid.

`...` 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.

Value

A `ggplot2::layer()` object for use in a ggplot.

Examples

```
set.seed(1221)
data <- data.frame(id = factor(1:2))
data$f <- tf_rgp(2)
data$ymax <- data$f + 1
data$ymin <- data$f - 1
library(ggplot2)
ggplot(data, aes(y = f, color = id)) +
  geom_spaghetti() +
  geom_errorband(aes(ymax = ymax, ymin = ymin, fill = id)) +
  facet_wrap(~id)
```

---

ggfboxplot *Functional boxplots for* `tf` *objects*

Description

Draw functional boxplots based on functional depth rankings.

Usage

```
stat_fboxplot(
  mapping = NULL,
  data = NULL,
  geom = "fboxplot",
  position = "identity",
  ...,
  orientation = NA,
  coef = 1.5,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  depth = "MBD",
  depth_fn = NULL,
  fence_fn = NULL,
  central = 0.5,
  arg = NULL
)

geom_fboxplot(
  mapping = NULL,
  data = NULL,
  stat = "fboxplot",
  position = "identity",
  ...,
  outliers = TRUE,
  outlier.colour = NULL,
  outlier.color = NULL,
  outlier.fill = NULL,
  outlier.shape = NULL,
  outlier.size = NULL,
  outlier.stroke = 0.5,
  outlier.alpha = NULL,
  whisker.colour = NULL,
  whisker.color = NULL,
  whisker.linetype = NULL,
  whisker.linewidth = NULL,
  staple.colour = NULL,
  staple.color = NULL,
  staple.linetype = NULL,
  staple.linewidth = NULL,
  median.colour = NULL,
  median.color = NULL,
  median.linetype = NULL,
  median.linewidth = NULL,
  box.colour = NULL,
  box.color = NULL,
  box.linetype = NULL,
```

```
  box.linewidth = NULL,
  notch = FALSE,
  notchwidth = 0.5,
  staplewidth = 0,
  varwidth = FALSE,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE,
  depth = "MBD",
  depth_fn = NULL,
  fence_fn = NULL,
  central = 0.5,
  arg = NULL
)
```

Arguments

`mapping`, `data`, `position`, `show.legend`, `inherit.aes`, `na.rm`, `orientation`, `...`	Passed on to `ggplot2::layer()`.
`coef`	Inflation factor for the central envelope used to define outer fences, defaults to 1.5.
`depth`	Character scalar naming the built-in depth measure passed to `tf::tf_depth()` when `depth_fn` is `NULL`.
`depth_fn`	Optional custom depth function. Must return one numeric depth value per function.
`fence_fn`	Optional custom fence function. Must return a list with elements `lower`, `upper`, and `outliers`.
`central`	Fraction of deepest curves used to construct the central envelope. Defaults to `0.5`.
`arg`	Optional evaluation grid used for depth calculation, envelopes, and drawing. Defaults to the natural grid of the mapped `tf`.
`stat`, `geom`	Use the functional boxplot stat/geom.
`outliers`	Should outlying curves be drawn?
`outlier.colour`, `outlier.color`, `outlier.fill`, `outlier.shape`, `outlier.size`	Styling parameters for outlier curves.
`outlier.stroke`	Accepted for interface compatibility with `ggplot2::geom_boxplot()` but ignored because functional outliers are drawn as curves.
`outlier.alpha`	Alpha used for outlier curves.
`whisker.colour`, `whisker.color`, `whisker.linetype`, `whisker.linewidth`	Styling parameters for fence lines.
`staple.colour`, `staple.color`, `staple.linetype`, `staple.linewidth`	Accepted for interface compatibility with `ggplot2::geom_boxplot()` but currently unused.

`median.colour`, `median.color`, `median.linetype`, `median.linewidth`
: Styling parameters for the median curve.

`box.colour`, `box.color`, `box.linetype`, `box.linewidth`
: Styling parameters for the central band outline.

`notch`, `notchwidth`, `staplewidth`, `varwidth`
: Accepted for interface compatibility with `ggplot2::geom_boxplot()` but currently unused.

Details

`stat_fboxplot()` computes a median curve (thick line), a central region envelope (filled ribbon), functional fences (dashed lines), and optional outlying curves (defined as "exceeds the fences somewhere", plotted as solid lines) from a `tf` column. `geom_fboxplot()` draws these summaries as a band plus line layers.

The interface intentionally follows `ggplot2::stat_boxplot()` and `ggplot2::geom_boxplot()` where this is meaningful for functional data. Use `aes(tf = f)` to map a `tf` column. Separate functional boxplots inside a panel are defined by `group`, `colour`/`color`, or `fill`; separate panels are handled through facetting. Note that only `color` or `fill` need to be specified explicitly, as the other will be automatically mapped to the same variable if not provided.

Value

A ggplot2 layer.

Examples

```
library(ggplot2)
set.seed(1312)
data <- data.frame(id = 1:50, grp = rep(c("A", "B"), each = 25))
data$f <- tf_rgp(50) + 5 * (data$grp == "A")

tf_ggplot(data, aes(tf = f, fill = grp)) + # same as `colour = grp` here!
  geom_fboxplot(alpha = 0.3)

tf_ggplot(data, aes(tf = f)) +
  geom_fboxplot(orientation = "y")
```

---

gglasagna *Lasagna plots for* `tf`*s using* `ggplot2`

Description

Lasagna plots show one color bar for each function.

Usage

```
gglasagna(
  data,
  tf,
  order = NULL,
  label = NULL,
  arg = NULL,
  order_by = NULL,
  order_ticks = TRUE
)
```

Arguments

data	A data frame containing the tf column to visualize.
tf	bare name of the tf column to visualize
order	(optional) bare name of a column in data to define vertical order of lasagna layers.
label	(optional) bare name of a column in data to define labels for lasagna layers. Defaults to names of y, if present, or row numbers.
arg	arg to evaluate y on
order_by	a function applied to each row in y[, arg] that must return a scalar value to define the order of lasagna layers.
order_ticks	add horizontal lines indicating borders between levels of order (if it is a discrete variable) and labels for its levels? Defaults to TRUE. Supply a named list to override tick appearance, including label styling, line type, alpha, rotation, and label placement. Disable this when faceting; the tick annotations are not designed for faceted layouts.

Details

The vertical order of the lasagna layers is **increasing** in

- order (if provided),
- the values returned by order_by (if provided),
- and the row number of the observations.

i.e., lowest values are on top so that by default the first layer is the first observation in data and the vertical order of the layers is the ordering of observations obtained by dplyr::arrange(data, order, order_by(value), row_number()).

Value

a ggplot2 object

See Also

Other tidyfun visualization: autoplot.tf(), ggcapellini, ggspaghetti

Examples

```
library(ggplot2)
set.seed(1221)
data <- expand.grid(group = factor(1:5), rep = 1:10)
data <- dplyr::mutate(data,
  id = paste(group, rep, sep = "-"),
  f = tf_rgp(50),
  fb = tfb(f)
)

gglasagna(data, f, label = id)
gglasagna(data, fb, label = id, order = group)
# order is lowest first / on top by default
gglasagna(data, f, label = id, order = tf_depth(f))
gglasagna(data, f, label = id, order_by = dplyr::first) +
  facet_wrap(~group, scales = "free")
# order of layers is by "order_by" within "order":
gglasagna(data, fb, label = id, order = group, order_by = dplyr::first)
```

---

ggplot_build.tf_ggplot

ggplot_build method for tf_ggplot

Description

ggplot_build method for tf_ggplot

Usage

```
## S3 method for class 'tf_ggplot'
ggplot_build(plot, ...)
```

Arguments

plot	A tf_ggplot object
...	Additional arguments passed through from ggplot2::ggplot_build().

Value

A built ggplot object (class ggplot_built).

ggspaghetti *Spaghetti plots for* `tf` *objects*

Description

Plots a line for each entry of a `tf`-object. `geom_spaghetti` draws a line through each function, and `geom_meatballs` adds points for the evaluated grid values.

Usage

```
stat_tf(
  mapping = NULL,
  data = NULL,
  geom = "spaghetti",
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  arg = NULL,
  ...
)

geom_spaghetti(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  arg = NULL,
  ...
)

geom_meatballs(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  arg = NULL,
  spaghetti = TRUE,
  ...
)
```

Arguments

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.
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)).
geom	The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point". • For more information and other ways to specify the geom, see the layer geom 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.
na.rm	remove NAs? defaults to TRUE
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. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
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. annotation_borders().
arg	where to evaluate the functions in y; defaults to the object's default evaluation grid.

`...` 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.

`spaghetti` plot noodles along with meatballs? defaults to TRUE.

Details

"Flipped" aesthetics are not implemented for these geoms.

Value

A `ggplot2::layer()` object for use in a ggplot.

y aesthetic

Mandatory. Used to designate a column of class `tf` to be visualized.

See Also

`geom_cappelini()` for glyph plots, `gglasagna()` for heatmaps.

Other tidyfun visualization: `autoplot.tf()`, `ggcapellini`, `gglasagna()`

Other tidyfun visualization: `autoplot.tf()`, `ggcapellini`, `gglasagna()`

Other tidyfun visualization: `autoplot.tf()`, `ggcapellini`, `gglasagna()`

Other tidyfun visualization: `autoplot.tf()`, `ggcapellini`, `gglasagna()`

Other tidyfun visualization: `autoplot.tf()`, `ggcapellini`, `gglasagna()`

Other tidyfun visualization: `autoplot.tf()`, `ggcapellini`, `gglasagna()`

Other tidyfun visualization: `autoplot.tf()`, `ggcapellini`, `gglasagna()`

Examples

```
set.seed(1221)
data <- data.frame(col = sample(gl(5, 2)))
data$f <- tf_rgp(10)
data$fi <- tf_jiggle(data$f)
data$fb <- tfb(data$f)
library(ggplot2)
ggplot(data, aes(y = f, color = tf_depth(f))) +
  geom_spaghetti()
ggplot(data, aes(y = fi, shape = col, color = col)) +
  geom_meatballs()
ggplot(data, aes(y = fi)) +
  geom_meatballs(spaghetti = FALSE) +
  facet_wrap(~col)
```

---

is_tf_ggplot *Check if object is a tf_ggplot*

Description

Check if object is a tf_ggplot

Usage

```
is_tf_ggplot(x)
```

Arguments

x	Object to test

Value

TRUE if x inherits from "tf_ggplot", FALSE otherwise.

Examples

```
p <- tf_ggplot(data.frame(x = 1))
is_tf_ggplot(p)
is_tf_ggplot(ggplot2::ggplot())
```

parse_tf_aesthetics *Parse aesthetic mappings to separate tf and regular aesthetics*

Description

Parse aesthetic mappings to separate tf and regular aesthetics

Usage

```
parse_tf_aesthetics(mapping, data = NULL)
```

Arguments

mapping	An aesthetic mapping created with ggplot2::aes().
data	Data frame to evaluate expressions against.

Value

A list with components tf_aes, scalar_tf_aes, and regular_aes.

Examples

```
parse_tf_aesthetics(ggplot2::aes(tf = f, color = group))
parse_tf_aesthetics(ggplot2::aes(x = x, y = y))
```

print.tf_ggplot *Print method for tf_ggplot*

Description

Print method for tf_ggplot

Usage

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

Arguments

x	A tf_ggplot object
...	Additional arguments

Value

x, invisibly. Called for its side effect of printing the plot.

`tf_evaluate.data.frame` *Evaluate* `tf`*s inside a* `data.frame`

Description

Evaluate `tf`s inside a `data.frame`

Usage

```
## S3 method for class 'data.frame'
tf_evaluate(object, ..., arg)
```

Arguments

`object`	a `data.frame`-like object with `tf` columns.
`...`	optional: a selection of `tf`-columns. If empty, all `tf`-variables in the data frame are selected. You can supply bare variable names, select all variables between `x` and `z` with `x:z`, exclude `y` with `-y`. For more options, see the `dplyr::select()` documentation.
`arg`	optional evaluation grid (vector or list of vectors). Defaults to `tf_arg(object)`.

Details

The `arg`-argument of `tf_evaluate.data.frame` method can be a list of `arg`-vectors or -lists used as the `arg` argument for the `tf::tf_evaluate()`-method for the respective `tf`-columns in `object`. `...` is not used for a `tf`-`object`, but a second unnamed argument to these methods will be interpreted as `arg`.

Value

Replaces `tf`-columns with list columns of smaller `data.frames` containing the functions' arguments (`arg`) and evaluations (`value`) and returns the modified nested dataframe.

See Also

Other tidyfun data wrangling functions: `tf_gather()`, `tf_nest()`, `tf_spread()`, `tf_unnest()`

Examples

```
d <- dplyr::tibble(id = 1:3)
d$f <- tf_rgp(3, 11L)
str(tf_evaluate(d))
```

tf_gather *Gather all columns representing functional measurements into a* tfd-*object*

Description

Similar in spirit to tidyr::gather(), but does NOT put the values in the gathered columns into one very long "value"-column while labeling the different original columns in a very long "key"-column – instead it creates a tfd-column containing the functional measurements of the columns given in ....

Usage

```
tf_gather(
  data,
  ...,
  key = ".tfd",
  arg = NULL,
  domain = NULL,
  evaluator = tf_approx_linear
)
```

Arguments

data	a data frame – note that dplyr does not handle matrix columns well, if data contains more than one of those, tf_gather will fail...
...	A selection of columns to collect as a tfd object. Each column represents measurements of a functional variable at a specific arg-val. Can also be the name of a matrix-valued column, but see above. If empty, all variables are selected. You can supply bare variable names, select all variables between x and z with x:z, exclude y with -y. For more options, see the dplyr::select() documentation.
key	the name of the created tfd-column. Defaults to ".tfd", but the function will try to guess the name based on the column names of the gathered columns in this case. If a common prefix of all column names is found, this is used instead. You also get a message about this.
arg	optional. Argument values for the functions. If not provided, will be guessed from the column names as well. See also tf::tfd().
domain	optional. Range of possible arg-values. See tf::tfd() for details.
evaluator	optional. A function accepting arguments x, arg, evaluations. See tf::tfd() for details.

Value

a modified data.frame with a tfd column replacing the ....

See Also

dplyr::select()

Other tidyfun data wrangling functions: tf_evaluate.data.frame(), tf_nest(), tf_spread(), tf_unnest()

Examples

```
d <- tf_spread(growth[1:5, ]) |> dplyr::mutate(id = 1:dplyr::n())
dplyr::glimpse(d)
# tidyselect syntax for column selection:
tf_gather(d, starts_with("height"))
tf_gather(d, height_1:height_18)
tf_gather(d, -gender, -id)
# custom key name and arg values:
tf_gather(d, starts_with("height"), key = "height")
tf_gather(d, starts_with("height"), arg = seq(0, 1, length.out = 31))
```

---

tf_ggplot *Create a tf-aware ggplot*

Description

tf_ggplot() creates a ggplot object that can handle tf (functional data) aesthetics. It works similarly to ggplot() but automatically transforms tf objects into long-format data suitable for standard ggplot2 geoms.

Usage

```
tf_ggplot(data = NULL, mapping = aes(), ..., arg = NULL, interpolate = TRUE)
```

Arguments

data	Default dataset to use for plot. If not provided, must be supplied in each layer added to the plot.
mapping	Default list of aesthetic mappings to use for plot. Can include tf-specific aesthetics like tf, tf_x, tf_y, tf_ymin, tf_ymax.
...	Other arguments passed to ggplot2 functions.
arg	Optional. Evaluation grid for tf objects. A numeric vector of arg values, or a single integer specifying the desired grid length (resolved to an equidistant grid over the tf domain). If NULL (default), uses the natural grid of the tf objects.
interpolate	Logical. Should tf objects be interpolated to the evaluation grid? Defaults to TRUE.

Details

`tf_ggplot` supports the following tf-specific aesthetics:

- `tf`: Maps a `tf` object to `y` aesthetic (shorthand for `tf_y`)
- `tf_x`: Maps a `tf` object to `x` aesthetic
- `tf_y`: Maps a `tf` object to `y` aesthetic
- `tf_ymin`: Maps a tf object to ymin aesthetic (for ribbons)
- `tf_ymax`: Maps a tf object to ymax aesthetic (for ribbons)

When tf aesthetics are used, the data is automatically transformed:

- tf objects are evaluated on a common grid
- Each function becomes multiple rows (one per evaluation point)
- Group identifiers are created to maintain function identity
- Non-tf columns are replicated appropriately

Value

A tf_ggplot object that inherits from ggplot

Examples

```
# Basic usage
data <- data.frame(
id = 1:10,
group = sample(c("A", "B"), 10, replace = TRUE)
)
data$f <- tf_rgp(10)

# Method 1: tf aesthetic in constructor
tf_ggplot(data, ggplot2::aes(tf = f, color = group)) + ggplot2::geom_line()

# Method 2: tf aesthetic in geom (equivalent)
tf_ggplot(data) + ggplot2::geom_line(ggplot2::aes(tf = f, color = group))

# Confidence bands
tf_ggplot(data) +
  ggplot2::geom_ribbon(
    ggplot2::aes(tf_ymin = mean(f) - sd(f), tf_ymax = mean(f) + sd(f)),
    alpha = 0.3
  ) +
  ggplot2::geom_line(ggplot2::aes(tf = mean(f)))
```

`tf_nest` *Turn "long" tables into tidy data frames with* `tf`*-objects*

Description

Similar in spirit to `tidyr::nest()`. This turns tables in "long" format, where one column (`.id`) defines the unit of observation, one column (`.arg`) defines the evaluation grids of the functional observations, and other columns (`...`) define the values of the functions at those points into a (much shorter) table containing `tfd`-objects. All other variables are checked for constancy over `.id` and appended as well.

Usage

```
tf_nest(
  data,
  ...,
  .id = "id",
  .arg = "arg",
  domain = NULL,
  evaluator = "tf_approx_linear"
)
```

Arguments

`data`	a data frame
`...`	A selection of columns. If empty, all variables except the `.id` and `.arg` columns are selected. You can supply bare variable names, select all variables between `x` and `z` with `x:z`, exclude `y` with `-y`. For more options, see the `dplyr::select()` documentation.
`.id`	the (bare or quoted) name of the column defining the different observations. Defaults to "id".
`.arg`	the (bare or quoted) name of the column defining the `arg`-values of the observed functions. Defaults to "arg".
`domain`	optional. Range of possible `arg`-values. See `tf::tfd()` for details.
`evaluator`	optional. A function accepting arguments x, arg, evaluations. See `tf::tfd()` for details.

Details

`domain` and `evaluator` can be specified as lists or vectors if you are nesting multiple functional data columns with different properties. Because this interface captures evaluator names as text, supply the evaluator as a string rather than a bare function name.

Value

a data frame with (at least) `.id` and `tfd` columns

See Also

tf::tfd() for details on domain and evaluator.

Other tidyfun data wrangling functions: tf_evaluate.data.frame(), tf_gather(), tf_spread(), tf_unnest()

Examples

```
d <- dplyr::tibble(id = rep(1:3, each = 5), arg = rep(1:5, 3), value = rnorm(15))
tf_nest(d, .id = id, .arg = arg)
```

---

tf_spread

Spread a tf*-column into many columns representing the function evaluations.*

Description

Similar in spirit to tidyr::spread(), but does NOT shorten, just widens the data frame – a tf-column is spread out into many columns containing the functional measurements.

Usage

```
tf_spread(data, value, arg, sep = "_", interpolate = FALSE)
```

Arguments

data	a data frame with at least one tf-column
value	the name of the tf-column to 'spread'/evaluate. You can supply bare variable names etc., see the dplyr::select() documentation. Also works without this if there's only one tf in data, see examples.
arg	(Semi-)optional. A vector of arg-values on which to evaluate the functions. If not provided, uses the default args. Should be specified for tf_irreg, otherwise *all* observed gridpoints are used for *every* function.
sep	separating character used to create column names for the new columns, defaults to "_" for column names "<name of the tf>_<arg-value>". Set to NULL to get column names that only contain the arg-value.
interpolate	interpolate-argument for evaluating the functional data. Defaults to FALSE, i.e., tfds are *not* inter/extrapolated on unobserved arg-values.

Value

a wider dataframe with the tf-column spread out into many columns each containing the functional measurements for one arg-value.

See Also

Other tidyfun data wrangling functions: tf_evaluate.data.frame(), tf_gather(), tf_nest(), tf_unnest()

Examples

```
d <- dplyr::tibble(g = 1:3)
d$f <- tf_rgp(3, 11L)
tf_spread(d, f)
tf_spread(d, -g)
tf_spread(d)
```

`tf_unnest` *Turn (data frames with)* `tf`*-objects / list columns into "long" tables.*

Description

Similar in spirit to `tidyr::unnest()`, the reverse of `tf_nest()`. The `tf`-method simply turns a single `tfd` or `tfb` vector into a "long" `tibble::tibble()`.

Usage

```
tf_unnest(data, cols, arg, interpolate = TRUE, ...)

## S3 method for class 'tf'
tf_unnest(data, cols, arg, interpolate = TRUE, ...)

## S3 method for class 'data.frame'
tf_unnest(
  data,
  cols,
  arg,
  interpolate = TRUE,
  keep_empty = FALSE,
  ptype = NULL,
  names_sep = "_",
  names_repair = "check_unique",
  ...
)
```

Arguments

`data`	a data.frame or a `tf`-object
`cols`	<`tidy-select`> List-columns to unnest. When selecting multiple columns, values from the same row will be recycled to their common size.
`arg`	optional values for the `arg` argument of `tf::tf_evaluate()`
`interpolate`	return function values for `arg`-values not on original grid? Defaults to `TRUE`.
`...`	not used currently

keep_empty
: By default, you get one row of output for each element of the list that you are unchopping/unnesting. This means that if there's a size-0 element (like NULL or an empty data frame or vector), then that entire row will be dropped from the output. If you want to preserve all rows, use keep_empty = TRUE to replace size-0 elements with a single row of missing values.

ptype
: Optionally, a named list of column name-prototype pairs to coerce cols to, overriding the default that will be guessed from combining the individual values. Alternatively, a single empty ptype can be supplied, which will be applied to all cols.

names_sep
: If NULL, the default, the outer names will come from the inner names. If a string, the outer names will be formed by pasting together the outer and the inner column names, separated by names_sep.

names_repair
: Used to check that output data frame has valid names. Must be one of the following options:

  - "minimal": no name repair or checks, beyond basic existence,
  - "unique": make sure names are unique and not empty,
  - "check_unique": (the default), no name repair, but check they are unique,
  - "universal": make the names unique and syntactic
  - a function: apply custom name repair.
  - tidyr_legacy: use the name repair from tidyr 0.8.
  - a formula: a purrr-style anonymous function (see rlang::as_function())

  See vctrs::vec_as_names() for more details on these terms and the strategies used to enforce them.

Details

- Caution: this uses slightly different defaults for names of unnested columns than tidyr::unnest().
- For data.frames, include an ID column with a unique row identifier before unnesting. Without it, arg-value pairs cannot be matched back to their original functions after unnesting.

Value

a "long" data frame with tf-columns expanded into arg, value- columns.

See Also

tf_evaluate.data.frame()

Other tidyfun data wrangling functions: tf_evaluate.data.frame(), tf_gather(), tf_nest(), tf_spread()

Examples

```
d <- dplyr::tibble(id = 1:3)
d$f <- tf_rgp(3, 11L)
tf_unnest(d, f)
```

type_sum.tf *Format tidy functional data for tibbles*

Description

Summarize tidy functional data for tibble

Usage

```
## S3 method for class 'tf'
type_sum(x, ...)

## S3 method for class 'tf'
obj_sum(x)

## S3 method for class 'tf'
pillar_shaft(x, ...)
```

Arguments

x	object of class tf
...	ignored

Details

see pillar::type_sum()

Index