Package 'tidySingleCellExperiment'

January 16, 2022

Type Package

Title Brings SingleCellExperiment to the Tidyverse

Version 1.4.0

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

License GPL-3

Depends R (>= 4.0.0), SingleCellExperiment

Imports SummarizedExperiment, dplyr, tibble, tidyr, ggplot2, plotly, magrittr, rlang, purrr, lifecycle, methods, utils, S4Vectors, tidyselect, ellipsis, pillar, stringr, cli, fansi

Suggests BiocStyle, testthat, knitr, markdown, SingleCellSignalR, SingleR, scater, scran, tidyHeatmap, igraph, GGally, Matrix, uwot, celldex, dittoSeq, EnsDb.Hsapiens.v86

VignetteBuilder knitr

RdMacros lifecycle

Biarch true

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

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Roxygen list(markdown = TRUE)

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

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

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

git_branch RELEASE_3_14

git_last_commit e143961

git_last_commit_date 2021-10-26
Date/Publication 2022-01-16
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 an existing object, such as a data frame or matrix, into a so-called tibble, a data frame with class `tbl_df`. This is in contrast with `tibble()`, which builds a tibble from individual columns. `as_tibble()` is to `tibble()` as `base::as.data.frame()` is to `base::data.frame()`.

`as_tibble()` is an S3 generic, with methods for:

- `data.frame`: Thin wrapper around the `list` method that implements tibble's treatment of rownames.
- `matrix`, `poly`, `ts`, `table`
- Default: Other inputs are first coerced with `base::as.data.frame()`.

[Maturing]

`glimpse()` is like a transposed version of `print()`: columns run down the page, and data runs across. This makes it possible to see every column in a data frame. It's a little like `str()` applied to a data frame but it tries to show you as much data as possible. (And it always shows the underlying data, even when applied to a remote data source.)

This generic will be moved to **pillar**, and reexported from there as soon as it becomes available.

Arguments

rownames	How to treat existing row names of a data frame or matrix: • NULL: remove row names. This is the default. • NA: keep row names. • A string: the name of a new column. Existing rownames are transferred into this column and the row.names attribute is deleted. Read more in rownames.
.name_repair	see tidyr For compatibility only, do not use for new code.
x	An object to glimpse at.
width	Width of output: defaults to the setting of the option tibble.width (if finite) or the width of the console.
...	Unused, for extensibility.

Value

A tibble

x original x is (invisibly) returned, allowing glimpse() to be used within a data pipe line.

Row names

The default behavior is to silently remove row names.

New code should explicitly convert row names to a new column using the rownames argument.

For existing code that relies on the retention of row names, call pkgconfig::set_config("tibble::rownames"=NA) in your script or in your package's .onLoad() function.

Life cycle

Using as_tibble() for vectors is superseded as of version 3.0.0, prefer the more expressive maturing as_tibble_row() and as_tibble_col() variants for new code.

S3 methods

glimpse is an S3 generic with a customised method for tbls and data.frames, and a default method that calls str().

See Also

tibble() constructs a tibble from individual columns. enframe() converts a named vector to a tibble with a column of names and column of values. Name repair is implemented using vctrs::vec_as_names().

Examples

```
pbmc_small %>%

    as_tibble()
pbmc_small %>% tidy %>% glimpse()
```

`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 SingleCellExperiment 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::`%>%`
tt <- pbmc_small
bind_rows(tt, tt)

tt_bind <- tt %>% select(nCount_RNA, nFeature_RNA)
tt %>% bind_cols(tt_bind)
```

---

cell_type_df *Cell types of 80 PBMC single cells*

Description

A dataset containing the barcodes and cell types of 80 PBMC single cells.

Usage

```
data(cell_type_df)
```

Format

A tibble containing 80 rows and 2 columns. Cells are a subsample of the Peripheral Blood Mononuclear Cells (PBMC) dataset of 2,700 single cell. Cell types were identified with SingleR.

cell cell identifier, barcode

first.labels cell type

Source

https://satijalab.org/seurat/v3.1/pbmc3k_tutorial.html

---

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)
)

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

## Default S3 method:
add_count(
  x,
  ...,
  wt = NULL,
  sort = FALSE,
  name = NULL,
  .drop = group_by_drop_default(x)
)

## S3 method for class 'SingleCellExperiment'
add_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::`%>%`
pbmc_small %>%

    count(groups)
```

---

ggplot *Create a new ggplot from a tidySingleCellExperiment 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)

tidySingleCellExperiment::pbmc_small %>%

    tidySingleCellExperiment::ggplot(aes(groups, nCount_RNA)) +
    geom_boxplot()
```

---

join_features *Add differential featureion information to a tbl using edgeR.*

Description

join_features() extracts and joins information for specific features

Usage

```
join_features(
  .data,
  features = NULL,
  all = FALSE,
  exclude_zeros = FALSE,
  shape = "long",
  ...
)
```

Arguments

.data	A tidy SingleCellExperiment object
features	A vector of feature identifiers to join
all	If TRUE return all
exclude_zeros	If TRUE exclude zero values
shape	Format of the returned table "long" or "wide"
...	Parameters to pass to join wide, i.e. assay name to extract feature abundance from and gene prefix, for shape="wide"

Details

[Experimental]

This function extracts information for specified features and returns the information in either long or wide format.

Value

A `tbl` containing the information.for the specified features

Examples

```
tidySingleCellExperiment::pbmc_small %>%

    join_features(features=c("HLA-DRA", "LYZ"))
```

`join_transcripts` *(DEPRECATED) Extract and join information for transcripts.*

Description

join_transcripts() extracts and joins information for specified transcripts

Usage

```
join_transcripts(
  .data,
  transcripts = NULL,
  all = FALSE,
  exclude_zeros = FALSE,
  shape = "long",
  ...
)
```

Arguments

`.data`	A tidyseurat object
`transcripts`	A vector of transcript identifiers to join
`all`	If TRUE return all
`exclude_zeros`	If TRUE exclude zero values
`shape`	Format of the returned table "long" or "wide"
`...`	Parameters to pass to join wide, i.e. assay name to extract transcript abundance from

Details

DEPRECATED, please use join_features()

Value

A `tbl` containing the information.for the specified transcripts

Examples

```
print("DEPRECATED")
```

`pbmc_small` *pbmc_small*

Description

PBMC single cell RNA-seq data in SingleCellExperiment format

Usage

```
data(pbmc_small)
```

Format

A SingleCellExperiment object containing 80 Peripheral Blood Mononuclear Cells (PBMC) from 10x Genomics. Generated by subsampling the PBMC dataset of 2,700 single cells.

Source

https://satijalab.org/seurat/v3.1/pbmc3k_tutorial.html

`pbmc_small_nested_interactions`
Intercellular ligand-receptor interactions for 38 ligands from a single cell RNA-seq cluster.

Description

A dataset containing ligand-receptor interactions withibn a sample. There are 38 ligands from a single cell cluster versus 35 receptors in 6 other clusters.

Usage

```
data(pbmc_small_nested_interactions)
```

Format

A tibble containing 100 rows and 9 columns. Cells are a subsample of the PBMC dataset of 2,700 single cells. Cell interactions were identified with SingleCellSignalR.

sample sample identifier

ligand cluster and ligand identifier

receptor cluster and receptor identifier

ligand.name ligand name

receptor.name receptor name

origin cluster containing ligand

destination cluster containing receptor

interaction.type type of interation, paracrine or autocrine

LRscore interaction score

Source

https://satijalab.org/seurat/v3.1/pbmc3k_tutorial.html

---

pivot_longer *Pivot data from wide to long*

Description

[Maturing]

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

Learn more in `vignette("pivot")`.

Arguments

`data`	A data frame to pivot.
`cols`	<tidy-select> Columns to pivot into longer format.
`names_to`	A string specifying the name of the column to create from the data stored in the column names of `data`. Can be a character vector, creating multiple columns, if `names_sep` or `names_pattern` is provided. In this case, there are two special values you can take advantage of: • `NA` will discard that component of the name. • `.value` indicates that component of the name defines the name of the column containing the cell values, overriding `values_to`.
`names_prefix`	A regular expression used to remove matching text from the start of each variable name.

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
: A list of column name-function pairs. Use these arguments if you need to change the type of specific columns. For example, names_transform=list(week=as.integer) would convert a character week variable to an integer.

names_ptypes, values_ptypes
: A list of column name-prototype pairs. 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 to confirm that the created columns are the types that you expect.

  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.

...
: Additional arguments passed on to methods.

Details

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

Value

A tidySingleCellExperiment objector a tibble depending on input

Examples

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

library(dplyr)
pbmc_small %>%

    pivot_longer(c(orig.ident, groups), names_to="name", values_to="value")
```

---

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 animations (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

```
## 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)
```

`print` *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.

Only the first 5 reduced dimensions are displayed, while all of them are queriable (e.g. ggplot). All dimensions are returned/displayed if as_tibble is used.

Usage

```
## S3 method for class 'SingleCellExperiment'
print(x, ..., n = NULL, width = NULL, n_extra = NULL)
```

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.
- `pillar.bold`: Use bold font, e.g. for column headers? This currently defaults to `FALSE`, because many terminal fonts have poor support for bold fonts.
- `pillar.subtle`: Use subtle style, e.g. for row numbers and data types? Default: `TRUE`.
- `pillar.subtle_num`: Use subtle style for insignificant digits? Default: `FALSE`, is also affected by the `pillar.subtle` option.

- `pillar.neg`: Highlight negative numbers? Default: `TRUE`.
- `pillar.sigfig`: The number of significant digits that will be printed and highlighted, default: `3`. Set the `pillar.subtle` option to `FALSE` to turn off highlighting of significant digits.
- `pillar.min_title_chars`: The minimum number of characters for the column title, default: `15`. Column titles may be truncated up to that width to save horizontal space. Set to `Inf` to turn off truncation of column titles.
- `pillar.min_chars`: The minimum number of characters wide to display character columns, default: `0`. Character columns may be truncated up to that width to save horizontal space. Set to `Inf` to turn off truncation of character columns.
- `pillar.max_dec_width`: The maximum allowed width for decimal notation, default 13.

Examples

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

`tidy` *tidy for SingleCellExperiment*

Description

tidy for SingleCellExperiment

Usage

```
tidy(object)
```

Arguments

`object`	A SingleCellExperiment object

Value

A tidySingleCellExperiment object

Examples

```
tidySingleCellExperiment::pbmc_small
```

unite *Unite multiple columns into one by pasting strings together*

Description

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

`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 represent 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

Value

A tidySingleCellExperiment objector a tibble depending on input

A tidySingleCellExperiment objector a tibble depending on input

See Also

separate(), the complement.

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

Examples

```
pbmc_small %>%

    unite("new_col", c(orig.ident, groups))

un <- pbmc_small %>%

    unite("new_col", c(orig.ident, groups))
un %>% separate(col=new_col, into=c("orig.ident", "groups"))
```

---

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.

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).
names_sep	If NULL, the default, the names will be left as is. In nest(), inner names will come from the former outer names; in unnest(), the new outer names will come from the inner names. If a string, the inner and outer names will be used together. In nest(), the names of the new outer columns will be formed by pasting together the outer and the inner column names, separated by names_sep. In unnest(), the new inner names will have the outer names (+ names_sep) automatically stripped. This makes names_sep roughly symmetric between nesting and unnesting.
keep_empty	See tidyr::unnest
names_repair	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
data	A tidySingleCellExperiment object
col	Column name or position. This is passed to tidyselect::vars_pull(). This argument is passed by expression and supports quasiquotation (you can unquote column names or column positions).
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.
remove	If TRUE, remove input column from output data frame.
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.
...	Additional arguments passed on to methods.

Value

A tidySingleCellExperiment objector a tibble depending on input

A tidySingleCellExperiment objector a tibble depending on input

A tidySingleCellExperiment objector a tibble depending on input

See Also

separate() to split up by a separator.

Examples

```
library(dplyr)
pbmc_small %>%

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


library(dplyr)
pbmc_small %>%

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

pbmc_small %>%

    extract(groups, into="g", regex="g([0-9])", convert=TRUE)
```

Index