Package 'tidySingleCellExperiment'

October 16, 2023

Type Package

Title Brings SingleCellExperiment to the Tidyverse

Version 1.10.0

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

License GPL-3

Depends R (>= 4.1.0), ttservice, SingleCellExperiment

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

Suggests BiocStyle, testthat, knitr, markdown, SingleCellSignalR, SingleR, scater, scran, tidyHeatmap, igraph, GGally, 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 7.2.3

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_17

git_last_commit d180646

git_last_commit_date 2023-04-25

Date/Publication 2023-10-15

Author Stefano Mangiola [aut, cre]

Maintainer Stefano Mangiola <mangiolastefano@gmail.com>

R topics documented:

`add_class` *Add class to abject*

Description

Add class to abject

Usage

```
add_class(var, name)
```

Arguments

var	A tibble
name	A character name of the attribute

Value

A tibble with an additional attribute

---

aggregate_cells *Aggregate cells*

Description

Combine cells into groups based on shared variables and aggregate feature counts.

Usage

```
aggregate_cells(
  .data,
  .sample = NULL,
  slot = "data",
  assays = NULL,
  aggregation_function = rowSums
)
```

Arguments

.data	A tidySingleCellExperiment object
.sample	A vector of variables by which cells are aggregated
slot	The slot to which the function is applied
assays	The assay to which the function is applied
aggregation_function	The method of cell-feature value aggregation

Value

A SummarizedExperiment object

Examples

```
data("pbmc_small")
pbmc_small_pseudo_bulk <- pbmc_small |>
  aggregate_cells(c(groups, ident), assays = "counts")
```

arrange *Arrange rows by column values*

Description

`arrange()` order the rows of a data frame rows by the values of selected columns.

Unlike other dplyr verbs, `arrange()` largely ignores grouping; you need to explicit mention grouping variables (or use `by_group=TRUE`) in order to group by them, and functions of variables are evaluated once per data frame, not once per group.

`filter()` retains the rows where the conditions you provide a `TRUE`. Note that, unlike base subsetting with `[`, rows where the condition evaluates to `NA` are dropped.

Most data operations are done on groups defined by variables. `group_by()` takes an existing tbl and converts it into a grouped tbl where operations are performed "by group". `ungroup()` removes grouping.

`summarise()` creates a new data frame. It will have one (or more) rows for each combination of grouping variables; if there are no grouping variables, the output will have a single row summarising all observations in the input. It will contain one column for each grouping variable and one column for each of the summary statistics that you have specified.

`summarise()` and `summarize()` are synonyms.

`mutate()` adds new variables and preserves existing ones; `transmute()` adds new variables and drops existing ones. New variables overwrite existing variables of the same name. Variables can be removed by setting their value to `NULL`.

Rename individual variables using `new_name=old_name` syntax.

See this repository for alternative ways to perform row-wise operations.

`slice()` lets you index rows by their (integer) locations. It allows you to select, remove, and duplicate rows. It is accompanied by a number of helpers for common use cases:

- `slice_head()` and `slice_tail()` select the first or last rows.
- `slice_sample()` randomly selects rows.
- `slice_min()` and `slice_max()` select rows with highest or lowest values of a variable.

If `.data` is a grouped_df, the operation will be performed on each group, so that (e.g.) `slice_head(df, n=5)` will select the first five rows in each group.

Select (and optionally rename) variables in a data frame, using a concise mini-language that makes it easy to refer to variables based on their name (e.g. `a:f` selects all columns from `a` on the left to `f` on the right). You can also use predicate functions like is.numeric to select variables based on their properties.

[Superseded] `sample_n()` and `sample_frac()` have been superseded in favour of `slice_sample()`. While they will not be deprecated in the near future, retirement means that we will only perform critical bug fixes, so we recommend moving to the newer alternative.

These functions were superseded because we realised it was more convenient to have two mutually exclusive arguments to one function, rather than two separate functions. This also made it to clean up a few other smaller design issues with `sample_n()`/`sample_frac`:

- The connection to `slice()` was not obvious.
- The name of the first argument, `tbl`, is inconsistent with other single table verbs which use `.data`.
- The `size` argument uses tidy evaluation, which is surprising and undocumented.
- It was easier to remove the deprecated `.env` argument.
- `...` was in a suboptimal position.

`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.

`pull()` is similar to `$`. It's mostly useful because it looks a little nicer in pipes, it also works with remote data frames, and it can optionally name the output.

Usage

```
bind_rows(..., .id = NULL, add.cell.ids = NULL)

bind_cols(..., .id = NULL)
```

Arguments

`...`	For use by methods.
`.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.
`.by_group`	If TRUE, will sort first by grouping variable. Applies to grouped data frames only.
`.keep_all`	If TRUE, keep all variables in .data. If a combination of ... is not distinct, this keeps the first row of values. (See dplyr)
`.preserve`	when `FALSE` (the default), the grouping structure is recalculated based on the resulting data, otherwise it is kept as is.
`.add`	When `FALSE`, the default, `group_by()` will override existing groups. To add to the existing groups, use `.add=TRUE`. This argument was previously called `add`, but that prevented creating a new grouping variable called `add`, and conflicts with our naming conventions.
`.data`	Input data frame.
`y`	tbls to join. (See dplyr)

by	A character vector of variables to join by. (See dplyr)
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. (See dplyr)
suffix	If there are non-joined duplicate variables in x and y, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2. (See dplyr)
tbl	A data.frame.
size	<tidy-select> For sample_n(), the number of rows to select. For sample_frac(), the fraction of rows to select. If tbl is grouped, size applies to each group.
replace	Sample with or without replacement?
weight	<tidy-select> Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1.
.env	DEPRECATED.
x	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr).
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.
.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.
name	An optional parameter that specifies the column to be used as names for a named vector. Specified in a similar manner as var.

Details

Locales:

The sort order for character vectors will depend on the collating sequence of the locale in use: see locales().

Missing values:

Unlike base sorting with sort(), NA are:

- always sorted to the end for local data, even when wrapped with desc().
- treated differently for remote data, depending on the backend.

dplyr is not yet smart enough to optimise filtering optimisation on grouped datasets that don't need grouped calculations. For this reason, filtering is often considerably faster on ungroup()ed data.

rowwise() is used for the results of do() when you create list-variables. It is also useful to support arbitrary complex operations that need to be applied to each row.

Currently, rowwise grouping only works with data frames. Its main impact is to allow you to work with list-variables in summarise() and mutate() without having to use [[1]]. This makes summarise() on a rowwise tbl effectively equivalent to plyr::ldply().

Slice does not work with relational databases because they have no intrinsic notion of row order. If you want to perform the equivalent operation, use filter() and row_number().

Value

An object of the same type as `.data`.

- All rows appear in the output, but (usually) in a different place.
- Columns are not modified.
- Groups are not modified.
- Data frame attributes are preserved.

A tidySingleCellExperiment object

An object of the same type as `.data`.

- Rows are a subset of the input, but appear in the same order.
- Columns are not modified.
- The number of groups may be reduced (if `.preserve` is not `TRUE`).
- Data frame attributes are preserved.

A grouped data frame, unless the combination of `...` and `add` yields a non empty set of grouping columns, a regular (ungrouped) data frame otherwise.

An object *usually* of the same type as `.data`.

- The rows come from the underlying `group_keys()`.
- The columns are a combination of the grouping keys and the summary expressions that you provide.
- If `x` is grouped by more than one variable, the output will be another grouped_df with the right-most group removed.
- If `x` is grouped by one variable, or is not grouped, the output will be a tibble.
- Data frame attributes are **not** preserved, because `summarise()` fundamentally creates a new data frame.

An object of the same type as `.data`.

For `mutate()`:

- Rows are not affected.
- Existing columns will be preserved unless explicitly modified.
- New columns will be added to the right of existing columns.
- Columns given value `NULL` will be removed
- Groups will be recomputed if a grouping variable is mutated.
- Data frame attributes are preserved.

For `transmute()`:

- Rows are not affected.
- Apart from grouping variables, existing columns will be remove unless explicitly kept.
- Column order matches order of expressions.
- Groups will be recomputed if a grouping variable is mutated.

- Data frame attributes are preserved.

An object of the same type as `.data`.

- Rows are not affected.
- Column names are changed; column order is preserved
- Data frame attributes are preserved.
- Groups are updated to reflect new names.

A `tbl`

A `tbl`

A tidySingleCellExperiment object

A tidySingleCellExperiment object

A tidySingleCellExperiment object

A tidySingleCellExperiment object

An object of the same type as `.data`. The output has the following properties:

- Each row may appear 0, 1, or many times in the output.
- Columns are not modified.
- Groups are not modified.
- Data frame attributes are preserved.

An object of the same type as `.data`. The output has the following properties:

- Rows are not affected.
- Output columns are a subset of input columns, potentially with a different order. Columns will be renamed if `new_name=old_name` form is used.
- Data frame attributes are preserved.
- Groups are maintained; you can't select off grouping variables.

A tidySingleCellExperiment object

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

A vector the same size as `.data`.

Methods

This function is a **generic**, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

The following methods are currently available in loaded packages:

This function is a **generic**, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

The following methods are currently available in loaded packages:

These function are **generic**s, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

Methods available in currently loaded packages:

This function is a **generic**, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

The following methods are currently available in loaded packages:

These function are **generic**s, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

Methods available in currently loaded packages:

This function is a **generic**, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

The following methods are currently available in loaded packages:

These function are **generic**s, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

Methods available in currently loaded packages:

- `slice()`: no methods found.
- `slice_head()`: no methods found.
- `slice_tail()`: no methods found.
- `slice_min()`: no methods found.
- `slice_max()`: no methods found.
- `slice_sample()`: no methods found.

This function is a **generic**, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

The following methods are currently available in loaded packages: no methods found.

This function is a **generic**, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

The following methods are currently available in loaded packages: no methods found.

Useful filter functions

- `==`, `>`, `>=` etc
- `&`, `|`, `!`, `xor()`
- `is.na()`
- `between()`, `near()`

Grouped tibbles

Because filtering expressions are computed within groups, they may yield different results on grouped tibbles. This will be the case as soon as an aggregating, lagging, or ranking function is involved. Compare this ungrouped filtering:

The former keeps rows with `mass` greater than the global average whereas the latter keeps rows with `mass` greater than the gender

average.

Because mutating expressions are computed within groups, they may yield different results on grouped tibbles. This will be the case as soon as an aggregating, lagging, or ranking function is involved. Compare this ungrouped mutate:

With the grouped equivalent:

The former normalises `mass` by the global average whereas the latter normalises by the averages within gender levels.

Useful functions

- Center: `mean()`, `median()`
- Spread: `sd()`, `IQR()`, `mad()`
- Range: `min()`, `max()`, `quantile()`
- Position: `first()`, `last()`, `nth()`,
- Count: `n()`, `n_distinct()`
- Logical: `any()`, `all()`

Backend variations

The data frame backend supports creating a variable and using it in the same summary. This means that previously created summary variables can be further transformed or combined within the summary, as in `mutate()`. However, it also means that summary variables with the same names as previous variables overwrite them, making those variables unavailable to later summary variables.

This behaviour may not be supported in other backends. To avoid unexpected results, consider using new names for your summary variables, especially when creating multiple summaries.

Useful mutate functions

- `+`, `-`, `log()`, etc., for their usual mathematical meanings
- `lead()`, `lag()`
- `dense_rank()`, `min_rank()`, `percent_rank()`, `row_number()`, `cume_dist()`, `ntile()`
- `cumsum()`, `cummean()`, `cummin()`, `cummax()`, `cumany()`, `cumall()`
- `na_if()`, `coalesce()`
- `if_else()`, `recode()`, `case_when()`

Scoped selection and renaming

Use the three scoped variants (`rename_all()`, `rename_if()`, `rename_at()`) to renaming a set of variables with a function.

See Also

filter_all(), filter_if() and filter_at().

Examples

```
`%>%` <- magrittr::`%>%`
pbmc_small %>%

    arrange(nFeature_RNA)

`%>%` <- magrittr::`%>%`
pbmc_small %>%

    distinct(groups)


`%>%` <- magrittr::`%>%`
pbmc_small %>%

    filter(groups == "g1")

# Learn more in ?dplyr_tidy_eval

`%>%` <- magrittr::`%>%`
pbmc_small %>%

    group_by(groups)

`%>%` <- magrittr::`%>%`
pbmc_small %>%

    summarise(mean(nCount_RNA))

`%>%` <- magrittr::`%>%`
pbmc_small %>%

    mutate(nFeature_RNA=1)

`%>%` <- magrittr::`%>%`
pbmc_small %>%

    rename(s_score=nFeature_RNA)

`%>%` <- magrittr::`%>%`

`%>%` <- magrittr::`%>%`

tt <- pbmc_small
tt %>% left_join(tt %>% distinct(groups) %>% mutate(new_column=1:2))
`%>%` <- magrittr::`%>%`

tt <- pbmc_small
```

```
tt %>% inner_join(tt %>% distinct(groups) %>% mutate(new_column=1:2) %>% slice(1))

`%>%` <- magrittr::`%>%`

tt <- pbmc_small
tt %>% right_join(tt %>% distinct(groups) %>% mutate(new_column=1:2) %>% slice(1))

`%>%` <- magrittr::`%>%`

tt <- pbmc_small
tt %>% full_join(tibble::tibble(groups="g1", other=1:4))


`%>%` <- magrittr::`%>%`
pbmc_small %>%

    slice(1)

`%>%` <- magrittr::`%>%`
pbmc_small %>%

    select(cell, orig.ident)

`%>%` <- magrittr::`%>%`
pbmc_small %>%

    sample_n(50)
pbmc_small %>%

    sample_frac(0.1)


`%>%` <- magrittr::`%>%`
pbmc_small %>%

    count(groups)

`%>%` <- magrittr::`%>%`
pbmc_small %>%

    pull(groups)
```

`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

drop_class *Remove class to abject*

Description

Remove class to abject

Usage

```
drop_class(var, name)
```

Arguments

var	A tibble
name	A character name of the class

Value

A tibble with an additional attribute

extract *Extract a character column into multiple columns using regular expression groups*

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.

Usage

```
## S3 method for class 'SingleCellExperiment'
extract(
  data,
  col,
  into,
  regex = "([[:alnum:]]+)",
  remove = TRUE,
  convert = FALSE,
  ...
)
```

Arguments

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

See Also

separate() to split up by a separator.

Examples

```
pbmc_small %>%

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

---

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 *Extract and join information for features.*

Description

join_features() extracts and joins information for specified features

Usage

```
## S4 method for signature 'SingleCellExperiment'
join_features(
  .data,
  features = NULL,
  all = FALSE,
  exclude_zeros = FALSE,
  shape = "long",
  ...
)
```

Arguments

.data	A 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

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

Value

An object containing the information.for the specified features

An object containing the information.for the specified features

Examples

```
data("pbmc_small")
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 tidySingleCellExperiment 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")
```

---

nest *nest*

Description

nest

Arguments

.data	A tbl. (See tidyr)
...	Name-variable pairs of the form new_col=c(col1, col2, col3) (See tidyr)
.names_sep	See ?tidyr::nest

Value

A tidySingleCellExperiment objector a tibble depending on input

Examples

```
library(dplyr)
pbmc_small %>%

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

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.
`cols_vary`	When pivoting `cols` into longer format, how should the output rows be arranged relative to their original row number? • `"fastest"`, the default, keeps individual rows from `cols` close together in the output. This often produces intuitively ordered output when you have at least one key column from `data` that is not involved in the pivoting process. • `"slowest"` keeps individual columns from `cols` close together in the output. This often produces intuitively ordered output when you utilize all of the columns from `data` in the pivoting process.
`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_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 `NA`s 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.

`...` 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.

Examples

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

quo_names — *Convert array of quosure (e.g. c(col_a, col_b)) into character vector*

Description

Convert array of quosure (e.g. c(col_a, col_b)) into character vector

Usage

```
quo_names(v)
```

Arguments

v	A array of quosures (e.g. c(col_a, col_b))

Value

A character vector

separate — *Separate a character column into multiple columns with a regular expression or numeric locations*

Description

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

Arguments

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

See Also

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

Examples

```
un <- pbmc_small %>%

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

---

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()` for your class which is called by the default method.

---

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.

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
sep	Separator to use between values.
na.rm	If TRUE, missing values will be remove prior to uniting each value.
remove	If TRUE, remove input columns from output data frame.

Value

A tidySingleCellExperiment objector a tibble depending on input

See Also

separate(), the complement.

Examples

```
pbmc_small %>%

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

unnest *unnest*

Description

unnest

unnest_single_cell_experiment

Usage

```
## S3 method for class 'tidySingleCellExperiment_nested'
unnest(
  data,
  cols,
  ...,
  keep_empty = FALSE,
  ptype = NULL,
  names_sep = NULL,
  names_repair = "check_unique",
  .drop,
  .id,
  .sep,
  .preserve
)

unnest_single_cell_experiment(
  data,
  cols,
  ...,
  keep_empty = FALSE,
  ptype = NULL,
  names_sep = NULL,
  names_repair = "check_unique",
  .drop,
  .id,
  .sep,
  .preserve
)
```

Arguments

data	A tbl. (See tidyr)
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).

...	<tidy-select> Columns to nest, specified using name-variable pairs of the form new_col=c(col1, col2, col3). The right hand side can be any valid tidy select expression. **[Deprecated]**: previously you could write df %>% nest(x, y, z) and df %>% unnest(x, y, z). Convert to df %>% nest(data=c(x, y, z)). and df %>% unnest(c(x, y, z)). If you previously created new variable in unnest() you'll now need to do it explicitly with mutate(). Convert df %>% unnest(y=fun(x, y, z)) to df %>% mutate(y=fun(x, y, z)) %>% unnest(y).
keep_empty	See tidyr::unnest
ptype	See tidyr::unnest
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.
names_repair	See tidyr::unnest
.drop	See tidyr::unnest
.id	tidyr::unnest
.sep	tidyr::unnest
.preserve	See tidyr::unnest
sep	tidyr::unnest

Value

A tidySingleCellExperiment objector a tibble depending on input

A tidySingleCellExperiment objector a tibble depending on input

Examples

```
library(dplyr)
pbmc_small %>%

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


library(dplyr)
pbmc_small %>%

    nest(data=-groups) %>%
    unnest_single_cell_experiment(data)
```

`%>%` *Pipe operator*

Description

See `magrittr::%>%` for details.

Usage

```
lhs %>% rhs
```

Arguments

`lhs`	A value or the magrittr placeholder.
`rhs`	A function call using the magrittr semantics.

Value

The result of calling `rhs(lhs)`.

Index