Package 'tidySummarizedExperiment'

April 11, 2023

Type Package

Title Brings SummarizedExperiment to the Tidyverse

Version 1.8.1

Description
tidySummarizedExperiment is an adapter that abstracts the 'SummarizedExperiment' container
in the form of tibble and allows the data manipulation, plotting and nesting using 'tidyverse'

License GPL-3

Depends R (>= 4.0.0), SummarizedExperiment

Imports tibble (>= 3.0.4), dplyr, magrittr, tidyr, ggplot2, rlang,
purrr, lifecycle, methods, plotly, utils, S4Vectors,
tidyselect, ellipsis, vctrs, pillar, stringr, cli, fansi

Suggests BiocStyle, testthat, knitr, markdown

VignetteBuilder knitr

RdMacros lifecycle

Biarch true

biocViews AssayDomain, Infrastructure, RNASeq, DifferentialExpression,
GeneExpression, Normalization, Clustering, QualityControl,
Sequencing, Transcription, Transcriptomics

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

LazyDataCompression xz

URL https://github.com/stemangiola/tidySummarizedExperiment

BugReports https://github.com/stemangiola/tidySummarizedExperiment/issues

git_url https://git.bioconductor.org/packages/tidySummarizedExperiment

git_branch RELEASE_3_16

git_last_commit fe4668c

git_last_commit_date 2023-03-14

Author Stefano Mangiola [aut, cre]

Maintainer Stefano Mangiola <mangiolastefano@gmail.com>

R topics documented:

---

as_tibble	*Coerce lists, matrices, and more to data frames*

Description

[Maturing]

`as_tibble()` turns a SummarizedExperiment existing object into a so-called tibble, a data frame with class `tbl_df`.

Arguments

x	A `SummarizedExperiment`
...	This parameter includes `.subset` that can be set to any tidyselect expression. For example .subset = c(sample, type), or .subset = contains("PC").

Value

A tibble

Examples

```
tidySummarizedExperiment::pasilla %>%
    as_tibble()

tidySummarizedExperiment::pasilla %>%
    as_tibble(.subset = -c(condition, type))
```

---

bind *Efficiently bind multiple data frames by row and column*

Description

This is an efficient implementation of the common pattern of `do.call(rbind, dfs)` or `do.call(cbind, dfs)` for binding many data frames into one.

Arguments

`...`	Data frames to combine.
	Each argument can either be a data frame, a list that could be a data frame, or a list of data frames.
	When row-binding, columns are matched by name, and any missing columns will be filled with NA.
	When column-binding, rows are matched by position, so all data frames must have the same number of rows. To match by value, not position, see [mutate-joins](#).
`.id`	Data frame identifier.
	When `.id` is supplied, a new column of identifiers is created to link each row to its original data frame. The labels are taken from the named arguments to `bind_rows()`. When a list of data frames is supplied, the labels are taken from the names of the list. If no names are found a numeric sequence is used instead.
`add.cell.ids`	from SummarizedExperiment 3.0 A character vector of length(x=c(x, y)). Appends the corresponding values to the start of each objects' cell names.

Details

The output of `bind_rows()` will contain a column if that column appears in any of the inputs.

Value

`bind_rows()` and `bind_cols()` return the same type as the first input, either a data frame, `tbl_df`, or `grouped_df`.

Examples

```
`%>%` <- magrittr::`%>%`
library(tibble)
tt <- tidySummarizedExperiment::pasilla
bind_rows(tt, tt)

num_rows <- nrow(tidySummarizedExperiment::as_tibble(tt))
tt %>% bind_cols(tibble(a=0, num_rows))
```

---

count *Count observations by group*

Description

count() lets you quickly count the unique values of one or more variables: df %>% count(a,
b) is roughly equivalent to df %>% group_by(a, b) %>% summarise(n=n()). count() is paired
with tally(), a lower-level helper that is equivalent to df %>% summarise(n=n()). Supply wt to
perform weighted counts, switching the summary from n=n() to n=sum(wt).

add_count() are add_tally() are equivalents to count() and tally() but use mutate() instead
of summarise() so that they add a new column with group-wise counts.

Usage

```
count(
  x,
  ...,
  wt = NULL,
  sort = FALSE,
  name = NULL,
  .drop = group_by_drop_default(x)
)
```

Arguments

x	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
...	<[data-masking](#)> Variables to group by.
wt	<[data-masking](#)> Frequency weights. Can be NULL or a variable:
	• If NULL (the default), counts the number of rows in each group.
	• If a variable, computes sum(wt) for each group.
sort	If TRUE, will show the largest groups at the top.
name	The name of the new column in the output.
	If omitted, it will default to n. If there's already a column called n, it will error, and require you to specify the name.
.drop	For count(): if FALSE will include counts for empty groups (i.e. for levels of factors that don't exist in the data). Deprecated in add_count() since it didn't actually affect the output.

Value

An object of the same type as .data. count() and add_count() group transiently, so the output
has the same groups as the input.

Examples

```
`%>%` <- magrittr::`%>%`
tidySummarizedExperiment::pasilla %>%

    count(.sample)
```

---

formatting	*Printing tibbles*

Description

[Maturing]

One of the main features of the `tbl_df` class is the printing:

- Tibbles only print as many rows and columns as fit on one screen, supplemented by a summary of the remaining rows and columns.

- Tibble reveals the type of each column, which keeps the user informed about whether a variable is, e.g., `<chr>` or `<fct>` (character versus factor).

Printing can be tweaked for a one-off call by calling `print()` explicitly and setting arguments like `n` and `width`. More persistent control is available by setting the options described below.

Arguments

x	Object to format or print.
...	Other arguments passed on to individual methods.
n	Number of rows to show. If `NULL`, the default, will print all rows if less than option `tibble.print_max`. Otherwise, will print `tibble.print_min` rows.
width	Width of text output to generate. This defaults to `NULL`, which means use `getOption("tibble.width")` or (if also `NULL`) `getOption("width")`; the latter displays only the columns that fit on one screen. You can also set `options(tibble.width = Inf)` to override this default and always print all columns.
n_extra	Number of extra columns to print abbreviated information for, if the width is too small for the entire tibble. If `NULL`, the default, will print information about at most `tibble.max_extra_cols` extra columns.

Value

Nothing

Package options

The following options are used by the tibble and pillar packages to format and print `tbl_df` objects. Used by the formatting workhorse `trunc_mat()` and, therefore, indirectly, by `print.tbl()`.

- `tibble.print_max`: Row number threshold: Maximum number of rows printed. Set to `Inf` to always print all rows. Default: 20.
- `tibble.print_min`: Number of rows printed if row number threshold is exceeded. Default: 10.
- `tibble.width`: Output width. Default: `NULL` (use `width` option).
- `tibble.max_extra_cols`: Number of extra columns printed in reduced form. Default: 100.

Examples

```
library(dplyr)
pasilla  %>% print()
```

---

ggplot   *Create a new ggplot from a tidySummarizedExperiment object*

Description

`ggplot()` initializes a ggplot object. It can be used to declare the input data frame for a graphic and to specify the set of plot aesthetics intended to be common throughout all subsequent layers unless specifically overridden.

Arguments

`.data`	Default dataset to use for plot. If not already a data.frame, will be converted to one by [`fortify()`](). If not specified, must be supplied in each layer added to the plot.
`mapping`	Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot.
`...`	Other arguments passed on to methods. Not currently used.
`environment`	DEPRECATED. Used prior to tidy evaluation.

Details

`ggplot()` is used to construct the initial plot object, and is almost always followed by + to add component to the plot. There are three common ways to invoke `ggplot()`:

The first method is recommended if all layers use the same data and the same set of aesthetics, although this method can also be used to add a layer using data from another data frame. See the first example below. The second method specifies the default data frame to use for the plot, but no aesthetics are defined up front. This is useful when one data frame is used predominantly as layers are added, but the aesthetics may vary from one layer to another. The third method initializes a skeleton `ggplot` object which is fleshed out as layers are added. This method is useful when multiple data frames are used to produce different layers, as is often the case in complex graphics.

Value

A ggplot

Examples

```
library(ggplot2)

tidySummarizedExperiment::pasilla %>%

    tidySummarizedExperiment::ggplot(aes(sample, counts)) +
    geom_boxplot()
```

---

pasilla	*Read counts of RNA-seq samples of Pasilla knock-down by Brooks et al.*

Description

A SummarizedExperiment dataset containing the transcriptome information for Drosophila Melanogaster.

Usage

```
data(pasilla)
```

Format

containing 14599 features and 7 biological replicates.

Source

https://bioconductor.org/packages/release/data/experiment/html/pasilla.html

---

plot_ly	*Initiate a plotly visualization*

Description

This function maps R objects to [plotly.js](), an (MIT licensed) web-based interactive charting library. It provides abstractions for doing common things (e.g. mapping data values to fill colors (via color) or creating [animation]s (via frame)) and sets some different defaults to make the interface feel more 'R-like' (i.e., closer to [plot()]() and [ggplot2::qplot()]()).

Usage

```
plot_ly(
  data = data.frame(),
  ...,
  type = NULL,
  name = NULL,
  color = NULL,
  colors = NULL,
  alpha = NULL,
  stroke = NULL,
  strokes = NULL,
  alpha_stroke = 1,
  size = NULL,
  sizes = c(10, 100),
  span = NULL,
  spans = c(1, 20),
  symbol = NULL,
  symbols = NULL,
  linetype = NULL,
  linetypes = NULL,
  split = NULL,
  frame = NULL,
  width = NULL,
  height = NULL,
  source = "A"
)
```

Arguments

data	A data frame (optional) or crosstalk::SharedData object.
...	Arguments (i.e., attributes) passed along to the trace type. See schema() for a list of acceptable attributes for a given trace type (by going to traces -> type -> attributes). Note that attributes provided at this level may override other arguments (e.g. plot_ly(x=1:10, y=1:10, color=I("red"), marker=list(color="blue"))).
type	A character string specifying the trace type (e.g. "scatter", "bar", "box", etc). If specified, it *always* creates a trace, otherwise
name	Values mapped to the trace's name attribute. Since a trace can only have one name, this argument acts very much like split in that it creates one trace for every unique value.
color	Values mapped to relevant 'fill-color' attribute(s) (e.g. fillcolor, marker.color, textfont.color, etc.). The mapping from data values to color codes may be controlled using colors and alpha, or avoided altogether via I() (e.g., color=I("red")). Any color understood by grDevices::col2rgb() may be used in this way.
colors	Either a colorbrewer2.org palette name (e.g. "YlOrRd" or "Blues"), or a vector of colors to interpolate in hexadecimal "#RRGGBB" format, or a color interpolation function like colorRamp().

alpha	A number between 0 and 1 specifying the alpha channel applied to color. Defaults to 0.5 when mapping to [fillcolor](#) and 1 otherwise.
stroke	Similar to color, but values are mapped to relevant 'stroke-color' attribute(s) (e.g., [marker.line.color](#) and [line.color](#) for filled polygons). If not specified, stroke inherits from color.
strokes	Similar to colors, but controls the stroke mapping.
alpha_stroke	Similar to alpha, but applied to stroke.
size	(Numeric) values mapped to relevant 'fill-size' attribute(s) (e.g., [marker.size](#), [textfont.size](#), and [error_x.width](#)). The mapping from data values to symbols may be controlled using sizes, or avoided altogether via [I()](#) (e.g., size=I(30)).
sizes	A numeric vector of length 2 used to scale size to pixels.
span	(Numeric) values mapped to relevant 'stroke-size' attribute(s) (e.g., [marker.line.width](#), [line.width](#) for filled polygons, and [error_x.thickness](#)) The mapping from data values to symbols may be controlled using spans, or avoided altogether via [I()](#) (e.g., span=I(30)).
spans	A numeric vector of length 2 used to scale span to pixels.
symbol	(Discrete) values mapped to [marker.symbol](#). The mapping from data values to symbols may be controlled using symbols, or avoided altogether via [I()](#) (e.g., symbol=I("pentagon")). Any [pch](#) value or [symbol name](#) may be used in this way.
symbols	A character vector of [pch](#) values or [symbol names](#).
linetype	(Discrete) values mapped to [line.dash](#). The mapping from data values to symbols may be controlled using linetypes, or avoided altogether via [I()](#) (e.g., linetype=I("dash")). Any lty (see [par](#)) value or [dash name](#) may be used in this way.
linetypes	A character vector of lty values or [dash names](#)
split	(Discrete) values used to create multiple traces (one trace per value).
frame	(Discrete) values used to create animation frames.
width	Width in pixels (optional, defaults to automatic sizing).
height	Height in pixels (optional, defaults to automatic sizing).
source	a character string of length 1. Match the value of this string with the source argument in [event_data()](#) to retrieve the event data corresponding to a specific plot (shiny apps can have multiple plots).

Details

Unless type is specified, this function just initiates a plotly object with 'global' attributes that are passed onto downstream uses of [add_trace()](#) (or similar). A [formula](#) must always be used when referencing column name(s) in data (e.g. plot_ly(mtcars, x=~wt)). Formulas are optional when supplying values directly, but they do help inform default axis/scale titles (e.g., plot_ly(x=mtcars$wt) vs plot_ly(x=~mtcars$wt))

Value

A plotly

Author(s)

Carson Sievert

References

https://plotly-r.com/overview.html

See Also

- For initializing a plotly-geo object: `plot_geo()`
- For initializing a plotly-mapbox object: `plot_mapbox()`
- For translating a ggplot2 object to a plotly object: `ggplotly()`
- For modifying any plotly object: `layout()`, `add_trace()`, `style()`
- For linked brushing: `highlight()`
- For arranging multiple plots: `subplot()`, `crosstalk::bscols()`
- For inspecting plotly objects: `plotly_json()`
- For quick, accurate, and searchable plotly.js reference: `schema()`

Examples

```
# Plotly better not run
print("See below examples")

## Not run:
# plot_ly() tries to create a sensible plot based on the information you
# give it. If you don't provide a trace type, plot_ly() will infer one.
plot_ly(economics, x=~pop)
plot_ly(economics, x=~date, y=~pop)
# plot_ly() doesn't require data frame(s), which allows one to take
# advantage of trace type(s) designed specifically for numeric matrices
plot_ly(z=~volcano)
plot_ly(z=~volcano, type="surface")

# plotly has a functional interface: every plotly function takes a plotly
# object as it's first input argument and returns a modified plotly object
add_lines(plot_ly(economics, x=~date, y=~ unemploy / pop))

# To make code more readable, plotly imports the pipe operator from magrittr
economics %>%
    plot_ly(x=~date, y=~ unemploy / pop) %>%
    add_lines()

# Attributes defined via plot_ly() set 'global' attributes that
# are carried onto subsequent traces, but those may be over-written
plot_ly(economics, x=~date, color=I("black")) %>%
    add_lines(y=~uempmed) %>%
    add_lines(y=~psavert, color=I("red"))

# Attributes are documented in the figure reference -> https://plot.ly/r/reference
```

```
# You might notice plot_ly() has named arguments that aren't in this figure
# reference. These arguments make it easier to map abstract data values to
# visual attributes.
p <- plot_ly(iris, x=~Sepal.Width, y=~Sepal.Length)
add_markers(p, color=~Petal.Length, size=~Petal.Length)
add_markers(p, color=~Species)
add_markers(p, color=~Species, colors="Set1")
add_markers(p, symbol=~Species)
add_paths(p, linetype=~Species)

## End(Not run)
```

---

se   *Read counts of RNA-seq samples derived from Pasilla knock-down by*
                            Brooks et al.

Description

A SummarizedExperiment dataset containing the transcriptome information for Drosophila Melanogaster.

Usage

```
data(se)
```

Format

containing 14599 features and 7 biological replicates.

Source

<https://bioconductor.org/packages/release/data/experiment/html/pasilla.html>

---

tbl_format_header   *Format the header of a tibble*

Description

[Experimental]

For easier customization, the formatting of a tibble is split into three components: header, body, and footer. The tbl_format_header() method is responsible for formatting the header of a tibble.

Override this method if you need to change the appearance of the entire header. If you only need to change or extend the components shown in the header, override or extend [tbl_sum()](tbl_sum()) for your class which is called by the default method.

---

tidy   *tidy for SummarizedExperiment*

Description

DEPRECATED. Not needed any more.

Usage

```
tidy(object)
```

Arguments

object   A SummarizedExperiment object

Value

A tidySummarizedExperiment object

Examples

```
tidySummarizedExperiment::pasilla %>% tidy()
```

---

unnest   *unnest*

Description

Given a regular expression with capturing groups, extract() turns each group into a new column.
If the groups don't match, or the input is NA, the output will be NA.

pivot_wider() "widens" data, increasing the number of columns and decreasing the number of
rows. The inverse transformation is pivot_longer().

Learn more in vignette("pivot").

Convenience function to paste together multiple columns into one.

Given either a regular expression or a vector of character positions, separate() turns a single
character column into multiple columns.

Arguments

cols	<[tidy-select](#)> Columns to unnest. If you unnest() multiple columns, parallel entries must be of compatible sizes, i.e. they're either equal or length 1 (following the standard tidyverse recycling rules).
keep_empty	See tidyr::unnest
ptype	See tidyr::unnest
.drop	See tidyr::unnest
.id	tidyr::unnest
.sep	tidyr::unnest
.preserve	See tidyr::unnest
.data	A tbl. (See tidyr)
.names_sep	See ?tidyr::nest
into	Names of new variables to create as character vector. Use NA to omit the variable in the output.
regex	a regular expression used to extract the desired values. There should be one group (defined by ()) for each element of into.
convert	If TRUE, will run [type.convert()](#) with as.is=TRUE on new columns. This is useful if the component columns are integer, numeric or logical.
	NB: this will cause string "NA"s to be converted to NAs.
names_to	A character vector specifying the new column or columns to create from the information stored in the column names of data specified by cols.

- If length 0, or if NULL is supplied, no columns will be created.
- If length 1, a single column will be created which will contain the column names specified by cols.
- If length >1, multiple columns will be created. In this case, one of names_sep or names_pattern must be supplied to specify how the column names should be split. There are also two additional character values you can take advantage of:
  - NA will discard the corresponding component of the column name.
  - ".value" indicates that the corresponding component of the column name defines the name of the output column containing the cell values, overriding values_to entirely.

names_sep, names_pattern

If names_to contains multiple values, these arguments control how the column name is broken up.

names_sep takes the same specification as [separate()](#), and can either be a numeric vector (specifying positions to break on), or a single string (specifying a regular expression to split on).

names_pattern takes the same specification as [extract()](#), a regular expression containing matching groups (()).

If these arguments do not give you enough control, use pivot_longer_spec() to create a spec object and process manually as needed.

names_repair   What happens if the output has invalid column names? The default, "check_unique" is to error if the columns are duplicated. Use "minimal" to allow duplicates in the output, or "unique" to de-duplicated by adding numeric suffixes. See [vctrs::vec_as_names()](#) for more options.

values_to   A string specifying the name of the column to create from the data stored in cell values. If names_to is a character containing the special .value sentinel, this value will be ignored, and the name of the value column will be derived from part of the existing column names.

values_drop_na   If TRUE, will drop rows that contain only NAs in the value_to column. This effectively converts explicit missing values to implicit missing values, and should generally be used only when missing values in data were created by its structure.

names_transform, values_transform

                  Optionally, a list of column name-function pairs. Alternatively, a single function can be supplied, which will be applied to all columns. Use these arguments if you need to change the types of specific columns. For example, names_transform = list(week = as.integer) would convert a character variable called week to an integer.

                  If not specified, the type of the columns generated from names_to will be character, and the type of the variables generated from values_to will be the common type of the input columns used to generate them.

names_ptypes, values_ptypes

                  Optionally, a list of column name-prototype pairs. Alternatively, a single empty prototype can be supplied, which will be applied to all columns. A prototype (or ptype for short) is a zero-length vector (like integer() or numeric()) that defines the type, class, and attributes of a vector. Use these arguments if you want to confirm that the created columns are the types that you expect. Note that if you want to change (instead of confirm) the types of specific columns, you should use names_transform or values_transform instead.

id_cols   [<tidy-select>](#) A set of columns that uniquely identify each observation. Typically used when you have redundant variables, i.e. variables whose values are perfectly correlated with existing variables.

                  Defaults to all columns in data except for the columns specified through names_from and values_from. If a tidyselect expression is supplied, it will be evaluated on data after removing the columns specified through names_from and values_from.

id_expand   Should the values in the id_cols columns be expanded by [expand()](#) before pivoting? This results in more rows, the output will contain a complete expansion of all possible values in id_cols. Implicit factor levels that aren't represented in the data will become explicit. Additionally, the row values corresponding to the expanded id_cols will be sorted.

names_from, values_from

                  [<tidy-select>](#) A pair of arguments describing which column (or columns) to get the name of the output column (names_from), and which column (or columns) to get the cell values from (values_from).

                  If values_from contains multiple values, the value will be added to the front of the output column.

names_sep	If `names_from` or `values_from` contains multiple variables, this will be used to join their values together into a single string to use as a column name.
names_prefix	String added to the start of every variable name. This is particularly useful if `names_from` is a numeric vector and you want to create syntactic variable names.
names_glue	Instead of `names_sep` and `names_prefix`, you can supply a glue specification that uses the `names_from` columns (and special `.value`) to create custom column names.
names_sort	Should the column names be sorted? If `FALSE`, the default, column names are ordered by first appearance.
names_vary	When `names_from` identifies a column (or columns) with multiple unique values, and multiple `values_from` columns are provided, in what order should the resulting column names be combined?

- `"fastest"` varies `names_from` values fastest, resulting in a column naming scheme of the form: `value1_name1, value1_name2, value2_name1, value2_name2`. This is the default.
- `"slowest"` varies `names_from` values slowest, resulting in a column naming scheme of the form: `value1_name1, value2_name1, value1_name2, value2_name2`.

names_expand	Should the values in the `names_from` columns be expanded by [expand()](#) before pivoting? This results in more columns, the output will contain column names corresponding to a complete expansion of all possible values in `names_from`. Implicit factor levels that aren't represented in the data will become explicit. Additionally, the column names will be sorted, identical to what `names_sort` would produce.
values_fill	Optionally, a (scalar) value that specifies what each `value` should be filled in with when missing.
	This can be a named list if you want to apply different fill values to different value columns.
values_fn	Optionally, a function applied to the value in each cell in the output. You will typically use this when the combination of `id_cols` and `names_from` columns does not uniquely identify an observation.
	This can be a named list if you want to apply different aggregations to different `values_from` columns.
unused_fn	Optionally, a function applied to summarize the values from the unused columns (i.e. columns not identified by `id_cols`, `names_from`, or `values_from`).
	The default drops all unused columns from the result.
	This can be a named list if you want to apply different aggregations to different unused columns.
	`id_cols` must be supplied for `unused_fn` to be useful, since otherwise all unspecified columns will be considered `id_cols`.
	This is similar to grouping by the `id_cols` then summarizing the unused columns using `unused_fn`.
data	A data frame.

col   The name of the new column, as a string or symbol.

                     This argument is passed by expression and supports [quasiquotation](you can
                     unquote strings and symbols). The name is captured from the expression with
                     [rlang::ensym()](note that this kind of interface where symbols do not repre-
                     sent actual objects is now discouraged in the tidyverse; we support it here for
                     backward compatibility).

...   [<tidy-select>](Columns to unite

na.rm   If TRUE, missing values will be remove prior to uniting each value.

remove   If TRUE, remove input columns from output data frame.

sep   Separator between columns.

                     If character, sep is interpreted as a regular expression. The default value is a
                     regular expression that matches any sequence of non-alphanumeric values.

                     If numeric, sep is interpreted as character positions to split at. Positive values
                     start at 1 at the far-left of the string; negative value start at -1 at the far-right of
                     the string. The length of sep should be one less than into.

extra   If sep is a character vector, this controls what happens when there are too many
                     pieces. There are three valid options:

                        • "warn" (the default): emit a warning and drop extra values.
                        • "drop": drop any extra values without a warning.
                        • "merge": only splits at most length(into) times

fill   If sep is a character vector, this controls what happens when there are not
                     enough pieces. There are three valid options:

                        • "warn" (the default): emit a warning and fill from the right
                        • "right": fill with missing values on the right
                        • "left": fill with missing values on the left

Details

pivot_wider() is an updated approach to [spread()](), designed to be both simpler to use and to
handle more use cases. We recommend you use pivot_wider() for new code; spread() isn't
going away but is no longer under active development.

Value

A tidySummarizedExperiment objector a tibble depending on input

A tidySummarizedExperiment objector a tibble depending on input

A tidySummarizedExperiment objector a tibble depending on input

A tidySummarizedExperiment objector a tibble depending on input

A tidySummarizedExperiment objector a tibble depending on input

A tidySummarizedExperiment objector a tibble depending on input

See Also

separate() to split up by a separator.

pivot_wider_spec() to pivot "by hand" with a data frame that defines a pivotting specification.

separate(), the complement.

unite(), the complement, extract() which uses regular expression capturing groups.

Examples

```
tidySummarizedExperiment::pasilla %>%


    nest(data=-condition) %>%
    unnest(data)


tidySummarizedExperiment::pasilla %>%


    nest(data=-condition)


tidySummarizedExperiment::pasilla %>%


    extract(type, into="sequencing", regex="([a-z]*)_end", convert=TRUE)

# See vignette("pivot") for examples and explanation

library(dplyr)
tidySummarizedExperiment::pasilla %>%


    pivot_longer(c(condition, type), names_to="name", values_to="value")

# See vignette("pivot") for examples and explanation

library(dplyr)
tidySummarizedExperiment::pasilla %>%


    pivot_wider(names_from=feature, values_from=counts)

tidySummarizedExperiment::pasilla %>%


    unite("group", c(condition, type))
```

Index