

Package: ttservice (via r-universe)

November 2, 2024

Type Package

Title A Service for Tidy Transcriptomics Software Suite

Version 0.4.1

Description It provides generic methods that are used by more than one package, avoiding conflicts. This package will be imported by 'tidySingleCellExperiment' and 'tidyseurat'.

License GPL-3

Depends R (>= 4.0.0)

Imports dplyr, Matrix, plotly

Suggests methods

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Repository <https://stemangiola.r-universe.dev>

RemoteUrl <https://github.com/stemangiola/ttservice>

RemoteRef HEAD

RemoteSha b5138f6af3324d4192b51e6d142bf8a24a7eb4d0

Contents

aggregate_cells	2
bind_rows	3
join_features	4
plot_ly	5

Index	9
--------------	----------

aggregate_cells	<i>Aggregate cells</i>
-----------------	------------------------

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 = Matrix::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
...	Used for future extensibility

Value

A tibble object

Examples

```
print("pbmc_small |> aggregate_cells(c(groups, ident), assays = \"counts\")")
```

bind_rows

*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.

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.

Usage

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

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

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 <code>mutate_joins</code> .
.id	Data frame identifier. When <code>'id'</code> 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 <code>'bind_rows()'</code> . 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 Seurat 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.

The output of `'bind_cols()'` 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'`.

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

Examples

```
print("small_pbmc |> bind_rows(small_pbmc)")
```

```
print("small_pbmc |> bind_cols(annotation_column)")
```

join_features	<i>join_features</i>
---------------	----------------------

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

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

```
print("this is a method generics Example is not applicable")
# <object> |> join_features(features=c("HLA-DRA", "LYZ"))
```

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 <code>schema()</code> for a list of acceptable attributes for a given trace type (by going to <code>traces -> type -> attributes</code>). Note that attributes provided at this level may override other arguments (e.g. <code>plot_ly(x = 1:10, y = 1:10, color = I("red"), marker = list(color = "blue"))</code>).
type	A character string specifying the trace type (e.g. "scatter", "bar", "box", etc). If specified, it <i>always</i> 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 <code>split</code> in that it creates one trace for every unique value.
color	Values mapped to relevant 'fill-color' attribute(s) (e.g. <code>fillcolor</code> , <code>marker.color</code> , <code>textfont.color</code> , etc.). The mapping from data values to color codes may be controlled using <code>colors</code> and <code>alpha</code> , or avoided altogether via <code>I()</code> (e.g., <code>color = I("red")</code>). Any color understood by <code>grDevices::col2rgb()</code> may be used in this way.
colors	Either a <code>colorbrewer2.org</code> palette name (e.g. "YlOrRd" or "Blues"), or a vector of colors to interpolate in hexadecimal "#RRGGBB" format, or a color interpolation function like <code>colorRamp()</code> .
alpha	A number between 0 and 1 specifying the alpha channel applied to color. Defaults to 0.5 when mapping to <code>fillcolor</code> and 1 otherwise.
stroke	Similar to <code>color</code> , but values are mapped to relevant 'stroke-color' attribute(s) (e.g., <code>marker.line.color</code> and <code>line.color</code> for filled polygons). If not specified, <code>stroke</code> inherits from <code>color</code> .
strokes	Similar to <code>colors</code> , but controls the stroke mapping.
alpha_stroke	Similar to <code>alpha</code> , but applied to <code>stroke</code> .
size	(Numeric) values mapped to relevant 'fill-size' attribute(s) (e.g., <code>marker.size</code> , <code>textfont.size</code> , and <code>error_x.width</code>). The mapping from data values to symbols may be controlled using <code>sizes</code> , or avoided altogether via <code>I()</code> (e.g., <code>size = I(30)</code>).
sizes	A numeric vector of length 2 used to scale <code>size</code> to pixels.
span	(Numeric) values mapped to relevant 'stroke-size' attribute(s) (e.g., <code>marker.line.width</code> , <code>line.width</code> for filled polygons, and <code>error_x.thickness</code>) The mapping from data values to symbols may be controlled using <code>spans</code> , or avoided altogether via <code>I()</code> (e.g., <code>span = I(30)</code>).
spans	A numeric vector of length 2 used to scale <code>span</code> to pixels.
symbol	(Discrete) values mapped to <code>marker.symbol</code> . The mapping from data values to symbols may be controlled using <code>symbols</code> , or avoided altogether via <code>I()</code> (e.g., <code>symbol = I("pentagon")</code>). Any <code>pch</code> value or <code>symbol name</code> may be used in this way.
symbols	A character vector of <code>pch</code> values or <code>symbol names</code> .
linetype	(Discrete) values mapped to <code>line.dash</code> . The mapping from data values to symbols may be controlled using <code>linetypes</code> , or avoided altogether via <code>I()</code> (e.g., <code>linetype = I("dash")</code>). Any <code>lty</code> (see <code>par</code>) value or <code>dash name</code> may be used in this way.
linetypes	A character vector of <code>lty</code> values or <code>dash names</code>

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 <code>event_data()</code> 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)`)

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://plotly.com/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(palmerpenguins::penguins, x = ~bill_length_mm, y = ~body_mass_g)
add_markers(p, color = ~bill_depth_mm, size = ~bill_depth_mm)
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)
```


Index

`add_trace()`, 7
`aggregate_cells`, 2
`animation`, 5

`bind_cols (bind_rows)`, 3
`bind_rows`, 3

`crostalk::bscols()`, 7
`crostalk::SharedData`, 5

`event_data()`, 7

`formula`, 7

`ggplot2::qplot()`, 5
`ggplotly()`, 7
`grDevices::col2rgb()`, 6

`highlight()`, 7

`I()`, 6

`join_features`, 4

`layout()`, 7

`par`, 6
`pch`, 6
`plot()`, 5
`plot_geo()`, 7
`plot_ly`, 5
`plot_mapbox()`, 7
`plotly_json()`, 7

`schema()`, 6, 7
`style()`, 7
`subplot()`, 7